Classifying Animals with Backbones

Has
eel-like
shape.

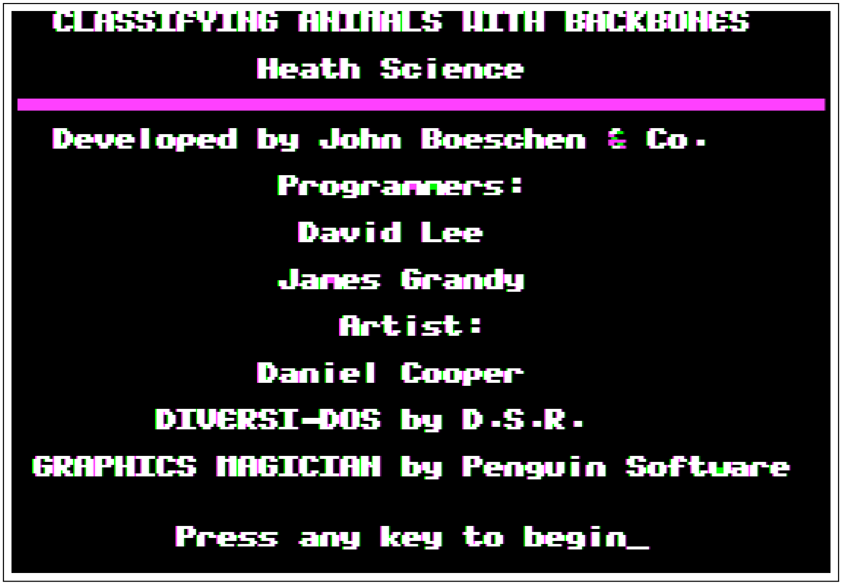
H)elp
G)lossary

Terrific! -> PRESS C
AMERICAN EEL

2015-01-30

to deprotect
and preserve

Contents



```
CLASSIFYING ANIMALS WITH BACKBONES
Heath Science

Developed by John Boeschen & Co.
Programmers:
David Lee
James Grandy
Artist:
Daniel Cooper
DIVERSI-DOS by D.S.R.
GRAPHICS MAGICIAN by Penguin Software

Press any key to begin_
```

```
---Classifying Animals with Backbones--
A 4am crack                 2015-01-30
---------------------------------------
```

Name: Classifying Animals with
  Backbones
Genre: educational
Year: 1985
Authors: John Boeschen & Co., David
  Lee, James Grandy, Daniel Cooper
  (artist)
Publisher: D.C. Heath and Company
Media: single-sided 5.25-inch floppy
OS: Diversi-DOS
Other versions: none (preserved here
  for the first time)

```
      _________________________
     {                         }
     { "Ed Gruberman, you must }
     {  learn patience."       }
     { "Yeah yeah yeah,        }
     {  patience. How long     }
     {  will that take?"       }
     {                         }
     {        Boot to the Head }
     {          The Frantics   }
     {_________________________}
```

Chapter 0
In Which Various Automated Tools Fail In Interesting Ways

COPYA
   no errors, but the copy boots DOS and
   stops

Locksmith Fast Disk Backup
   ditto

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

Copy][+ nibble editor
   nothing suspicious

Disk Fixer
   T00,S00 -> standard DOS 3.3 boot0
   T00-02 -> looks like DOS 3.3
   T01,S09 -> startup program is blank?!
   T11 -> DOS 3.3 disk catalog

Why didn't any of my copies work?
   Maybe a nibble check that sabotages
   the boot on failure?

Next steps:

   1. Capture bootloader with AUTOTRACE
   2. Find nibble check and disable it
   3. There is no step 3

Chapter 1
In Which We Have A Few False Starts, Then Our Adventure Begins In Earnest

```
[S6,D1=original 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

]CATALOG,S6,D1

C1983 DSR^C#254
014 FREE

 A 003 HELLO
 B 002 INPUT.OBJ0
 B 014 PICDRAWL
 B 010 HRCGL
 B 002 SOUND
 B 002 GLOSSARY PAGE.SPC
 T 009 ACT.1.DATA
 A 017 REVIEW RESULTS
 B 002 ACT.2.SPC
 T 003 ACT.3.CLASS STATEMENTS.DATA
 T 006 ACT.3.SPEC. STATEMENTS.DATA
 T 005 ACT.3.CLASS STATEMENTS
 T 012 ACT.3.SPECIFIC STATEMENTS
 A 019 CREDITS + NAME ENTRY
 B 002 UNPACK
 B 009 LOGO
 T 018 VERT.GLOS
 T 003 HELP SCREEN.MW
 A 013 MAIN MENU
 T 011 STUDENT ROSTER.CLASS
 [...]
 A 042 ACT.1
 A 045 ACT.2
 A 052 ACT.3
 B 005 COMMON WATER SNAKE.SPC
 B 005 GREEN TURTLE.SPC
 B 005 ALLIGATOR.SPC
 B 004 CROCODILE.SPC
 B 005 SHORT-HORNED LIZARD.SPC
 B 004 OPOSSUM.SPC
 B 004 CALIFORNIA MOLE.SPC
 B 004 BIG BROWN BAT.SPC
 B 005 SCUBA DIVER.SPC
 B 004 BEAVER.SPC
 B 004 ANTELOPE JACKRABBIT.SPC
 B 004 WHITE-TAILED DEER.SPC
 B 003 HARBOR PORPOISE.SPC
 B 004 NINE-BANDED ARMADILLO.SPC
 B 004 WHITE PELICAN.SPC
 B 004 BALD EAGLE.SPC
 B 005 TURKEY.SPC
 B 003 AMERICAN FLAMINGO.SPC
 B 004 ROCK DOVE.SPC
 B 003 ROADRUNNER.SPC
 B 004 GREAT HORNED OWL.SPC
 B 003 BELTED KING FISHER.SPC
 B 003 RED-HEADED WOODPECKER.SPC
 B 004 BLUEJAY.SPC
 B 004 ANOLE LIZARD.SPC
 B 004 GLASS SNAKE LIZARD.SPC
 B 005 SCARLET KING SNAKE.SPC
 B 004 RED SALAMANDER.SPC
 B 003 SIREN.SPC
 B 004 NEWT.SPC
 B 003 GREAT PLAINS TOAD.SPC
 B 004 SPADEFOOT TOAD.SPC
 B 005 GREEN TREE FROG.SPC
 B 004 LEOPARD FROG.SPC
 B 003 BULLFROG TADPOLE.SPC
 B 003 HAGFISH.SPC
 B 003 LAMPREY.SPC
[...]
```

```
 B 004 HAMMERHEAD SHARK.SPC
 B 003 GIANT DEVIL RAY.SPC
 B 004 RAINBOW TROUT.SPC
 B 003 BLACK BULLHEAD CATFISH.SPC
 B 004 AMERICAN EEL.SPC
 B 004 AMERICAN SEA HORSE.SPC
 B 003 ROCK.SPC
 B 004 VENUS FLYTRAP.SPC
 B 005 EARTHWORM.SPC
 B 005 PARAMECIUM.SPC
 B 004 ATLANTIC HALIBUT.SPC
 B 003 HARBOR SEAL.SPC
 B 004 SPOTTED TURTLE.SPC
```

(Note to self: there may be personally identifiable information in the STUDENT ROSTER.CLASS file. Wipe it before release.)

```
]RUN HELLO
...computer freezes...
```

Hmm.

```
]PR#5
]LOAD HELLO,S6,D1
...computer freezes...
```

Double hmm.

```
]PR#5
]TLIST ACT.1.DATA,S6,D1
ENERGY TO MOVE COMES FROM WITHIN.
NEEDS FOOD FOR GROWTH AND ENERGY.
USES OXYGEN TO GET ENERGY FROM FOOD.
MOVES TOWARD OR AWAY FROM THINGS.
REPRODUCES ITS OWN KIND.
MADE OF CELLS WITH CYTOPLASM.
GROWS BY ADDING NEW CELLS AND CYTOPLASM
...
```

Well, at least that works. But I can't
LOAD or RUN any of the BASIC programs.
Not sure why yet.

Let's go find that nibble check.

```
]BLOAD BOOT1,A$2600,S5,D1
]CALL -151
*FE89G FE93G
*B600<2600.2FFFM
*B700L
.
. nothing unusual, until...
.
B738-   20 03 BB    JSR   $BB03

*BB03L

BB03-   4E 06 BB    LSR   $BB06
```

Uh oh. Self-modifying code. The 6502
processor has no instruction cache, so
one instruction can literally change
the next instruction in memory, and the
CPU will execute the new instruction.
Which is what's happening here.

Chapter 2
In Which We Learn The True Meaning Of Patience

To capture this self-modifying code, I need to reproduce the modifications without running the modified code. I'll start with a pristine copy (at $2B00), copy it into place (at $BB00), then reproduce the modifications and inspect the results. Lather, rinse, repeat.

```
2000-   A0 00           LDY   #$00
2002-   B9 00 2B        LDA   $2B00,Y
2005-   99 00 BB        STA   $BB00,Y
2008-   C8              INY
2009-   D0 F7           BNE   $2002
200B-   4E 06 BB        LSR   $BB06
200E-   60              RTS

*2000G
*BB03L

BB03-   4E 06 BB        LSR   $BB06
BB06-   38              SEC
BB07-   6E 0A BB        ROR   $BB0A
```

More self-modifying code.

```
*200E:38 6E 0A BB 60
*2000G
*BB0AL

BB0A-   A0 27           LDY   #$27
BB0C-   6E 0F BB        ROR   $BB0F
```

More.

```
*2012:A0 27 6E 0F BB 60
*2000G
*BB0FL

BB0F-   6E 1B BB        ROR   $BB1B
BB12-   6E 15 BB        ROR   $BB15
```

More.

```
*2017:6E 1B BB 6E 15 BB 60
*2000G
*BB15L

BB15-   6E 1E BB    ROR   $BB1E
BB18-   6E 25 BB    ROR   $BB25
BB1B-   B9 00 BB    LDA   $BB00,Y
```

More.

```
*201D:6E 1E BB 6E 25 BB B9 00 BB 60
*2000G
*BB1EL

BB1E-   59 00 B8    EOR   $B800,Y
BB21-   99 00 BB    STA   $BB00,Y
BB24-   C8          INY
BB25-   D0 F4       BNE   $BB1B
```

More, now using the page at $B800 as an encryption key.

```
*2026:59 00 B8 99 00 BB C8 D0 F4 60
*2000G
*BB27L

BB27-   A0 55       LDY   #$55
BB29-   B9 00 BC    LDA   $BC00,Y
BB2C-   59 00 B8    EOR   $B800,Y
BB2F-   99 00 BC    STA   $BC00,Y
BB32-   88          DEY
BB33-   10 F4       BPL   $BB29
```

More.

```
*202F:A0 55 B9 00 BC 59 00 B8 99 00 BC
      88 10 F4 60
*2000G
```

```
*BB35L
```

Finally some real code.

```
; cover our tracks in memory (overwrite
; the call to $BB03)
BB35-   A9 93       LDA   #$93
BB37-   8D 39 B7    STA   $B739
BB3A-   A9 B7       LDA   #$B7
BB3C-   8D 3A B7    STA   $B73A

; push an address to the stack
BB3F-   A9 B5       LDA   #$B5
BB41-   48          PHA
BB42-   A9 18       LDA   #$18
BB44-   48          PHA

; save some other values on the stack
BB45-   AD EC B7    LDA   $B7EC
BB48-   48          PHA
BB49-   AD ED B7    LDA   $B7ED
BB4C-   48          PHA

; set up an RWTS read
BB4D-   A9 00       LDA   #$00
BB4F-   8D EC B7    STA   $B7EC
BB52-   A9 06       LDA   #$06
BB54-   8D ED B7    STA   $B7ED
BB57-   A9 01       LDA   #$01
BB59-   8D F4 B7    STA   $B7F4
```

```
; $BB00 is going to get overwritten by
; the RWTS (it's used as scratch space)
; so this relocates the rest of the
; copy protection routine to as-yet-
; unused memory
BB5C-   A0 00       LDY   #$00
BB5E-   B9 6A BB    LDA   $BB6A,Y
BB61-   99 00 B4    STA   $B400,Y
BB64-   C8          INY
BB65-   D0 F7       BNE   $BB5E
BB67-   4C 00 B4    JMP   $B400

*B400<BB6A.BC69M
*B400L

; call the RWTS -- at this point, $B7F1
; is $B5 so this will read T00,S06 into
; $B500
B400-   A0 E8       LDY   #$E8
B402-   A9 B7       LDA   #$B7
B404-   20 B5 B7    JSR   $B7B5
B407-   A9 00       LDA   #$00
B409-   85 48       STA   $48

; branch on successful read
B40B-   90 05       BCC   $B412

; on disk read error, pop the saved
; track/sector from the stack and jump
; to the failure path at $B4B3
B40D-   68          PLA
B40E-   68          PLA
B40F-   4C B3 B4    JMP   $B4B3
```

```
; loop and keep reading the rest of
; track 0... into the same address
B412-    AC ED B7     LDY   $B7ED
B415-    88           DEY
B416-    98           TYA
B417-    29 0F        AND   #$0F
B419-    8D ED B7     STA   $B7ED
B41C-    C9 06        CMP   #$06
B41E-    D0 E0        BNE   $B400

; turn on drive motor
B420-    BD 89 C0     LDA   $C089,X

; restore RWTS parameters from stack
B423-    68           PLA
B424-    8D ED B7     STA   $B7ED
B427-    68           PLA
B428-    8D EC B7     STA   $B7EC

; initialize zero page $FF with $04
B42B-    A0 03        LDY   #$03
B42D-    C8           INY
B42E-    98           TYA
B42F-    29 0F        AND   #$0F
B431-    85 FF        STA   $FF

; look for a sync byte
B433-    A0 05        LDY   #$05
B435-    BD 8C C0     LDA   $C08C,X
B438-    10 FB        BPL   $B435
B43A-    48           PHA
B43B-    68           PLA
B43C-    49 FF        EOR   #$FF
B43E-    D0 F3        BNE   $B433
B440-    88           DEY
B441-    D0 F2        BNE   $B435
```

```
; look for $D5
B443-    BD 8C C0      LDA   $C08C,X
B446-    10 FB         BPL   $B443
B448-    EA            NOP
B449-    EA            NOP
B44A-    C9 D5         CMP   #$D5
B44C-    D0 F5         BNE   $B443
B44E-    F0 08         BEQ   $B458

; increment $FF but wrap around at $10
; (is this a sector number?)
B450-    A4 FF         LDY   $FF
B452-    C8            INY
B453-    98            TYA
B454-    29 0F         AND   #$0F
B456-    85 FF         STA   $FF

; skip some nibbles
B458-    A0 6F         LDY   #$6F
B45A-    BD 8C C0      LDA   $C08C,X
B45D-    10 FB         BPL   $B45A
B45F-    48            PHA
B460-    68            PLA
B461-    88            DEY
B462-    D0 F6         BNE   $B45A
B464-    EA            NOP
B465-    BD 8C C0      LDA   $C08C,X
B468-    10 FB         BPL   $B465

; look up an array value, using $FF
; as the index
B46A-    A4 FF         LDY   $FF
B46C-    B9 BA B4      LDA   $B4BA,Y
B46F-    85 FE         STA   $FE
B471-    B9 CA B4      LDA   $B4CA,Y
B474-    48            PHA
```

```
; skip some more nibbles
B475-   A0 00       LDY   #$00
B477-   BD 8C C0    LDA   $C08C,X
B47A-   10 FB       BPL   $B477
B47C-   48          PHA
B47D-   68          PLA
B47E-   88          DEY
B47F-   D0 F6       BNE   $B477
B481-   68          PLA
B482-   A8          TAY

; compare the next nibble to the
; expected value (in $B4D9,Y)
B483-   BD 8C C0    LDA   $C08C,X
B486-   10 FB       BPL   $B483
B488-   C6 FE       DEC   $FE
B48A-   C8          INY
B48B-   D9 D9 B4    CMP   $B4D9,Y
B48E-   D0 1D       BNE   $B4AD
B490-   C9 FF       CMP   #$FF
B492-   D0 EF       BNE   $B483

; skip over exact number of sync bytes
B494-   BD 8C C0    LDA   $C08C,X
B497-   10 FB       BPL   $B494
B499-   C9 FF       CMP   #$FF
B49B-   D0 10       BNE   $B4AD
B49D-   C6 FE       DEC   $FE
B49F-   D0 F3       BNE   $B494
B4A1-   EA          NOP
```

```
; next nibble must be $D5, otherwise
; we branch to the failure path at
; $B4B3
B4A2-   BD 8C C0    LDA   $C08C,X
B4A5-   10 FB       BPL   $B4A2
B4A7-   C9 D5       CMP   #$D5
B4A9-   F0 A5       BEQ   $B450
B4AB-   D0 06       BNE   $B4B3
B4AD-   A5 FF       LDA   $FF
B4AF-   C9 03       CMP   #$03
B4B1-   F0 02       BEQ   $B4B5

; failure path pops $B518 off the stack
B4B3-   68          PLA
B4B4-   68          PLA
; then falls through

; success path is here -- continue with
; RWTS call
B4B5-   A0 04       LDY   #$04
B4B7-   4C 93 B7    JMP   $B793
```

If the nibble check succeeds, execution continues at $B793, then jumps to $B519 (based on the $B5/$18 pair that was manually pushed to the stack at $BB3F).

If the nibble check fails, execution continues at $B793 and returns (because $B5/$18 was popped off the stack).

The routine at $B519 is the difference between the original disk and my non-working copies. The original ran it; my copies did not.

But before I trace it, I want to save
my work to date.

```
*2B00<BB00.BCFFM
*C500G
...
]BSAVE DECRYPT BB03,A$2000,L$3E
]BSAVE BB03 DECRYPTED,A$2B00,L$200
```

Now let's see what's at $B519.

Chapter 3
You Know Nothing Of Patience, Jon Snow

```
]CALL -151
*9600<C600.C6FFM

; set up first callback after boot0
96F8-   A9 4C       LDA   #$4C
96FA-   8D 4A 08    STA   $084A
96FD-   A9 0A       LDA   #$0A
96FF-   8D 4B 08    STA   $084B
9702-   A9 97       LDA   #$97
9704-   8D 4C 08    STA   $084C

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

; callback #1 is here -- set up second
; callback just before copy protection
970A-   A9 4C       LDA   #$4C
970C-   8D 38 B7    STA   $B738
970F-   A9 1C       LDA   #$1C
9711-   8D 39 B7    STA   $B739
9714-   A9 97       LDA   #$97
9716-   8D 3A B7    STA   $B73A

; continue the boot
9719-   4C 00 B7    JMP   $B700

; callback #2 is here -- skip copy
; protection and call RWTS directly
; to load the rest of DOS
971C-   20 93 B7    JSR   $B793
```

```
; relocate DOS to graphics page so it
; will survive a reboot
971F-   A2 23       LDX   #$23
9721-   A0 00       LDY   #$00
9723-   B9 00 9D    LDA   $9D00,Y
9726-   99 00 2D    STA   $2D00,Y
9729-   C8          INY
972A-   D0 F7       BNE   $9723
972C-   EE 25 97    INC   $9725
972F-   EE 28 97    INC   $9728
9732-   CA          DEX
9733-   D0 EE       BNE   $9723

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

*BSAVE TRACE2,A$9600,L$138
*9600G
...reboots slot 6...
...reboots slot 5...
]BSAVE BOOT2,A$2D00,L$2300
]CALL -151
*FE89G FE93G        ; disconnect old DOS
*9D00<2D00.4FFFM    ; move DOS into place
*B519L

B519-   98          TYA
B51A-   4E 1D B5    LSR   $B51D
```

Oh no. Here we go again.

```
2000-   A0 00       LDY   #$00
2002-   B9 00 45    LDA   $4500,Y
2005-   99 00 B5    STA   $B500,Y
2008-   C8          INY
2009-   D0 F7       BNE   $2002
200B-   98          TYA
200C-   4E 1D B5    LSR   $B51D
200F-   60          RTS
```

```
*2000G
*B51DL

B51D-   38          SEC
B51E-   6E 21 B5    ROR   $B521

*200F:38 6E 21 B5 60
*2000G
*B521L

B521-   A0 3E       LDY   #$3E
B523-   6E 26 B5    ROR   $B526

*2013:A0 3E 6E 26 B5 60
*2000G
*B526L

B526-   6E 32 B5    ROR   $B532
B529-   6E 2C B5    ROR   $B52C

*2018:6E 32 B5 6E 2C B5 60
*2000G
*B52CL

B52C-   6E 35 B5    ROR   $B535
B52F-   6E 3C B5    ROR   $B53C
B532-   B9 00 B5    LDA   $B500,Y

*201E:6E 35 B5 6E 3C B5 B9 0 B5 60
*2000G
*B535L

B535-   59 00 B8    EOR   $B800,Y
B538-   99 00 B5    STA   $B500,Y
B53B-   C8          INY
B53C-   D0 F4       BNE   $B532

*2027:59 0 B8 99 0 B5 C8 D0 F4 60
*2000G
```

```
*B53EL

B53E-   A0 1A       LDY   #$1A
B540-   88          DEY
B541-   B9 00 B5    LDA   $B500,Y
B544-   59 00 B8    EOR   $B800,Y
B547-   99 00 B5    STA   $B500,Y
B54A-   88          DEY
B54B-   10 F4       BPL   $B541
B54D-   A0 00       LDY   #$00
B54F-   B9 00 B4    LDA   $B400,Y
B552-   59 00 B8    EOR   $B800,Y
B555-   99 00 B4    STA   $B400,Y
B558-   88          DEY
B559-   D0 F4       BNE   $B54F

*2030<B53E.B55AM
*204D:60
*2000G
*B55BL
```

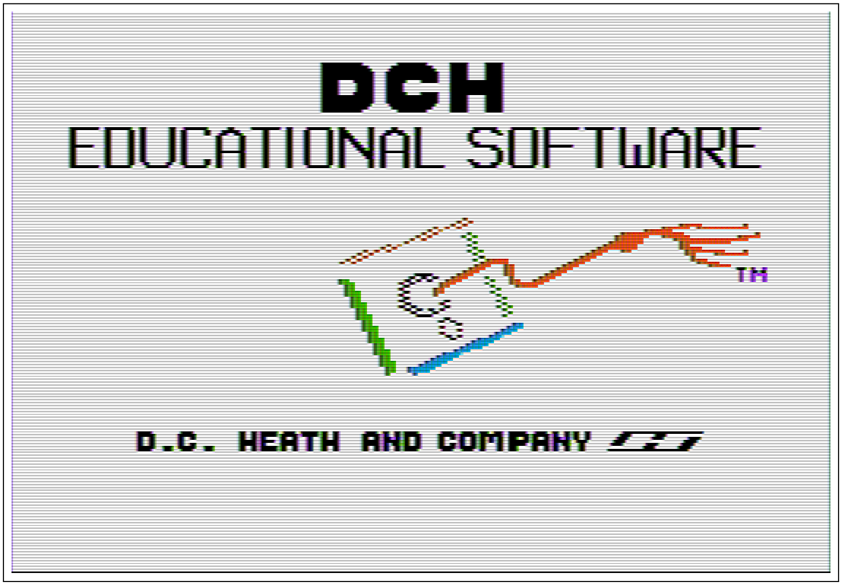
Finally some real code.

```
; set reset vector
B55B-   A9 4B       LDA   #$4B
B55D-   8D F2 03    STA   $03F2
B560-   A9 B7       LDA   #$B7
B562-   8D F3 03    STA   $03F3
B565-   49 A5       EOR   #$A5
B567-   8D F4 03    STA   $03F4
```

```
; a loop to change various memory
; locations, based on a list of patches
; at $B400
B56A-   A9 00        LDA   #$00
B56C-   85 00        STA   $00
B56E-   A9 B4        LDA   #$B4
B570-   85 01        STA   $01
B572-   A0 00        LDY   #$00
B574-   F0 0B        BEQ   $B581
B576-   B1 00        LDA   ($00),Y
B578-   99 FF FF     STA   $FFFF,Y
B57B-   88           DEY
B57C-   D0 F8        BNE   $B576
B57E-   A4 02        LDY   $02
B580-   C8           INY
B581-   B1 00        LDA   ($00),Y
B583-   F0 1C        BEQ   $B5A1
B585-   85 02        STA   $02
B587-   C8           INY
B588-   B1 00        LDA   ($00),Y
B58A-   8D 79 B5     STA   $B579
B58D-   C8           INY
B58E-   B1 00        LDA   ($00),Y
B590-   8D 7A B5     STA   $B57A
B593-   98           TYA
B594-   18           CLC
B595-   A4 02        LDY   $02
B597-   65 00        ADC   $00
B599-   85 00        STA   $00
B59B-   90 D9        BCC   $B576
B59D-   E6 01        INC   $01
B59F-   D0 D5        BNE   $B576
B5A1-   60           RTS
```

Each patch is a variable-length record, starting at $B400 (pointed to by ($00)) and continuing until the first byte of the record is $00 (compared at $B583). The general format of each record is

```
+0   [1 byte]       length of data, or $00
+1   [2 bytes]      starting address (-1)
+3   [variable]     data to write in memory
```



Once decrypted, the raw patch records look like this:

```
*B400.B519

B400- 1E 74 AA C8 C5 CC CC CF
B408- A0 A0 A0 A0 A0 A0 A0 A0
B410- A0 A0 A0 A0 A0 A0 A0 A0
B418- A0 A0 A0 A0 A0 A0 A0 A0
B420- A0 01 26 A4 A1 21 4C A4
B428- A5 68 48 38 A5 AF E5 67
B430- A8 A5 B0 E5 68 AA E8 65
B438- 68 85 68 C6 68 20 BC A3
B440- CA D0 F8 68 85 68 6C 60
B448- 9D 12 BB A3 98 49 AA 51
B450- 67 91 67 88 C0 FF D0 F4
B458- 60 A9 01 20 B1 A4 13 2F
B460- 9E A9 80 85 D6 30 0B AD
B468- 00 C0 C9 83 F0 F9 4C D2
B470- D7 EA A9 06 03 02 A5 4C
B478- 36 9E 30 49 B7 60 A0 20
B480- B9 59 B7 99 00 03 88 10
B488- F7 4C 00 03 A9 BF 85 01
B490- A0 00 84 00 91 00 C8 D0
B498- FB C6 01 A5 01 C9 08 B0
B4A0- F3 AD 81 C0 20 93 FE 20
B4A8- 89 FE 4C 00 E0 01 C1 B7
B4B0- 60 03 72 9E 4B B7 12 02
B4B8- 96 A3 18 60 03 8A A3 4C
B4C0- 82 A5 32 7E A5 4C 84 9D
B4C8- 20 71 A4 A5 68 48 A5 67
B4D0- 48 38 AE 61 AA AC 60 AA
B4D8- D0 01 CA 88 8A E8 6D 73
B4E0- AA 85 68 AD 72 AA 85 67
B4E8- C6 68 20 BC A3 CA D0 F8
B4F0- 68 85 67 68 85 68 60 02
B4F8- 51 A3 9A A3 02 5A A3 A6
B500- A3 15 96 A3 EA 18 60 8D
B508- 61 AA 8C 60 AA 20 E0 A3
B510- 4C 85 A5 20 FF A3 4C 85
B518- A5 00
```

Which translates to a series of patches all throughout DOS:

Location	Description	Value
$B400	length of data	$1E
$B401/$B402	starting address	$AA74
$B403..$B420	data	

The first patch sets the name of the startup program, which is blank on disk (T01,S09) but is now patched in memory (at $AA75) as "HELLO".

The second record follows immediately; there is no record separator.

Location	Description	Value
$B421	length of data	$01
$B422/$B423	starting address	$A426
$B424..$B424	data	$A1

The second patch munges one branch instruction in the middle of the LOAD command handler at $A413 (c.f. "Beneath Apple DOS" p. 8-12).

Location	Description	Value
$B425	length of data	$21
$B426/$B427	starting address	$A44C
$B428..$B448	data	

The $21 bytes from $B428..$B448 end up
at $A44D, and they look like this:

```
A44D-   A5 68       LDA   $68
A44F-   48          PHA
A450-   38          SEC
A451-   A5 AF       LDA   $AF
A453-   E5 67       SBC   $67
A455-   A8          TAY
A456-   A5 80       LDA   $80
A458-   E5 68       SBC   $68
A45A-   AA          TAX
A45B-   E8          INX
A45C-   65 68       ADC   $68
A45E-   85 68       STA   $68
A460-   C6 68       DEC   $68
A462-   20 BC A3    JSR   $A3BC
A465-   CA          DEX
A466-   D0 F8       BNE   $A460
A468-   68          PLA
A469-   85 68       STA   $68
A46B-   6C 60 9D    JMP   ($9D60)
```

This is changing the behavior of the
LOAD command for loading Applesoft
BASIC programs into memory. It extends
past $A450, which is normally the part
of DOS that handles loading Integer
BASIC programs. It also adds a call to
$A3BC, which is normally a test for
Integer BASIC, but which I'm guessing
is about to get overwritten in a later
patch.

Location	Description	Value
$B449	length of data	$12
$B44A/$B44B	starting address	$A3BB
$B44C..$B45D	data	

The $12 bytes from $B44C..$B45D end up at $A3BC, and they look like this:

```
A3BC-   98             TYA
A3BD-   49 AA          EOR   #$AA
A3BF-   51 67          EOR   ($67),Y
A3C1-   91 67          STA   ($67),Y
A3C3-   88             DEY
A3C4-   C0 FF          CPY   #$FF
A3C6-   D0 F4          BNE   $A3BC
A3C8-   60             RTS
A3C9-   A9 01          LDA   #$01
A3CB-   20 B1 A4       JSR   $A4B1
```

This is an on-the-fly decryption that occurs as Applesoft BASIC programs are loaded. ($67) points to the BASIC program in memory. This explains why I couldn't LOAD or RUN any of the BASIC programs on this disk when booting from my work disk: the files themselves are encrypted.

Location	Description	Value
$B45E	length of data	$13
$B45F/$B460	starting address	$9E2F
$B461..$B473	data	

The $12 bytes from $B461..$B473 end up
at $9E30 (part of the late-stage boot),
and they look like this:

```
9E30-   A9 80          LDA   #$80
9E32-   85 D6          STA   $D6
9E34-   30 0B          BMI   $9E41
9E36-   AD 00 C0       LDA   $C000
9E39-   C9 83          CMP   #$83
9E3B-   F0 F9          BEQ   $9E36
9E3D-   4C D2 D7       JMP   $D7D2
9E40-   EA             NOP
9E41-   A9 06          LDA   #$06
```

This part of late-stage boot usually
sets the reset vector to something
useful. Instead, this patch will set
the Applesoft RUN flag (zero page $D6),
which makes any command typed from the
BASIC prompt RUN the current program in
memory instead. The rest of the new
code (at $9E36) checks for <Ctrl-C> and
hangs until you press something else.
That part is skipped for now, but I'm
guessing it's called later.

Location	Description	Value
$B474	length of data	$03
$B475/$B476	starting address	$A502
$B477..$B479	data	

The 3 bytes at $B477 end up at $A503, which is the tail end of the RUN entry point. It's just a JMP to the code that was just patched earlier:

```
A503-   4C 36 9E    JMP   $9E36
```

Thus, trying to <Ctrl-C> break to the prompt during boot will hang until you press something else. (Even if you did manage to get to the prompt, the RUN flag would ensure you couldn't do anything useful. Defense in depth!)

Location	Description	Value
$B47A	length of data	$30
$B47B/$B47C	starting address	$B749
$B47D..$B4AC	data	

The $30 bytes at $B47D end up at $B74A, which is normally the part of the disk initialization routine that writes DOS to a freshly initialized disk. The new code looks like this:

```
B74A-   60              RTS
B74B-   A0 20           LDY   #$20
B74D-   B9 59 B7        LDA   $B759,Y
B750-   99 00 03        STA   $0300,Y
B753-   88              DEY
B754-   10 F7           BPL   $B74D
B756-   4C 00 03        JMP   $0300
B759-   A9 BF           LDA   #$BF
B75B-   85 01           STA   $01
B75D-   A0 00           LDY   #$00
B75F-   84 00           STY   $00
B761-   91 00           STA   ($00),Y
B763-   C8              INY
B764-   D0 FB           BNE   $B761
B766-   C6 01           DEC   $01
B768-   A5 01           LDA   $01
B76A-   C9 08           CMP   #$08
B76C-   B0 F3           BCS   $B761
B76E-   AD 81 C0        LDA   $C081
B771-   20 93 FE        JSR   $FE93
B774-   20 89 FE        JSR   $FE89
B777-   4C 00 E0        JMP   $E000
```

Looks like this is going to be The Badlands routine that wipes main memory and exits.

Location	Description	Value
$B4AD	length of data	$01
$B4AE/$B4AF	starting address	$B7C1
$B4B0	data	$60

This puts an RTS instruction at $B7C2, which would normally set up the RWTS parameters for writing DOS after INIT.

Location	Description	Value
$B4B1	length of data	$03
$B4B2/$B4B3	starting address	$9E72
$B4B4..$B4B6	data	

This modifies DOS's image of the page 3 jump vectors so that <Ctrl-Reset> will jump to $B74B, a.k.a. The Badlands.

Location	Description	Value
$B4B7	length of data	$02
$B4B8/$B4B9	starting address	$A396
$B4BA..$B4BB	data	18 60

This patch neutralizes the SAVE handler at $A397 so it does nothing but claims to have succeeded.

Location	Description	Value
$B4BC	length of data	$03
$B4BD/$B4BE	starting address	$A38A
$B4BF..$B4C1	data	

This patch adds a "JMP $A582" to the end of the BLOAD command handler that starts at $A35D.

Location	Description	Value
$B4C2	length of data	$32
$B4C3/$B4C4	starting address	$A57E
$B4C5..$B4F6	data	

The $32 bytes at $B4C5 end up at $A57F,
where they look like this:

```
A57F-   4C 84 9D    JMP   $9D84
A582-   20 71 A4    JSR   $A471
A585-   A5 68       LDA   $68
A587-   48          PHA
A588-   A5 67       LDA   $67
A58A-   48          PHA
A58B-   38          SEC
A58C-   AE 61 AA    LDX   $AA61
A58F-   AC 60 AA    LDY   $AA60
A592-   D0 01       BNE   $A595
A594-   CA          DEX
A595-   88          DEY
A596-   8A          TXA
A597-   E8          INX
A598-   6D 73 AA    ADC   $AA73
A59B-   85 68       STA   $68
A59D-   AD 72 AA    LDA   $AA72
A5A0-   85 67       STA   $67
A5A2-   C6 68       DEC   $68
A5A4-   20 BC A3    JSR   $A3BC
A5A7-   CA          DEX
A5A8-   D0 F8       BNE   $A5A2
A5AA-   68          PLA
A5AB-   85 67       STA   $67
A5AD-   68          PLA
A5AE-   85 68       STA   $68
A5B0-   60          RTS
```

The previous patch set up a jump to
$A582 at the end of the BLOAD handler.
It looks like this is reusing the
on-the-fly decryption routine at $A3BC
(already used for Applesoft programs)
for binary programs as well. Encrypt
all the things!

Location	Description	Value
$B4F7	length of data	$02
$B4F8/$B4F9	starting address	$A351
$B4FA..$B4FB	data	9A A3

This sets up a jump to $A39A in the
middle of the BSAVE command handler.

Location	Description	Value
$B4FC	length of data	$02
$B4FD/$B4FE	starting address	$A35A
$B4FF..$B500	data	A6 A3

This sets up a jump to $A3A6 at the end
of the BSAVE command handler.

Location	Description	Value
$B501	length of data	$15
$B502/$B503	starting address	$A396
$B504..$B518	data	

The $15 bytes at $B504 end up at $A397,
overwriting the SAVE command handler.
They look like this:

```
A397-   EA          NOP
A398-   18          CLC
A399-   60          RTS
A39A-   8D 61 AA    STA   $AA61
A39D-   8C 60 AA    STY   $AA60
A3A0-   20 E0 A3    JSR   $A3E0
A3A3-   4C 85 A5    JMP   $A585
A3A6-   20 FF A3    JSR   $A3FF
A3A9-   4C 85 A5    JMP   $A585
```

It looks like this *encrypts* binary
files on-the-fly. One branch of the
BSAVE handler jumps to $A39A; the other
jumps to $A3A6. The latter routes the
data in memory through the routine at
$A3FF, which serves as both an
encryption and decryption routine (it's
just XOR after all).

That's it. The next byte is $00, so the
BEQ at $B583 branches and the patch loo
exits gracefully via RTS.

The result is a really messed up DOS
that is maximally unfriendly to prying
eyes and maximally incompatible with
any other version of DOS. It decrypts
both BASIC and binary files on the fly,
traps <Ctrl-Reset>, traps <Ctrl-C>,
sets the RUN flag, and disables the
SAVE command.

It does not, however, hinder copying
the disk itself. The only patch I need
to bypass the copy protection is at
$BB03, to unconditionally push $B5/$18
to the stack and jump to $B793.

```
BB03-   A9 B5         LDA   #$B5
BB05-   48            PHA
BB06-   A9 18         LDA   #$18
BB08-   48            PHA
BB09-   4C 93 B7      JMP   $B793
```

```
T00,S05,$03
  change "4E 06 BB 71 6E 0A BB 40 27"
      to "A9 B5 48 A9 18 48 4C 93 B7"
```

Quod erat liberandum.