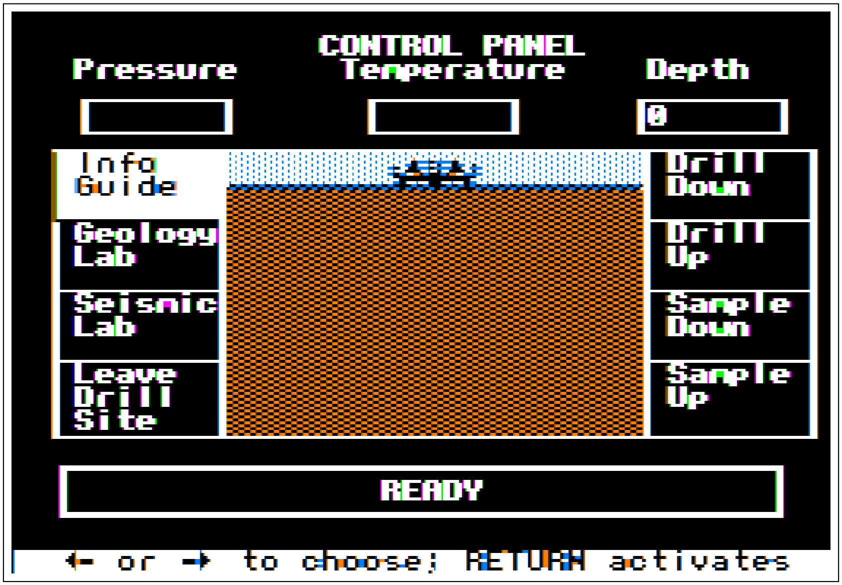
The Changing Earth



2015-09-07

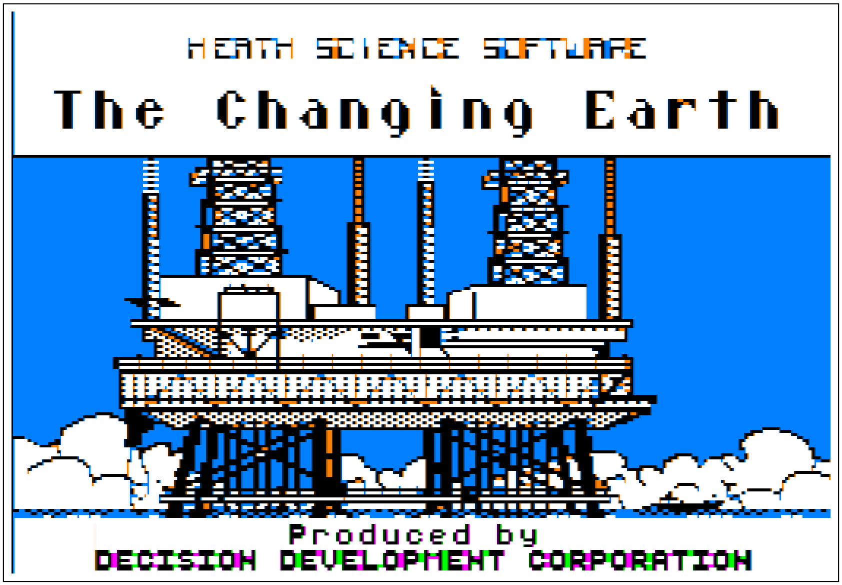
Contents

0 In Which Various Automated Tools Fail In Interesting Ways 4

1 In Which Our Automated Tools Are Very Proud Of Themselves But Don't Actually Accomplish Anything 6

2 In Which We Painstakingly Create A Repeatable Decryption Routine, And It Stakes About As Much Pain As We Expected 10

3 In Which We Are Triumphant, But Only For A Moment, And Then After Many More Moments We Are Triumphant Again 21



```
-----------The Changing Earth-----------
A 4am crack                   2015-09-07
----------------------------------------

Name: The Changing Earth
Genre: educational
Year: 1985
Authors: Decision Development
Publisher: D.C. Heath and Company
Media: single-sided 5.25-inch floppy
OS: custom with DOS 3.3 bootloader
Previous cracks: none
```

Chapter 0
In Which Various Automated Tools Fail In Interesting Ways

COPYA
  no read errors, but copy displays
  message "DISK IS DEFECTIVE" and quits
  to a BASIC prompt with no DOS loaded

Locksmith Fast Disk Backup
  ditto

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

Disk Fixer
  T00 looks like a DOS 3.3 boot0/boot1
  No sign of the rest of DOS
  No sign of a disk catalog anywhere

Why didn't any of my copies work?
  I don't know. Maybe a nibble check
  during boot?

Next steps:

  1. Trace the boot
  2. ???

Chapter 1
In Which Our Automated Tools Are Very Proud Of Themselves But Don't Actually Accomplish Anything

```
[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

]BLOAD BOOT1,A$2600
]CALL -151

; 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

; fill a standard RWTS parameter table
B700-   8E E9 B7    STX   $B7E9
B703-   8E F7 B7    STX   $B7F7
B706-   A9 01       LDA   #$01
B708-   8D F8 B7    STA   $B7F8
B70B-   8D EA B7    STA   $B7EA
B70E-   AD E0 B7    LDA   $B7E0
B711-   8D E1 B7    STA   $B7E1

; track $08
B714-   A9 08       LDA   #$08
B716-   8D EC B7    STA   $B7EC
```

```
; sector $0B
B719-   A9 0B       LDA   #$0B
B71B-   8D ED B7    STA   $B7ED
B71E-   AC E7 B7    LDY   $B7E7
B721-   88          DEY
B722-   8C F1 B7    STY   $B7F1
B725-   A9 01       LDA   #$01
B727-   8D F4 B7    STA   $B7F4
B72A-   8A          TXA
B72B-   4A          LSR
B72C-   4A          LSR
B72D-   4A          LSR
B72E-   4A          LSR
B72F-   AA          TAX
B730-   A9 00       LDA   #$00
B732-   9D F8 04    STA   $04F8,X
B735-   9D 78 04    STA   $0478,X

; hmm
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          ???
```

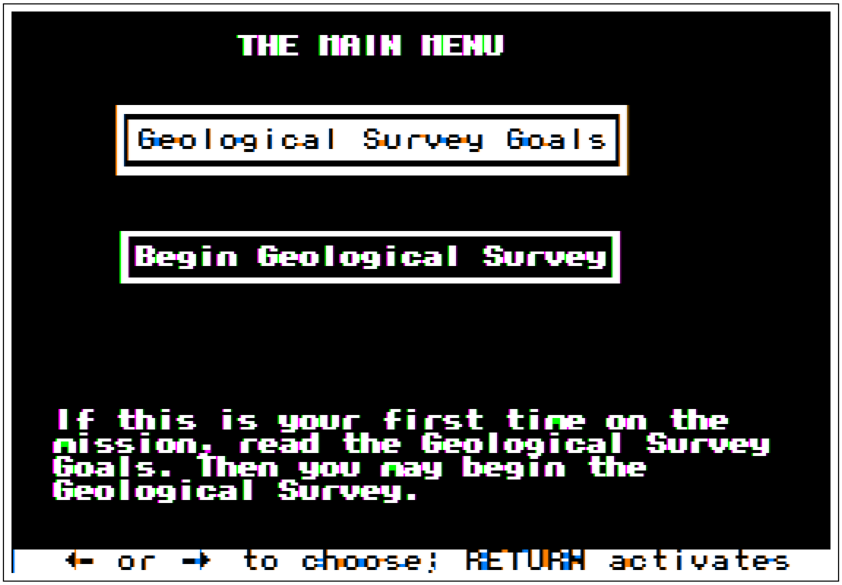
Oh look, self-modifying code. This should be fun(*).

(*) not guaranteed, actual fun may vary

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.


THE MAIN MENU

Geological Survey Goals

Begin Geological Survey

If this is your first time on the mission, read the Geological Survey Goals. Then you may begin the Geological Survey.

← or → to choose; RETURN activates

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.

```
*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!)

```
; set JMP that brought us here to $E000
BB35-   A9 00       LDA   #$00
BB37-   8D 39 B7    STA   $B739
BB3A-   A9 E0       LDA   #$E0
BB3C-   8D 3A B7    STA   $B73A

; sets an unfriendly reset vector
BB3F-   20 C3 B7    JSR   $B7C3
```

```
; read a sector from track $00 via the
; disk controller ROM
BB42-   A9 60       LDA   #$60
BB44-   8D 01 08    STA   $0801
BB47-   A9 0E       LDA   #$0E
BB49-   85 27       STA   $27
BB4B-   85 3D       STA   $3D
BB4D-   A6 2B       LDX   $2B
BB4F-   8A          TXA
BB50-   4A          LSR
BB51-   4A          LSR
BB52-   4A          LSR
BB53-   4A          LSR
BB54-   09 C0       ORA   #$C0
BB56-   8D 5B BB    STA   $BB5B
BB59-   20 5C C6    JSR   $C65C
```

(The code below doesn't appear to access this sector data, so I think this was just to position the drive head for the thing we're about to do.)

```
; find $AA nibble
BB5C-   BD 8C C0    LDA   $C08C,X
BB5F-   10 FB       BPL   $BB5C
BB61-   C9 AA       CMP   #$AA
BB63-   D0 F7       BNE   $BB5C

; initialize a counter
BB65-   18          CLC
BB66-   A9 1F       LDA   #$1F
BB68-   85 00       STA   $00
```

```
; find 5 $FF nibbles in a row
BB6A-   A0 05          LDY   #$05
BB6C-   BD 8C C0       LDA   $C08C,X
BB6F-   10 FB          BPL   $BB6C
BB71-   48             PHA
BB72-   68             PLA
BB73-   49 FF          EOR   #$FF

; if we find something that isn't $FF,
; start over (reset Y to 5)
BB75-   D0 F3          BNE   $BB6A

; keep going until we find 5 in a row
BB77-   88             DEY
BB78-   D0 F2          BNE   $BB6C

BB7A-   F0 31          BEQ   $BBAD
```

```
*BBADL

; time the nibble to see how many
; timing bits are attached, and add to
; a running sum -- more timing bits
; add more to a rolling sum (held in
; accumulator)
BBAD-   BC 8C C0    LDY   $C08C,X
BBB0-   30 2A       BMI   $BBDC-------.
BBB2-   BC 8C C0    LDY   $C08C,X     |
BBB5-   30 20       BMI   $BBD7------.|
BBB7-   BC 8C C0    LDY   $C08C,X    ||
BBBA-   30 16       BMI   $BBD2-----.||
BBBC-   BC 8C C0    LDY   $C08C,X   |||
BBBF-   30 0C       BMI   $BBCD----.|||
BBC1-   BC 8C C0    LDY   $C08C,X  ||||
BBC4-   30 02       BMI   $BBC8---.||||
; no timing bits -> The Badlands  |||||
BBC6-   10 19       BPL   $BBE1   |||||
; (from $BBC4)                    |||||
BBC8-   69 04       ADC   #$04 <--/||||
BBCA-   90 B0       BCC   $BB7C    ||||
BBCC-   60          RTS            ||||
; (from $BBBF)                     ||||
BBCD-   69 03       ADC   #$03 <---/|||
BBCF-   90 AB       BCC   $BB7C     |||
BBD1-   60          RTS             |||
; (from $BBBA)                      |||
BBD2-   69 02       ADC   #$02 <----/||
BBD4-   90 A6       BCC   $BB7C      ||
BBD6-   60          RTS              ||
; (from $BBB5)                       ||
BBD7-   69 01       ADC   #$01 <-----/|
BBD9-   90 A1       BCC   $BB7C       |
BBDB-   60          RTS               |
; (from $BBB0)                        |
BBDC-   69 00       ADC   #$00 <------/
BBDE-   90 9C       BCC   $BB7C
BBE0-   60          RTS

All branches jump back to $BB7C.
```

```
*BB7CL

; kill a few cycles (not pointless,
; because the disk spins independently
; of the CPU, so all of these low-level
; disk reads are highly time-sensitive)
BB7C-   EA          NOP
BB7D-   EA          NOP

; decrement counter (initialized to $1F
; at $BB68)
BB7E-   C6 00       DEC   $00

; go back and count more bits
BB80-   D0 2B       BNE   $BBAD

; final sum in the accumulator must be
; $35 < A <= $48, or it's off to
; The Badlands
BB82-   C9 35       CMP   #$35
BB84-   90 5B       BCC   $BBE1
BB86-   C9 48       CMP   #$48
BB88-   B0 57       BCS   $BBE1

; But wait, there's more!
BB8A-   A0 0B       LDY   #$0B

; find a $D5 nibble
BB8C-   BD 8C C0    LDA   $C08C,X
BB8F-   10 FB       BPL   $BB8C
BB91-   48          PHA
BB92-   68          PLA
BB93-   C9 D5       CMP   #$D5
BB95-   D0 F5       BNE   $BB8C
```

```
; Now we get an entire nibble sequence
; and match it against an array of
; known nibbles
BB97-   BD 8C C0    LDA   $C08C,X
BB9A-   10 FB       BPL   $BB97
BB9C-   85 00       STA   $00

; interestingly, the array has some $00
; values which act as wildcards (match
; any nibble)
BB9E-   B9 0F BC    LDA   $BC0F,Y
BBA1-   F0 04       BEQ   $BBA7
BBA3-   C5 00       CMP   $00
BBA5-   D0 3A       BNE   $BBE1
BBA7-   88          DEY
BBA8-   10 ED       BPL   $BB97

; finally satisfied, continue elsewhere
BBAA-   4C 1B BC    JMP   $BC1B
```

Here is the array we're matching against:

```
*BC0F.BC1A

BC0F-                      AA
BC10- DE 00 00 AA AA 00 00 00
BC18- 00 96 AA
```

The array is stored in reverse order.
Y is the index, initialized to $0B at
$BB8A, then decremented. So after
finding a $D5 nibble, this is the
sequence we're looking for:

```
AA 96 * * * * AA AA * * DE AA
```

That's the rest of an address prologue,
an entire address field, and an address
epilogue. But the only part of the
address field we actually care about is
the sector number: AA AA, which is $00.

So, not only do we need to find the
right number of timing bits, we need to
end up at the right place in the track
after counting them. Which is why we
did a dummy sector read to begin with.

Meanwhile, for those unlucky souls who
fail this complicated check, this is
where you end up:

```
*BBE1L

; turn off drive motor
BBE1-   BD 88 C0    LDA   $C088,X

; switch to text, clear screen
BBE4-   AD 54 C0    LDA   $C054
BBE7-   AD 51 C0    LDA   $C051
BBEA-   AD 81 C0    LDA   $C081
BBED-   20 58 FC    JSR   $FC58
```

```
; print error message
BBF0-    A0 10          LDY   #$10
BBF2-    B9 FE BB       LDA   $BBFE,Y
BBF5-    99 0B 07       STA   $070B,Y
BBF8-    88             DEY
BBF9-    10 F7          BPL   $BBF2

; clear main memory and exit via $E000
; (not shown)
BBFB-    4C 4B B7       JMP   $B74B

BBFE-   ["DISK IS DEFECTIVE"]
```

For those few lucky souls who pass,
untold riches await you at $BC1B.

```
*BC1BL

; ha! just kidding! one final check!
BC1B-    AD AF BE       LDA   $BEAF
BC1E-    C9 A0          CMP   #$A0
BC20-    D0 BF          BNE   $BBE1
BC22-    A9 AA          LDA   #$AA
BC24-    8D AF BE       STA   $BEAF

; restore proper code in boot1
BC27-    A9 20          LDA   #$20
BC29-    8D 38 B7       STA   $B738
BC2C-    A9 93          LDA   #$93
BC2E-    8D 39 B7       STA   $B739
BC31-    A9 B7          LDA   #$B7
BC33-    8D 3A B7       STA   $B73A

; ...and jump there
BC36-    4C 38 B7       JMP   $B738
```

And that's all she wrote.

Chapter 3
In Which We Are Triumphant, But Only For A Moment, And Then After Many More Moments We Are Triumphant Again

Let's save our replication routine and
the decrypted result.

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

*2600<B600.BFFFM
*BSAVE BOOT1 DECRYPTED,A$2600,L$A00
```

And now, the triumphant patch to make
the modification to boot1 (from $BC27)
and bypass the entire copy protection
in one fell swoop:

```
T00,S01,$38 change "4C 03 BB"
                to "20 93 B7"

]PR#6
...boots, loads several tracks, then
   displays "RESTART COMPUTER"...
```

Well, that sucks. What did I miss?
Or is there simply a secondary
protection deeper within the program?

.
. [six months pass]
.

No seriously, I got that far and gave
up and put this disk on the back shelf,
both physically and metaphorically. Six
months later, I picked it up and took
another look at the protection routine
and immediately saw what I had missed:

```
``'-.,_,.-'``'-.,_,.='``'-.,_,.-'``'-.,
``'-.,_,.-'``'-.,_,.='``'-.,_,.-'``'-.,
``                                   .,
``  BC22-   A9 AA       LDA   #$AA   .,
``  BC24-   8D AF BE    STA   $BEAF  .,
``                                   .,
``'-.,_,.-'``'-.,_,.='``'-.,_,.-'``'-.,
``'-.,_,.-'``'-.,_,.='``'-.,_,.-'``'-.,
```

A completely innocuous-looking side
effect. It makes no sense, actually.
$BEAF is an entry point to a relatively
low-level RWTS routine, and $AA is not
a valid opcode, so that would make the
entire procedure crash.

Nonetheless, that is what the original
copy protection routine is doing. Could
it be that simple?

```
T00,S08,$AF change "A0" to "AA"

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

Note to self: there are no innocuous
side effects.

Quod erat liberandum.