

MCMURPHY'S MANSION
Written by David Martin
Crack Study

Tools used (Modern):

VICE
DirMaster
Visualize1541
G64Conv

Tools I would have used in 1986:

C64
Super Snapshot
Disector <https://csdb.dk/release/?id=74829>
The GCR Editor <https://csdb.dk/release/?id=67858>

Starting with a G64, a direct conversion to D64 produces a non-working copy. Inspection reveals that after an initial load, the program settles in to normal disk access. At this point it is clear that there are two paths to a working crack:

1. Simply snapshot the initially loaded data and bypass the custom loader entirely, or
2. Crack the custom loader.

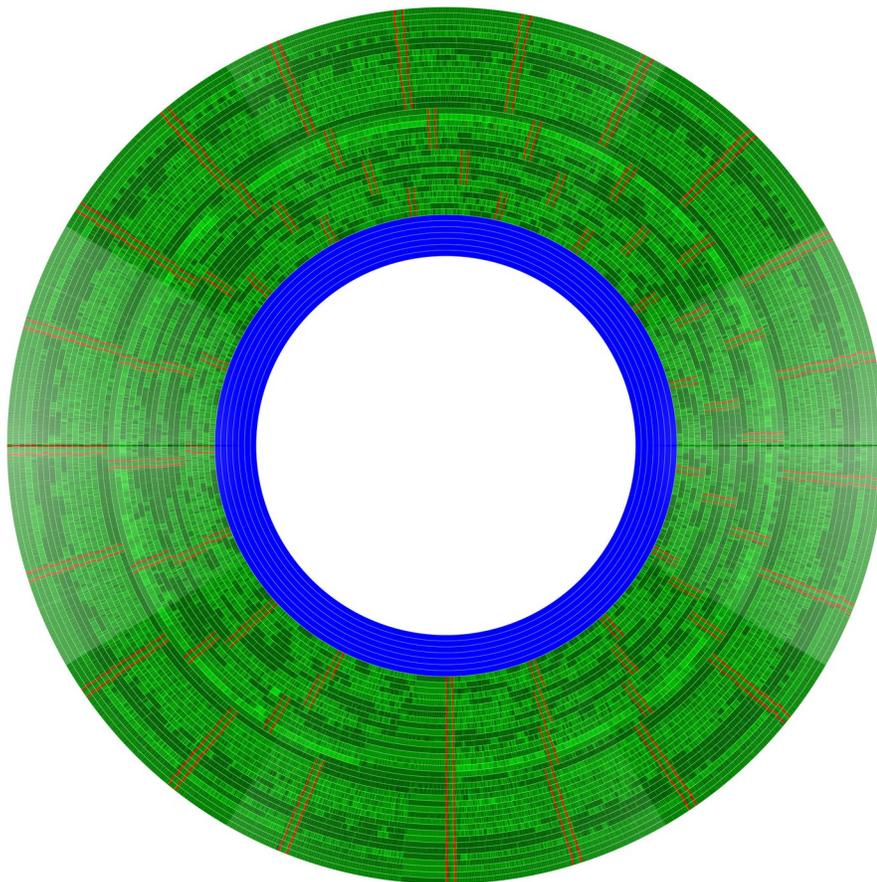
I chose 2. because it's more interesting. In the end, of course, after defeating the custom loader, the file will be extracted anyway.

Modeling the disk with a forensics tool, it becomes clear that there is data on those tracks, but when trying to graph it linearly using a sector index it is missing. This is clue #2.

Visualize-1541 Block Imaging



Visualize-1541 Disk Imaging

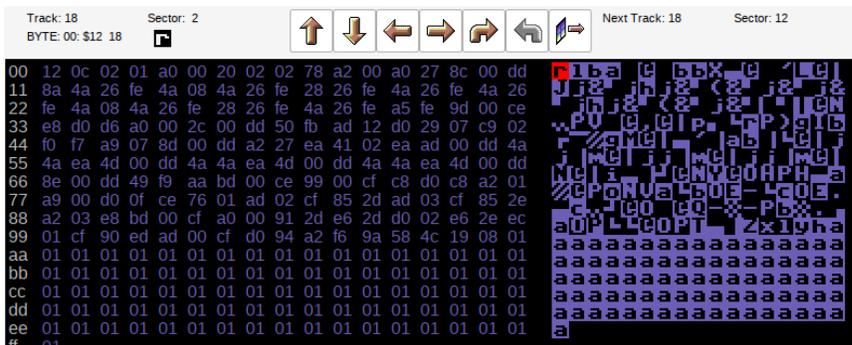


So enough of the forensics, let's have a look at what's actually going on.

The disk loads an initial BASIC stub. This stub loads and displays two koala pics with an asm routine up at \$c002 for copying the screen/color data in, nothing much to see there. It finishes with a load"main",8,1. It is completely possible to boot this game by simply loading "main".

```
1 sc=0:bo=0:ch=12:printchr$(147):printchr$(14)
2 poke53280,0:print:next
3 tab(15)="LOADING"
4 ifb=3thenpoke53280,0:sys49154:goto100
5 ifb=1thenpoke53280,3:b=2:sys49154
6 ifb=2thena$="pic b plaid
7 :f$=chr$(129)+left$(a$,14)
8 ifb=2thenb=3:loadf$,8,1
9 fora=49154to49317:readi:pokea,i:next
10 a$="pic a logo
11 f$=chr$(129)+left$(a$,14)
12 b=1:loadf$,8,1
13 end
14 fori=1to8000:next
15 poke53280,0
16 poke53265,peek(53265)and223
17 poke53272,(peek(53272)and240)or4:poke53270,200
18 print"@";poke53281,sc:poke53280,bo:poke53281,bo
19 printchr$(147):printchr$(14):fori=1to5:print:next:printtab(7)"The loading continues..."
20 print:print:print:print:print:print" (please wait)"
21 load"main",8,1
22 data173,14,200,41,239,141,17,200
```

A look at main itself reveals that even though it is 114 blocks on the disk, it only loads two blocks 18/2 and 18/12. Looking at the raw sector data for these:



There's an interesting message in here about the copy protection – I'd assume this was one of these protection kits that was available from the back of computer magazines in the 1980s.



If you wondered where the drive-side code block itself comes from (not the initial stub, but the custom loader itself), it can be found at 18/18



C64-side of loader code

```

    *=$0102
    ldy #$00
    jsr EXECUTE_DRIVE_CODE
    sei
    ldx #$00
L10a
    ldy #$27
    sty $dd00
    txa
    lsr a
    rol $fe
    lsr a
    php
    lsr a
    rol $fe
    plp
    rol $fe
    lsr a
    rol $fe
    lsr a
    rol $fe
    lsr a
    php
    lsr a
    rol $fe
    plp
    rol $fe
    lsr a
    rol $fe
    lda $fe
    sta $ce00,x
    inx
    bne L10a

    L134
    ldy #$00
L136
    bit $dd00
    bvc $0136

    L13b
    lda $d012
    and #$07
    cmp #$02
    beq L13b
    lda #$07
    sta $dd00
    ldx #$27
    nop
    eor ($02,x)
    nop
    lda $dd00
    lsr a
    lsr a
    nop
    eor $dd00
    lsr a
    lsr a
    nop
    eor $dd00
    lsr a
    lsr a
    nop
    eor $dd00
    stx $dd00
    eor #$f9
    tax
    lda $ce00,x
    sta $cf00,y
        iny
    bne L13b
    ldx #$01

    L176 = *+1
    lda #$00
    bne L188
    dec L176
    lda $cf02
    sta $2d
    lda $cf03
    sta $2e
    ldx #$03
    inx
    lda $cf00,x
    ldy #$00
    sta ($2d),y
    inc $2d
    bne L196

    inc $2e
L196
    cpx $cf01
    bcc L188

    lda $cf00
    bne L134

    ldx #$f6
    txs
    cli
    jmp $0819

    *=$0202
    lda $ba
    jsr $ffb1
    lda #$ff
    jsr $ff93

L20c
    lda COMMAND_DATA,y
    jsr $ffa8
    iny
    cmp #$0d
    bne L20c
    jmp $ffae

    *=$021a
    COMMAND_DATA
        .text "M-E"
        .byte $a8,$07,$0d
```

1541-side of loader code

```

BUFFER_0_COMMAND_STATUS = $00
BUFFER_1_COMMAND_STATUS = $01
BUFFER_2_COMMAND_STATUS = $02
BUFFER_3_COMMAND_STATUS = $03
BUFFER_4_COMMAND_STATUS = $04
BUFFER_0_TRACK = $06
BUFFER_1_TRACK = $08
BUFFER_2_TRACK = $0a
BUFFER_3_TRACK = $0c
BUFFER_4_TRACK = $0e
BUFFER_0_SECTOR = $07
BUFFER_1_SECTOR = $09
BUFFER_2_SECTOR = $0b
BUFFER_3_SECTOR = $0d
BUFFER_4_SECTOR = $0f
UNIT_0_CURRENT_TRACK = $22
CURRENT_BUFFER_PTR = $30
CURRENT_BUFFER_LO = $30
CURRENT_BUFFER_HI = $31
BUFFER_0_RAM = $0300
BUFFER_1_RAM = $0400
BUFFER_2_RAM = $0500
BUFFER_3_RAM = $0600
BUFFER_4_RAM = $0700
SERIAL_BUS = $1800
DISK_DATA_IN = $1C01
INITIALIZE_COMMAND = $d005
FIND_DATA_BLOCK_START = $f50a
DECODE_69_GCR_BYTES = $f8e0
CONTROLLER_ERROR = $f969
    *=$0400
START
    lda #$03
    sta CURRENT_BUFFER_HI
    lda UNIT_0_CURRENT_TRACK
    cmp BUFFER_1_TRACK
    bne ERROR_ROUTINE
    jsr FIND_DATA_BLOCK_START
READ_WAIT_0
    bvc READ_WAIT_0
    clv
    lda DISK_DATA_IN
    sta (CURRENT_BUFFER_PTR),y
    iny
    bne $040d
    ldy #$ba
    read_wait_1
        bvc READ_WAIT_1
        clv
        lda DISK_DATA_IN
        sta $0100,y
        iny
        bne READ_WAIT_1
        jsr DECODE_69_GCR_BYTES
        lda BUFFER_0_RAM+$01
        sta BUFFER_1_SECTOR
        lda BUFFER_0_RAM
        sta BUFFER_1_TRACK
        beq WRITE_TO_C64_LOOP
        lda #$ff
        sta $0301
        ;Now send to the c64
        WRITE_TO_C64_LOOP
        RAM_READ_SOURCE = *+1
        lda BUFFER_0_RAM
        sta $14
        lsr a
        lsr a
        lsr a
        lsr a
        tax
        lda #$01
        sta SERIAL_BUS
        SERIAL_WRITE_WAIT
        bit SERIAL_BUS
        bne SERIAL_WRITE_WAIT
        stx SERIAL_BUS
        txa
        asl a
        and #$0f
        sta SERIAL_BUS
        lda $0014
        and #$0f
        sta SERIAL_BUS
        asl a
        and #$0f
        nop
        sta SERIAL_BUS
        inc RAM_READ_SOURCE
        bne WRITE_TO_C64_LOOP
        ;do some finalization dance
        lda #$0c
        sta SERIAL_BUS
        lda $08
        bne START
        lda #$00
        sta SERIAL_BUS
        lda #$01
        bne ERROR_OUT
        ERROR_ROUTINE
        lda #$00
        ERROR_OUT
        jmp CONTROLLER_ERROR
        NEXT_READ
        lda #$e0
        sta BUFFER_1_COMMAND_STATUS
        BUFFER_1_CMD_WAIT
        lda BUFFER_1_COMMAND_STATUS
        bmi BUFFER_1_CMD_WAIT
        beq NEXT_READ
        jmp INITIALIZE_COMMAND

```

Initially-invoked drive-side stub

```

*=$07a8
LDA #$0C
STA $1800
LDA #$0F
STA $08
LDA #$02
    STA $09
    LDA #$12
    STA $0A
    STA $0B
    LDA #$80
    STA $02
    READYWAIT
    LDA $02
    BMI READYWAIT
    LDY #$00
COPYLOOP
    LDA $0500,Y
    STA $0400,Y
    INY
    BNE COPYLOOP
    JMP $0483

```


Looking at the flux graphs from our initial investigation it's clear they're not using a weird sync setup to have way more sectors than you'd expect or anything of that sort, the track looks normally formatted from that perspective. This is clue #3. At this point I suspect that they've simply modified the sector headers to have oddball sector numbers. That would explain why most D64 tools show this area as empty while the flux analysis shows it as a "normal" looking area. The D64 tools are rigidly indexed and expecting normal sector numbering. Dumping the g64 with nibconv confirms this:

NIBCONV track 12 data

```

track 12
  speed 3
  begin-at 5
  sync 43
  ; header
  gcr 08
  begin-checksum
    checksum 8c
    ; sector
    gcr 82
    ; track
    gcr 0c
    ; id2
    gcr 30
    ; id1
    gcr 32
  end-checksum
  gcr 0f
  gcr 0f
  ; Trk 12 Sec 130
  bytes 55 55 55 55 55 55 55 55 7f
  bits 11
  sync 38
  ; data
  gcr 07
  begin-checksum
    gcr 0c 8c 04 e2 04 89 11 89 0b 8a 00 e2 04 80 d0 db 4c 80 d0 db 92 aa 8a 02 2b c7 2b 81 82 a2 04
8a 00 2a 06 89 0b a0 08 81 2b 06 89 12 80 a2 04 8a 00 2a 06 4f 05 81 d0 db 90 db 81 2b 4f 18 90 f7 81
01 d0 f7 89 12 80 8a 00 e2 04 00 99 41 24 3a 80 d0 db 4c 80 d0 db 92 aa 8a 02 2b 38 2b 81 82 a2 04 8a
00 2a 06 89 0b a0 08 81 2b 06 89 12 84 a2 04 8a 00 2a 06 4f 05 81 d0 db 90 db 81 2b 4f 18 81 84 89 12
84 a2 04 e2 04 89 12 84 8a 00 e2 04 00 99 41 24 3a 4c 92 aa 8a 02 2b 5d 2b 90 56 89 0d 2b 06 89 0d 89
0b a2 04 8a 00 2b 06 4f 1d 00 99 22 d4 48 45 53 45 20 57 4f 4e 27 54 20 48 45 4c 50 20 59 4f 55 21 22
3a 4c 80 d0 db 92 aa 8a 02 2b c7 2b 89 0d 80 a2 04 8a 00 2a 06 4f 12 90 f7 81 01 d0 f7 89 0d 80 8a 00
e2 04 81 d0 db 90 db 81 2b 4f 0b 00 99 41 24 3a 80 d0 db 4c 80
    checksum 1f
  end-checksum
  gcr 00
  bits 0101001000
  bytes a9 55 55 55 7f
  bits 11

```

As you can see, all they've done here is customize the sector headers to have goofy sector numbers. Any copier based on normal disk geometry would fail to copy this disk.

So, how to crack? Simple – first make a chart of which “fake” sector numbers correspond to which “real” sector numbers. For this I used nibconv and modified the resulting text file. This gave me this mapping:

```
Track 12  82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f 90 91 92 93 94 80 81
-----  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14

Track 13  85 86 87 88 89 8a 8b 8c 8d 8e 8f 90 91 92 93 94 80 81 82 83 84
-----  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14

Track 14   8c 8d 8e 8f 90 91 92 93 94 80 81 82 83 84 85 86 87 88 89 8a 8b
-----  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14
```

Converting that back to a G64, I was then able to see sector data in DirMaster, and correct the T/S links using these tables.

FINALLY, to extract the real main file, I changed the T/S link in its directory entry to point to 15/2 where its sector chain begins.

At this point you have a working copy that can be directly converted to D64 and copied through any T/S based copier.