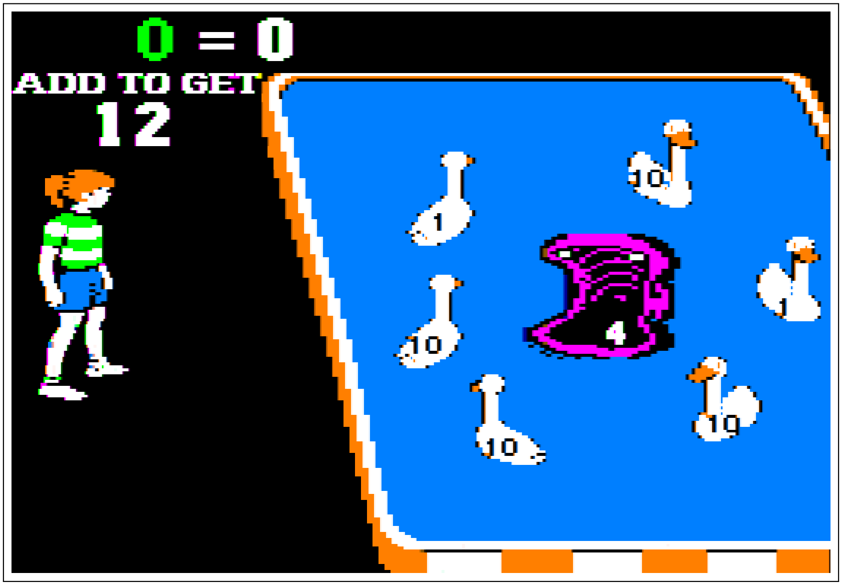
Sum Ducks



2015-10-16

4am
to deprotect
and preserve

Contents

STEPPING STONE SOFTWARE
SUM DUCKS
special thanks to Bill Budge for his encouragement
©1984 SPINNAKER SOFTWARE

```
---------------Sum Ducks----------------
A 4am quack                   2015-10-16
----------------------------------------

Name: Sum Ducks
Genre: educational
Year: 1984
Credits:
  Design: Barbara Jasinki, Diane Downie
  Software engineer: Mark Ravitz
  Programming: Bryan Moss
  Graphics: Marge Boots
Publisher: Spinnaker Software
Media: single-sided 5.25-inch floppy
OS: DOS 3.3
Previous cracks: Asimov has a crack by
  "BH", but it's corrupted
```

Chapter 0
In Which Various Automated Tools Fail In Ways Most Fowl

COPYA
  read error on second pass

Locksmith Fast Disk Backup
  unable to read track $09; copy hangs
  with drive motor on

EDD 4 bit copy (no sync, no count)
  no errors, but copy displays an error
  "THIS IS A DEFECTIVE DISK" and exits

Copy][+ nibble editor
  T09 is almost entirely sync bytes

Disk Fixer
  T00 looks like a DOS 3.3 boot0/boot1
  T00-T02 is a full copy of DOS 3.3
  T11 has a standard disk catalog
  T09 is unreadable

Why didn't any of my copies work?
  A nibble check on boot? Disks do not
  declare themselves defective unless
  someone tells them to.

Next steps:

  1. Trace the boot
  2. ???

Chapter 1
In Which Our Tools Do Not Save Us

```
[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
```

Well that's not a surprise; most of the
disk was readable, except track $09.

Hey wait, the disk is mostly readable.
Maybe I can just run it from my work
disk?

```
]CATALOG,S6,D1

C1983 DSR^C#254
052 FREE

 A 002 HELLO
 B 012 XMOD
 B 007 M
 B 002 SCREEN
 B 013 LOADER
 B 003 B0
 B 034 SCREEN2
 B 034 SL
 B 047 BLOCK2
 B 023 TITLE
 B 030 S2
 B 044 S3
 B 012 LZCD
 B 003 S1
 B 002 FONT
 B 034 LAYOUT
 B 003 STARTUP
 B 064 MAIN
 B 034 RULES
 B 016 LZMSK
 B 016 LZMAP
 B 009 UI.OBJ2

]RUN HELLO

?SYNTAX ERROR IN 776
```

```
]LIST

 776 @! FRE  TAN  =# READ = RUN 5
     TV[_USY?sto`pel{-j'$1 GR wzy
     a|rpxmqcB!LF h mqiqt~i}`jd SIN
     "CSK(99#N-@]3WFEFGZyz{v
```

Well, that is a syntax error. ("You are technically correct, the best kind of correct!")

```
]BRUN STARTUP
B33E-    A=0F X=FF Y=25 P=31 S=EF

*BRUN MAIN
74FD-    A=00 X=B0 Y=25 P=33 S=E5

*BRUN RULES
2006-    A=20 X=FF Y=25 P=39 S=DD
```

This is not a very fruitful path of investigation. Let's start over.

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

; move most of bootloader into place,
; except $BF00 (used by Diversi-DOS 64K
; on my work disk) -- so I can look at
; the code in its proper location but
; still load and save files as needed
*B600<2600.2EFFM
```

```
*B700L
.
. bog standard, until...
.
B738-   4C 03 BB    JMP   $BB03
```

Well that's definitely not normal. On a DOS 3.3 disk, there isn't usually anything in $BBxx at all. (It's used for scratch space during sector reads.)

```
*BB03L

BB03-   4E 06 BB    LSR   $BB06
BB06-   71 6E       ADC   ($6E),Y
BB08-   0A          ASL
BB09-   BB          ???
BB0A-   40          RTI
BB0B-   27          ???
```

Oh look, self-modifying code. This should be fun(*).

I'm going to make a new program that reproduces the self-modifications of the original routine at $BB03. When I'm done, I'll have

- a repeatable decryption routine, and
- complete documentation

Here we go.

(*) not guaranteed, actual fun
    may vary

Chapter 2
In Which We Painstakingly Create A Repeatable Decryption Routine, And It Stakes About As Much Pain As We Expected

The start of my self-decryption
replication program:

```
; copy $BB00 page into place from a
; pristine copy in lower memory (loaded
; as part of the BLOAD BOOT1,A$2600)
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-   60             RTS

; add the "LSR" instruction from $BB03,
; followed by an "RTS"
*200B:4E 06 BB 60

; execute it and look at the result
*2000G
*BB06L

BB06-   38             SEC
BB07-   6E 0A BB       ROR   $BB0A
```

Oh look, more self-modifying code.

```
; add these 2 instructions, followed
; by an "RTS"
*200E:38 6E 0A BB 60
*2000G

*BB0AL

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

Oh look, more...

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

*BB0FL

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

Oh look...

```
*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
```

Oh...

```
*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
```

Kill me.

Also, I need another part of boot1 in place before this will work.

```
*B800<2800.28FFM
```

Now to reproduce the code properly.

```
*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
```

Kill me now.

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

*BB35L
```

(Finally, a block of real code that does more than just decrypt the next block!)

```
; change the JMP that brought us here
BB35-    A9 E0          LDA   #$E0
BB37-    8D 3A B7       STA   $B73A

; sets an unfriendly reset vector
BB3A-    20 C3 B7       JSR   $B7C3
```

```
; save some addresses on the stack
BB3D-   AD EC B7    LDA   $B7EC
BB40-   48          PHA
BB41-   AD F4 B7    LDA   $B7F4
BB44-   48          PHA
BB45-   AD 4D BE    LDA   $BE4D
BB48-   48          PHA

; set up to seek to seek to track $09
; (the unreadable track)
BB49-   A9 09       LDA   #$09
BB4B-   8D EC B7    STA   $B7EC
BB4E-   A9 00       LDA   #$00
BB50-   8D F4 B7    STA   $B7F4

; disable the instruction that turns
; off the drive motor at the very end
; of an RWTS call
BB53-   A9 60       LDA   #$60
BB55-   8D 4D BE    STA   $BE4D

; seek to track $09 (and leave the
; motor running)
BB58-   A0 E8       LDY   #$E8
BB5A-   A9 B7       LDA   #$B7
BB5C-   20 B5 B7    JSR   $B7B5

; restore everything
BB5F-   68          PLA
BB60-   8D 4D BE    STA   $BE4D
BB63-   68          PLA
BB64-   8D F4 B7    STA   $B7F4
BB67-   68          PLA
BB68-   8D EC B7    STA   $B7EC
```

```
; here we go --
; first, find a $D5 nibble
BB6B-   BD 8C C0    LDA   $C08C,X
BB6E-   10 FB       BPL   $BB6B
BB70-   48          PHA
BB71-   68          PLA
BB72-   C9 D5       CMP   #$D5
BB74-   D0 F5       BNE   $BB6B

; count the number of $F7 nibbles (in Y
; register) before the next $D5 nibble
BB76-   A0 00       LDY   #$00
BB78-   8C 0F BC    STY   $BC0F
BB7B-   BD 8C C0    LDA   $C08C,X
BB7E-   10 FB       BPL   $BB7B
BB80-   C9 D5       CMP   #$D5
BB82-   F0 0F       BEQ   $BB93
BB84-   C9 F7       CMP   #$F7
BB86-   D0 01       BNE   $BB89
BB88-   C8          INY

; accumulator is always $F7 by now (the
; nibble we found -- anything else has
; branched off instead of falling
; through to this arithmetic)
BB89-   18          CLC
BB8A-   6D 0F BC    ADC   $BC0F
BB8D-   8D 0F BC    STA   $BC0F
BB90-   4C 7B BB    JMP   $BB7B

; execution continues here (from $BB82
; after we find the next $D5 nibble) --
; if we didn't find any $F7 nibbles,
; start over
BB93-   98          TYA
BB94-   F0 E0       BEQ   $BB76
```

```
; skip any number of $FF nibbles
BB96-   BD 8C C0    LDA   $C08C,X
BB99-   10 FB       BPL   $BB96

; killing time
BB9B-   48          PHA
BB9C-   68          PLA
BB9D-   C9 FF       CMP   #$FF
BB9F-   F0 F5       BEQ   $BB96

; if the first thing we find after the
; sequence of $FF nibbles is another
; $D5 nibble, fail immediately
BBA1-   C9 D5       CMP   #$D5
BBA3-   F0 35       BEQ   $BBDA

; skip next 5 nibbles
BBA5-   A0 05       LDY   #$05
BBA7-   BD 8C C0    LDA   $C08C,X
BBAA-   10 FB       BPL   $BBA7

; more time killing
BBAC-   48          PHA
BBAD-   68          PLA
BBAE-   88          DEY
BBAF-   D0 F6       BNE   $BBA7

; skip any number of $FF nibbles
BBB1-   BD 8C C0    LDA   $C08C,X
BBB4-   10 FB       BPL   $BBB1

; more time killing
BBB6-   48          PHA
BBB7-   68          PLA
BBB8-   C9 FF       CMP   #$FF
BBBA-   F0 F5       BEQ   $BBB1
```

```
; if the first thing we find after the
; sequence of $FF nibbles is another
; $D5 nibble, fail immediately
BBBC-   C9 D5         CMP   #$D5
BBBE-   D0 1A         BNE   $BBDA

; if the next nibble after that is not
; $FF, fail immediately
BBC0-   BD 8C C0      LDA   $C08C,X
BBC3-   10 FB         BPL   $BBC0
BBC5-   C9 FF         CMP   #$FF
BBC7-   D0 11         BNE   $BBDA

; check the counter (set at $BB8D)
BBC9-   AD 0F BC      LDA   $BC0F
BBCC-   38            SEC
BBCD-   E9 10         SBC   #$10

; if not zero, fail immediately
BBCF-   D0 09         BNE   $BBDA

; accumulator is 0 here, store it in
; $B739 (?!?!?)
BBD1-   8D 39 B7      STA   $B739   <-- !

; turn off drive motor
BBD4-   BD 88 C0      LDA   $C088,X

; continue elsewhere
BBD7-   4C 10 BC      JMP   $BC10
```

```
; The Badlands -- turn off drive motor,
; print error message, wipe memory,
; exit via $E000
BBDA-   BD 88 C0    LDA   $C088,X
BBDD-   AD 54 C0    LDA   $C054
BBE0-   AD 51 C0    LDA   $C051
BBE3-   AD 81 C0    LDA   $C081
BBE6-   20 58 FC    JSR   $FC58
BBE9-   A0 17       LDY   #$17
BBEB-   B9 F7 BB    LDA   $BBF7,Y
BBEE-   99 08 07    STA   $0708,Y
BBF1-   88          DEY
BBF2-   10 F7       BPL   $BBEB
BBF4-   4C 4B B7    JMP   $B74B

*FC58G N 400<BBF7.BC0FM

THIS IS A DEFECTIVE DISK
```

So judgmental.

Chapter 3
In Which Success Is Relative

Continuing from the success path at
$BC10...

```
*BC10L

BC10-   A0 00       LDY   #$00
BC12-   B9 1F BC    LDA   $BC1F,Y
BC15-   99 00 9A    STA   $9A00,Y
BC18-   C8          INY
BC19-   D0 F7       BNE   $BC12
BC1B-   4C 00 9A    JMP   $9A00

*BC1B:60
*BC10G

*9A00L

; This is actually the original call to
; $B793 that loads DOS from tracks 0-2
9A00-   20 93 B7    JSR   $B793

; save all status flags and registers,
; because we're about to do something
; else that is not loading DOS
9A03-   08          PHP
9A04-   48          PHA
9A05-   8A          TXA
9A06-   48          PHA
9A07-   98          TYA
9A08-   48          PHA

; set RWTS command = $01 (read)
9A09-   A9 01       LDA   #$01
9A0B-   8D F4 B7    STA   $B7F4

; sector $00
9A0E-   A9 00       LDA   #$00
9A10-   8D ED B7    STA   $B7ED
```

```
; track $0B (?!?)
9A13-   A9 0B       LDA   #$0B
9A15-   8D EC B7    STA   $B7EC

; address = $9900
9A18-   A9 00       LDA   #$00
9A1A-   8D F0 B7    STA   $B7F0
9A1D-   A9 99       LDA   #$99
9A1F-   8D F1 B7    STA   $B7F1

; read it
9A22-   A0 E8       LDY   #$E8
9A24-   A9 B7       LDA   #$B7
9A26-   20 B5 B7    JSR   $B7B5

; retry forever if that failed
9A29-   B0 FB       BCS   $9A26

; and continue there
9A2B-   4C 00 99    JMP   $9900
```

Dear Lord, there's still more to this copy protection.

```
*BSAVE DECRYPT BB03,A$2000,L$3E
*BSAVE BB00 DECRYPTED,A$BB00,L$156
```

A quick program to read T0B,S00 into
$9900 without having to trace up to
this point:

```
0300-   20 E3 03    JSR   $03E3
0303-   84 00       STY   $00
0305-   85 01       STA   $01
0307-   A0 01       LDY   #$01
0309-   A9 60       LDA   #$60
030B-   91 00       STA   ($00),Y
030D-   A0 04       LDY   #$04
030F-   A9 0B       LDA   #$0B
0311-   91 00       STA   ($00),Y
0313-   C8          INY
0314-   A9 00       LDA   #$00
0316-   91 00       STA   ($00),Y
0318-   A0 08       LDY   #$08
031A-   91 00       STA   ($00),Y
031C-   C8          INY
031D-   A9 99       LDA   #$99
031F-   91 00       STA   ($00),Y
0321-   A0 0C       LDY   #$0C
0323-   A9 01       LDA   #$01
0325-   91 00       STA   ($00),Y
0327-   20 E3 03    JSR   $03E3
032A-   4C D9 03    JMP   $03D9

*BSAVE READ T0BS00,A$300,L$2D
*300G
...read read read...
```

```
*BSAVE T0BS00 9900,A$9900,L$100

*9900L

9900-   4E 03 99    LSR   $9903
9903-   71 6E       ADC   ($6E),Y
9905-   07          ???
9906-   99 40 24    STA   $2440,Y
```

Are you !@#$%^& kidding me.

Chapter 4
In Which I Am Not !@#$%^& Kidding You

OK, here we go (again).

```
; make a copy of $9900
*2900<9900.99FFM
```

The start of my SECOND self-decryption replication program:

```
; copy $9900 page into place from a
; pristine copy in lower memory
2100-   A0 00       LDY   #$00
2102-   B9 00 29    LDA   $2900,Y
2105-   99 00 99    STA   $9900,Y
2108-   C8          INY
2109-   D0 F7       BNE   $2102
210B-   60          RTS

*210B:4E 03 99 60
*2100G

*9903L

9903-   38          SEC
9904-   6E 07 99    ROR   $9907
```

I tire of this, m'lord.

```
*210E:38 6E 07 99 60
*2100G

*9907L

9907-   A0 24       LDY   #$24
9909-   6E 0C 99    ROR   $990C
```

I'm gonna start singing.

```
*2112:A0 24 6E 0C 99 60
*2100G

*990CL

990C-   6E 18 99    ROR   $9918
990F-   6E 12 99    ROR   $9912
```

Nobody knows the trouble I've seen...

```
*2117:6E 18 99 6E 12 99 60
*2100G
*9912L

9912-   6E 1B 99    ROR   $991B
9915-   6E 22 99    ROR   $9922
9918-   B9 00 99    LDA   $9900,Y
```

Nobody knows but Woz...

```
*211D:6E 1B 99 6E 22 99 B9 00 99 60
*2100G

*991BL

991B-   59 00 B8    EOR   $B800,Y
991E-   99 00 99    STA   $9900,Y
9921-   C8          INY
9922-   D0 F4       BNE   $9918
```

Nobody knows the trouble I've seen...

```
*2126:59 00 B8 99 00 99 C8 D0 F4 60
*2100G

Glory, Hallelujah.

   _________________________________
  |___|\________________________|\__|
  |___|_________________|_______|___|
  |_(_)__________|\_____|___bb(_)___|
  |________|_____|____(_)._________|
          _|   (_)
        -(_)-
```

Chapter 5
In Which We Duckument The Most Unfriendly DOS Ever

```
*9924L

9924-   A0 23       LDY   #$23
9926-   B9 74 99    LDA   $9974,Y
9929-   99 4D A4    STA   $A44D,Y
992C-   88          DEY
992D-   10 F7       BPL   $9926
```

This overwrites part of DOS (at $A44D), which ends up looking 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.

```
992F-   A0 18       LDY   #$18
9931-   B9 95 99    LDA   $9995,Y
9934-   99 BC A3    STA   $A3BC,Y
9937-   88          DEY
9938-   10 F7       BPL   $9931
```

Another DOS patch. The end result:

```
| A3BC-   98          TYA
| A3BD-   4D 39 B7    EOR   $B739   <--
| A3C0-   51 67       EOR   ($67),Y <--
| A3C2-   91 67       STA   ($67),Y
| A3C4-   88          DEY
| A3C5-   C0 FF       CPY   #$FF
| A3C7-   D0 F3       BNE   $A3BC
| A3C9-   60          RTS
| A3CA-   A9 01       LDA   #$01
| A3CC-   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.

Note that there are two EOR statements, including $B739, the value of which was changed after the nibble check at $BB03 succeeded. So many layers...

```
993A-   A0 14         LDY   #$14
993C-   B9 A8 99      LDA   $99A8,Y
993F-   99 30 9E      STA   $9E30,Y
9942-   88            DEY
9943-   10 F7         BPL   $993C
```

DOS patch #3. The result:

```
| 9E30-   A9 80         LDA   #$80
| 9E32-   85 D6         STA   $D6
| 9E34-   A9 06         LDA   #$06
| 9E36-   D0 12         BNE   $9E4A
| 9E38-   AD 00 C0      LDA   $C000
| 9E3B-   C9 83         CMP   #$83
| 9E3D-   D0 03         BNE   $9E42
| 9E3F-   EA            NOP
| 9E40-   F0 F6         BEQ   $9E38
| 9E42-   4C D2 D7      JMP   $D7D2
```

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 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.

```
9945-   A0 02          LDY   #$02
9947-   B9 BD 99       LDA   $99BD,Y
994A-   99 03 A5       STA   $A503,Y
994D-   88             DEY
994E-   10 F7          BPL   $9947
```

DOS patch #4. The result:

```
| A503-   4C 38 9E       JMP   $9E38
```

This is the tail end of the RUN entry point. It's just a JMP to the code that was just patched earlier, that ensures that 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!)

```
9950-   A0 02          LDY   #$02
9952-   B9 C0 99       LDA   $99C0,Y
9955-   99 8B A3       STA   $A38B,Y
9958-   88             DEY
9959-   10 F7          BPL   $9952
```

DOS patch #5. The result:

```
| A38B-   4C 82 A5       JMP   $A582
```

This patch adds a "JMP $A582" to the end of the BLOAD command handler that starts at $A35D. Not sure what $A582 does, but I'm guessing I'm about to find out.

```
995B-   A0 31       LDY   #$31
995D-   B9 C3 99    LDA   $99C3,Y
9960-   99 7F A5    STA   $A57F,Y
9963-   88          DEY
9964-   10 F7       BPL   $995D
```

DOS patch #6. The result:

```
| 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
```

Patch #5 set up a jump to $A582 at the
end of the BLOAD handler. It looks like
patch #6 is reusing the decryption
routine at $A3BC (already used for
Applesoft programs) for binary programs
as well. Encrypt all the things!

```
9966-   A9 A2          LDA   #$A2
9968-   8D 27 A4       STA   $A427
```

This patches a branch in the middle of
the LOAD handler so that DOS doesn't
try to load Integer Basic programs.
(Previous patches overwrote the Integer
Basic handling for their own purposes.)

```
; restore everything and continue with
; the boot
996B-   68             PLA
996C-   AA             TAX
996D-   68             PLA
996E-   A8             TAY
996F-   68             PLA
9970-   28             PLP
9971-   4C 3B B7       JMP   $B73B
```

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>,
and sets the RUN flag.

It does not, however, hinder copying
the disk itself. To bypass the copy
protection, I can write the decrypted
$BB00/$BC00 back to disk, jump to a
short routine at $BC06 that sets the
only two long-term side effects I can
find (at $B739 and $B73A, and I'm not
even sure the second one is necessary
but I'm not willing to risk it), then
falls through to the success path at
$BC10.

```
*BLOAD BOOT1,A$2600
*BLOAD BB00 DECRYPTED,A$2B00

; change "JMP $BB03" to "JMP $BC06"
*2738:4C 06 BC

; set up patch at $BC06
*2C06:A9 00 8D 39 B7 A9 E0 8D 3A B7

*2C06L

; my patch
2C06-   A9 00       LDA   #$00
2C08-   8D 39 B7    STA   $B739
2C0B-   A9 E0       LDA   #$E0
2C0D-   8D 3A B7    STA   $B73A

; existing code at $BC10
2C10-   A0 00       LDY   #$00
2C12-   B9 1F BC    LDA   $BC1F,Y
2C15-   99 00 9A    STA   $9A00,Y
2C18-   C8          INY
2C19-   D0 F7       BNE   $2C12
2C1B-   4C 00 9A    JMP   $9A00
```

```
; short program to write the decrypted
; and patched boot1 back to disk
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 0A 00 00 00 00 00 00
         ^^
    sector count

08E8- 01 60 01 00 00 09 FB 08
                  ^^ ^^
         start track/sector

08F0- 00 2F 00 00 02 00 FE 60
         ^^       ^^ command (write)
   start address

08F8- 01 00 00 00 01 EF D8 00
```

```
[S6,D1=non-working copy]

*8C0G
...write write write...

*C600G
...works...
```

Quod erat liberandum.