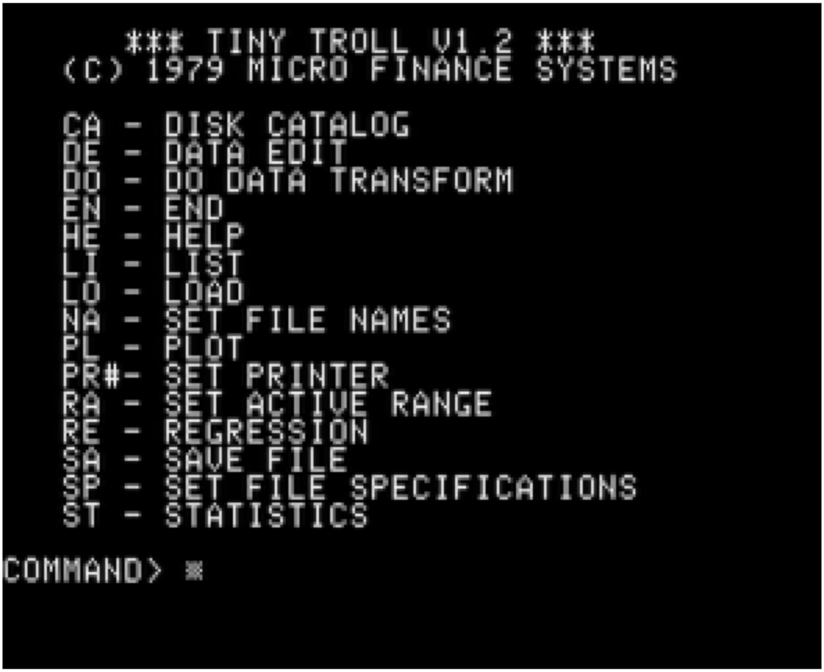
Tiny Troll



```
    *** TINY TROLL V1.2 ***
  (C) 1979 MICRO FINANCE SYSTEMS

  CA - DISK CATALOG
  DE - DATA EDIT
  DO - DO DATA TRANSFORM
  EN - END
  HE - HELP
  LI - LIST
  LO - LOAD
  NA - SET FILE NAMES
  PL - PLOT
  PR#- SET PRINTER
  RA - SET ACTIVE RANGE
  RE - REGRESSION
  SA - SAVE FILE
  SP - SET FILE SPECIFICATIONS
  ST - STATISTICS

COMMAND> 
```

2016-04-05

4am

to deprotect
and preserve

Contents

```
---------------Tiny Troll---------------
A 4am crack                   2016-04-05
---------------------------------------

Name: Tiny Troll
Version: 1.2
Genre: productivity
Year: 1979
Author: Mitch Kapor
Publisher: Micro Finance Systems
Platform: Apple ][+ or later
Media: single-sided 5.25-inch floppy
OS: DOS 3.2
Previous cracks: none
Identical cracks:
  #1068 Tiny Troll v1.1
```

Chapter 0
In Which Various Automated Tools Fail In Interesting Ways

This is a very old disk. It does not boot automatically on my Apple //e, presumably because it uses a 13-sector format and does not have a bootloader that allows it to boot on 16-sector disk drives.

I was able to boot the original disk by first booting "DOS 3.3 Basics" and swapping in Tiny Troll when prompted for a 13-sector diskette. The process looks something like this:

```
[S6,D1=DOS 3.3 Basics (system disk)]

]PR#6
...

                  --v--

   INSERT YOUR 13-SECTOR DISKETTE
           AND PRESS RETURN

                  --^--

[S6,D1=Tiny Troll (original disk)]

                  --v--

>

]

...read read read...
...eventually displays main menu...

                  --^--
```

This means that a wide array of 16-
sector disk tools will be useless.
Locksmith Fast Disk Backup? Ha! COPYA?
Instant read error. Even Disk Edit, my
favorite sector editor, only works with
16-sector disks.

I did try Essential Data Duplicator 4,
a powerful disk copy utility that
copies entire tracks at once. Here is
what happens when I boot DOS 3.3 Basics
and swap in the disk that EDD produced:

```
                  --v--

>

]

ERR


I/O ERROR

BREAK IN 30

                  --^--
```

At this point, I can LIST the program
in memory (more on that in a minute),
but any DOS command gives "I/O ERROR".

Chapter 1
Yesterday All My Troubles Seemed So Far Away

Next up: Copy II+ 5.5. This is the last version of Copy II+ that can read 13-sector disks, once you tell it that the disk in a certain slot and drive should be treated as DOS 3.2.

```
[S6,D1=original disk]
[S6,D2=formatted DOS 3.3 disk]

[Copy II+]
  [NEW DISK INFO]
    [A: S6, D1, DOS 3.2]
    [B: S6, D2, DOS 3.3]

  [CATALOG DISK]
    [NORMAL]
      [DISK A]
```

```
CATALOG DISK                          DISK A

DISK VOLUME 099

*A 002 HELLO
*A 057 TINY TROLL
*B 002
*B 002
*A 015 TT DEMO HEADER
*A 043 TT CAPABILITIES DEMO
*A 045 TT BUSINESS DEMO
*A 054 TT STOCK DEMO
*B 008 TEXT SET 2
*T 005 RF
*T 005 NYSE
*T 005 DEC
*T 003 IBM.P
*T 003 IBM.DIV
*T 003 IBM.EARN
*T 003 DUMMY
*T 005 R
*T 005 NYSER
*T 005 DECR
*T 003 TEXT101
*T 004 TEXT102
*T 002 TEXT103
*T 003 TEXT104
*T 002 TEXT105
*T 002 TEXT106
*T 002 TEXT107
*T 004 TEXT108
*T 004 TEXT109
*T 004 TEXT110
*T 004 TEXT201
*T 003 TEXT202
*T 002 TEXT203
*T 002 TEXT204
*T 002 TEXT205
*T 003 TEXT206
*T 003 TEXT401
[...]
```

```
*T 002 TEXT402
*T 002 TEXT403
*T 003 TEXT404
*T 002 TEXT405
*A 002 GOODBYE
*B 002 TT ROUTINES
```

--^--

Well that's extremely promising.

Copy II+ 5.5 can also copy files between DOS 3.2 and DOS 3.3 disks. This program appears to be file-based, so let's try copying all the files to a freshly formatted DOS 3.3 disk and booting that.

```
[Copy II+]
  [COPY]
    [FILES]
      [A TO B]
        (select all 42 files)

...read read read...
...write write write...

[S6,D1=my copy]

]PR#6
```

```
                 --v--

]

033D-    A=03 X=9D Y=3B P=30 S=F2
*

                 --^--
```

And that is... not extremely promising.
It boots to DOS, displays the expected
"]" prompt, and crashes to the monitor.

Chapter 2
I Don't Know Why You Say GOODBYE, I Say HELLO

At this point, I realize there is a
crucial step I forgot. After copying
all the files to a new disk, I should
ensure that my copy has the proper boot
program. Normally the boot program is
named HELLO, but like so many other
things in Apple II land, this is only a
convention.

Back in Copy II+ 5.5, I can find out
which file is the boot program on the
original disk by (somewhat confusingly)
selecting "CHANGE BOOT PROGRAM" but not
actually changing it.

```
[Copy II+]
  [CHANGE BOOT PROGRAM]
    [DISK A]

                 --v--

CHANGE BOOT PROGRAM              DISK A
---------------------------------------
    A 002 HELLO
    A 056 TINY TROLL
                  ...
    T 003 DUMMY
    T 005 R
----------------- MORE ----------------

FILE: GOODBYE
IS THE CURRENT BOOTING PROGRAM.
[G]O, [E]NTER FILENAME, [ESC] TO EXIT

                 --^--
```

That might explain why my copy is
crashing into the monitor at $033D. The
boot program is "GOODBYE", not "HELLO".

Let me fix that on my copy:

```
[Copy II+]
  [CHANGE BOOT PROGRAM]
    [DISK B]
      (scroll down to select "GOODBYE")
      [GO]

[S6,D1=my copy, once again]

]PR#6
```

```
                 --v--

]

ERR


I/O ERROR

BREAK IN 30

                 --^--
```

Hmm, that's the exact error I got on my failed EDD bit copy. So... I guess that counts as progress?

At this point, I'm convinced there is code explicitly checking whether the disk is original. (It's not.) Since I have access to the program in memory, let's take advantage of that.

```
]LIST

5 U1$ = "U1" +  CHR$ (8) +  CHR$
     (8):U2$ = "U2" +  CHR$ (8) +
      CHR$ (8)
10 D$ =  CHR$ (13) +  CHR$ (4): PRINT
     D$;"MAXFILES 1";D$;"BLOAD ";
     U1$;D$;"BLOAD ";U2$
20 CALL 49152 : REM
30  PRINT D$;"RUN TINY TROLL"
45312$;"RUN TINY TROLL"
```

OK, I don't understand why running a BASIC program could cause an I/O error, especially after two successful BLOAD commands. I guarantee there are no bad sectors (intentional or otherwise) on my copy.

I think it's time to revisit the original disk. Doing the DOS 3.3 Basics disk swapping dance, I eventually hit <Ctrl-C> just as Tiny Troll displays the "]" prompt, and it works!

```
                --v--

>

]
<Ctrl-C>

BREAK
]

                 --^--
```

Excellent! I'm able to break to a BASIC prompt at exactly the point in the boot where my non-working copy crashed.

```
]LIST

5 U1$ = "U1" +  CHR$ (8) +  CHR$
     (8):U2$ = "U2" +  CHR$ (8) +
      CHR$ (8)
10 D$ =  CHR$ (13) +  CHR$ (4): PRINT
     D$;"MAXFILES 1";D$;"BLOAD ";
     U1$;D$;"BLOAD ";U2$
20 CALL 49152 : REM
30  PRINT D$;"RUN TINY TROLL"
45312$;"RUN TINY TROLL"
```

And here we are again. On the bright side, it looks like the same boot program, so I've got that going for me, which is nice.

Chapter 3
Everybody's Got Something To Hide Except Me And My Monkey

Taking it line by line, there's some nice filename obfuscation on line 5. Each filename is two real characters, followed by two <Ctrl-H> characters, which act like a left arrow and move the cursor back over the preceding characters. Fun(*) fact: filenames are stored on disk as fixed length strings right-padded with spaces, which are ignored. So if you catalog this disk, DOS will hide these filenames entirely.

```
]CATALOG

DISK VOLUME 100

 A 002 HELLO
 A 056 TINY TROLL
 B 002
 B 002
 A 015 TT DEMO HEADER
 A 043 TT CAPABILITIES DEMO
 A 045 TT BUSINESS DEMO
 A 054 TT STOCK DEMO
 B 008 TEXT SET 2
...[snipped]...
```

The two blank entries in the disk catalog are "U1<Ctrl-H><Ctrl-H>" and "U2<Ctrl-H><Ctrl-H>" that the boot program BLOADs. Copy II+ showed them as their actual filenames, without the control characters, because that's how it rolls.

(*) not guaranteed, actual fun may vary

Rather than fighting with DOS to load those files some other way, I think the best course would be to leave the BASIC code in place, let it load those files for me, then insert an "END" statement so I regain control.

```
]15 END

]RUN
```

...read read read...

OK, let's see what we loaded. Line 20 is calling 49152, which in hex would be
.
.
.

$C000.

That can't be right. That is ROM space.

Wait, I have a hunch. That empty "REM" statement at the end of line 20 makes me suspicious. Let's slow down the listing and see if...

```
]SPEED=1
```

```
]LIST 20

20  CALL 16384: REM
```

...then slowly overwrites with what I thought was the original listing,

```
20 CALL 49152 : REM
```

The "REM" comment contains <Ctrl-H> characters that move the cursor to the left and overwrite the original CALL statement with a fake number.

Trolled by <Ctrl-H> again, 38 years later.

Chapter 4
She's A Big Teaser
She Took Me Half The Way There

16384 in hex is $4000, which is a much
more reasonable starting address.

```
]CALL -151

*4000L

; get address of RWTS parameter table
4000-   20 E3 03     JSR   $03E3
4003-   84 48        STY   $48
4005-   85 49        STA   $49

; RWTS+3 = disk volume (0 = wildcard)
4007-   A0 03        LDY   #$03
4009-   A9 00        LDA   #$00
400B-   91 48        STA   ($48),Y
400D-   C8           INY

; RWTS+4 = track (7)
400E-   A0 04        LDY   #$04
4010-   A9 07        LDA   #$07
4012-   91 48        STA   ($48),Y

; RWTS+5 = sector (0)
4014-   C8           INY
4015-   A9 00        LDA   #$00
4017-   91 48        STA   ($48),Y

; RWTS+6 = pointer to DCT (saving this
; in zero page for some reason?)
4019-   C8           INY
401A-   B1 48        LDA   ($48),Y
401C-   85 3C        STA   $3C
401E-   C8           INY
401F-   B1 48        LDA   ($48),Y
4021-   85 3D        STA   $3D
```

```
; RWTS+8 = address ($6000)
4023-   C8          INY
4024-   A9 00       LDA   #$00
4026-   91 48       STA   ($48),Y
4028-   C8          INY
4029-   A9 60       LDA   #$60
402B-   91 48       STA   ($48),Y

; RWTS+$0C = command (1 = read)
402D-   A0 0C       LDY   #$0C
402F-   A9 01       LDA   #$01
4031-   91 48       STA   ($48),Y

; woah woah woah woah woah woah woah
; woah woah
4033-   A0 01       LDY   #$01
4035-   A9 00       LDA   #$00
4037-   91 3C       STA   ($3C),Y
```

We're messing with the DCT. Nobody ever
messes with the DCT. I don't even know
what the DCT *is*, and I've cracked
over a thousand different disks. I
don't even know what it stands for.

According to the indispensible "Beneath Apple DOS" (p. 8-35), "DCT" stands for "Device Characteristics Table". It is only four bytes long and should never, ever change:

```
   DCT+0 - device type (should be $00)
   DCT+1 - phases per track (should be
           $01)
   DCT+2 - motor on time count (should
           be $EF, $D8)
```

But we are changing it, specifically the "phases per track" field, from #$01 to #$00. I don't know what effect this has, because literally no one ever does this.

One phase is half a track. One track is two phases. These are just definitions. If you want to seek to track N, you tell the RWTS to seek to phase N*2. All disks work this way.

Except this one.

I pored through memory until I found
where the DCT is used. Here, shortly
after the RWTS entry point at $BD00, we
get the DCT address from the RWTS
parameter table:

```
BD42-   A0 06         LDY   #$06
BD44-   B1 48         LDA   ($48),Y
BD46-   99 36 00      STA   $0036,Y
BD49-   C8            INY
BD4A-   C0 0A         CPY   #$0A
BD4C-   D0 F6         BNE   $BD44
```

Among other things, that will set up
($3C) to point to the DCT, because it's
taking RWTS+6 and putting it in $36+6.

Then, when we try to switch to a
different track, we see this code:

```
*BE3BL

; accumulator has the desired track at
; this point ($00-$22 on a normal disk)
BE3B-   48            PHA

; get DCT+1 (documented as "phases per
; track")
BE3C-   A0 01         LDY   #$01
BE3E-   B1 3C         LDA   ($3C),Y

; look at bit 0
BE40-   6A            ROR
BE41-   68            PLA

; if bit 0 is 0, skip ahead to seek
BE42-   90 08         BCC   $BE4C
```

```
; otherwise, multiply the desired track
; by 2 (now $00-$44)
BE44-   0A              ASL

; now seek
BE45-   20 4C BE        JSR   $BE4C
BE48-   4E 78 04        LSR   $0478
BE4B-   60              RTS
```

Then $BE4C initializes the drive motor and seeks to the phase it was given. So there are really only two code paths, one that moves two phases per track, and another that moves one phase per track. By changing DCT+1 from #$01 to #$00, we've told the drive seek routine to treat its input as a phase, not a track. We're passing the accumulator directly to the drive seek routine at $BE4C, without ever multiplying it by 2 (with the "ASL" at $BE44).

DCT+1 isn't really "phases per track." It's a boolean, either 0 or 1.

```
   1 - seek by track (default)
   0 - seek by phase
```

So we're seeking to track 3.5.

Chapter 5
Full Speed Ahead, Mr. Boatswain Full Speed Ahead

Continuing the disassembly at $4039...

```
; call the RWTS to read from track 3.5
4039-   20 E3 03    JSR   $03E3
403C-   20 D9 03    JSR   $03D9

; if that doesn't work, skip ahead
403F-   B0 29       BCS   $406A

; increment the target memory address
4041-   A0 09       LDY   #$09
4043-   B1 48       LDA   ($48),Y
4045-   69 01       ADC   #$01
4047-   91 48       STA   ($48),Y

; increment the sector to read
4049-   A0 05       LDY   #$05
404B-   B1 48       LDA   ($48),Y
404D-   69 01       ADC   #$01
404F-   91 48       STA   ($48),Y

; loop for 6 sectors
4051-   C9 06       CMP   #$06
4053-   90 E4       BCC   $4039

; get a pointer to DCT again
4055-   A0 06       LDY   #$06
4057-   B1 48       LDA   ($48),Y
4059-   85 3C       STA   $3C
405B-   C8          INY
405C-   B1 48       LDA   ($48),Y
405E-   85 3D       STA   $3D
```

```
; restore DCT+1 ("phases per track" or
; whatever) to its default value
4060-   A0 01       LDY   #$01
4062-   98          TYA
4063-   91 3C       STA   ($3C),Y

; RWTS all-clear signal
4065-   A9 00       LDA   #$00
4067-   85 48       STA   $48

; and return gracefully to the caller
; (the BASIC startup program)
4069-   60          RTS

; Execution ends up here if there was
; any RWTS error reading from track 3.5
; (will print "ERR" -- which I saw on
; both my failed bit copy and my failed
; file-by-file copy)
406A-   4C 2D FF    JMP   $FF2D
```

Hey, that also explains why my failed copies just gave "I/O ERROR" to any DOS command after this routine failed. In the event of any RWTS error, it jumps over the code at $4055 that restores DCT+1 to its default value. So DOS is still trying to move by half tracks instead of whole tracks, which doesn't work at all.

Returning to the boot program...

I should be able to change line 30 to
return to the BASIC prompt instead of
running the "TINY TROLL" program.
Whatever we're loading from track 3.5
should be in memory at $6000.

```
; clear memory with a custom byte
*6000:FD N 6001<6000.6FFEM

; return to BASIC
*3D0G

; end the program after reading from
; track 3.5
]30 END

; and go
]RUN
...read read read...

]
```

Excellent. I'm back at the BASIC prompt
with the data from track 3.5 in memory.

```
]CALL -151

*6000L

6000-   5F          ???
6001-   05 00       ORA   $00
6003-   01 0B       ORA   ($0B,X)
6005-   01 16       ORA   ($16,X)
6007-   01 1E       ORA   ($1E,X)
6009-   01 2A       ORA   ($2A,X)
600B-   01 35       ORA   ($35,X)
600D-   01 3E       ORA   ($3E,X)
```

OK well that's not executable code, but it wasn't there a minute ago, so I'll take it.

We loaded 6 sectors, so there should be new stuff in $6000..$65FF. And here's something interesting at $6500:

```
*6500L

6500-   A9 40       LDA   #$40
6502-   85 01       STA   $01
6504-   A0 00       LDY   #$00
6506-   84 00       STY   $00
6508-   B1 00       LDA   ($00),Y
650A-   49 7F       EOR   #$7F
650C-   91 00       STA   ($00),Y
650E-   C8          INY
650F-   D0 F7       BNE   $6508
6511-   E6 01       INC   $01
6513-   A5 01       LDA   $01
6515-   C9 60       CMP   #$60
6517-   90 EF       BCC   $6508
6519-   60          RTS
```

That also wasn't there a minute ago. It
looks like some kind of rudimentary
decryption routine for $4000..$5FFF. I
bet it gets called later.

At any rate, let's save this to my non-
working copy. The easiest way is
through an intermediate work disk that
won't overwrite the precious data+code
at $6000..$65FF.

```
[S6,D1=my work disk]

; reboot (preserves main memory)
*C600G
...

[S6,D1=non-working copy]

; save the data+code from track 3.5 to
; a regular file
]BSAVE OBJ.6000,A$6000,L$600

; patch the boot program to load that
; regular file instead of calling the
; routine that originally read it from
; track 3.5
]LOAD GOODBYE
]20 PRINT D$;"BLOAD OBJ.6000"
]UNLOCK GOODBYE
]SAVE GOODBYE
]LOCK GOODBYE

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

Quod erat liberandum.

Acknowledgments

The original disk came to me courtesy
of The Mitch Kapor Archive.