

Ships Ahoy

Unicorn © 1983 for Commodore 64

Cracked by Mr.Mouse/XeNTaX/Genesis Project

Released November 17, 2024 by Genesis Project

Copy protection report

1. Disk content:

```
0  "TITL" "EGG" "AHoy" " " 38 20
1  "UNICORN" PRG
3  "START" PRG
6  "START2" PRG
33  "PICUNI" PRG
34  "SCRUNI" PRG
42  "ATTUNI" PRG
32  "PICCLC" PRG
35  "ATTCLC" PRG
36  "SCRCLC" PRG
66  "MENU" PRG
17  "C-PAGE" PRG
39  "WIDEFONT.A800" PRG
39  "BOAT.SET.8400" PRG
39  "BOAT.SET.A000" PRG
39  "TITLE.SET.B000" PRG
39  "MAZE.SET.A000" PRG
39  "EQUATE.SET.A000" PRG
39  "MAZEREW.SET.A000" PRG
47  "EQUATE.BAS" PRG
39  "EQUATE.SET.REVER" PRG
33  "SHIPS_LOADER" PRG
33  "BOATMATH.BAS" PRG
35  "ATTMAZ" PRG
10  "DRAW" PRG
7  "SCRMZ" PRG
32  "MAZEREW.BAS" PRG
32  "PICDRW" PRG
17  "BLOCK.OBJ2" PRG
35  "ATTDRW" PRG
35  "SCRDRW" PRG
32  "PICMSWP" PRG
35  "ATTMSWP" PRG
32  "SCRMSWP" PRG
32  "PICMAZ" PRG
32  "PIC S-A" PRG
35  "ATT S-A" PRG
35  "SCR S-A" PRG
33  "MINE_LOADER" PRG
33  "MAZE_LOADER" PRG
33  "DRAW_LOADER" PRG
20  "MAZE.BAS" PRG
144 BLOCKS FREE.
```

2. Copy protection summary

- Expects a checksum error of Track 1 Sector 1
- Expects a correct checksum of Track 1 Sector 2
- Checks correct memory operation
- Localizes (in terms of memory offset) itself before operation

3. Crack changes

The game is part basic, part machine code. Basic source as well as machine code sources changed to allow the following:

- Title pictures, game title pictures each unified into one file
- Title pictures and main cracked game in one file
- Main game menu newly build in machine code and resident in main file
- Fast-loader installed
- Individual source files per sub game compressed and all fo them resident in main file, unpacked when needed
- Loading only needed for sub games and sub game title pictures
- Original game took 520 blocks (133.120 bytes), cracked game takes 194 blocks (49.664 bytes), only 37% of original size



4. Copy protection source code and comments

```
; Ship Ahoy by Unicorn Software 1983
; Copy-protection - filename "START"
; Crack by Mr.Mouse, October 2024
;-----
; Protection steps:
;
; - Unicorn file loads "START"
; - Unicorn file runs it using sys 52224+16+7
;
; - Code at cc17:
; - Entry point at $cc17 has a jump to cbe0
;
; - Code at cbe0:
;   > replace jmp at cc17 with jsr $ffe7 (kernel for close all channels and
;   files); this will now be the loader start address when called from basic
;   > jsr to the code at cd00 that checks if sector 1, track 1 has a checksum
;   error
;   > if yes, the program will return here and a jump to cc17 will be taken
;
; - Code at cc17:
;   > Will load "START2" basic file
;   > Will ensure 0843/0844 is pointing to $0849
;   > Will then set command buffer at 0277 to "run" + cr
;   > Sets keyboard buffer length to 4
;   > Finds end of loaded basic file "START2" and sets $2d/$2e, $2f/$30 and
;   $31/$32 to this to set basic pointers
;   > Executes basic warms start via vector at 0300 -> $e38b
;
; - Code at cd00 (Copy-protection):
;   > Sets 7fff to RTS ($60), JSRs there, upon return sets x to stack pointer,
;   gets the hi byte of the current page ($cd)
;   > Modifies 7 places in the code to $cd, to localize the code to where it was
;   loaded (and perhaps part of first protection)
;   > Some sort of memory check at $9000: loads y with whatever is there,
;   increases y, stores that at $9000 again and then compares y with $9000.
;   If not same it will crash.
;   > Will then proceed to check two sectors on track 1. Sector 2 and sector 1.
;   Sector 2 should give no error (or else crash), sector 1 should have
;   checksum error (or else crash)
;   > If that is all correct, resets the drive and returns with the address
;   $cdee in A/Y
;   > Upon failure the program will fill cd3b downward to 0000 with $cd
;
; - START2 will load title pictures, show them, play some music, and then reload
;   "START" to do another copy protection check.
;   But will modify the file to load to MENU and adjust the size of the filename
;   string to 4, before running it
;
; - After that, the routine at cc17 will be used to load new basic games from
;   the menu (without further Copy-protection checks), but other files are also
;   loaded via the basic programs.

* = $cbe0
;-----jumped to from cc17
1cbe0    lda #$20          ; replace jmp $cbe0 into jsr $ffe7 --> close all channels and files
        ldy #$e7
        ldx #$ff
        sta 1cc17
        sty $cc18
        stx $cc19
        jsr 1cd00          ; check the error track existence, return with a = ee and y = cd
(cdee)   jmp 1cc17          ; jump down again
;-----
1cc02    ldx #$04
        lda $cc8c,x
        cmp $03f4,x
        bne 1cc10
        dex
        bpl 1cc02
        jmp 1cc17
1cc10    lda #$35
        sta $01
        jmp $a000
;-----start (sys from UNICORN)
1cc17    jmp 1cbe0          ; this will be replaced above by jsr $ffe7 --> close all channels
and files    lda #$01          ; open channel 1, device 8
```

```

ldx #$08
ldy #$01
jsr $ffba ; open
lda #$06 ; set filename to "START2"
ldx #$8c
ldy #$cc
jsr $ffbd ; set fn
lda #$00 ;
jsr $ffd5 ; load the file
lda #$49 ; set 0843/0844 to 0849
sta $0843
lda #$08
sta $0844
jsr $ffe7 ; close all channels and files
ldx #$03 ; "type" run in command buffer
1cc40 lda $cc88,x
sta $0277,x
dex
bpl 1cc40
lda #$04 ; set accu to 4
sta $c6 ; store at c6 --> Length of keyboard buffer to 4
lda #$00 ; set fb/fc to $0800
sta $fb
lda #$08
sta $fc
1cc55 ldy #$00 ; or stuff: 00, with 43, with 08 --> 4b etc. all the way until the
end of the (basic) file
lda ($fb),y
iny
ora ($fb),y
iny
ora ($fb),y
beq 1cc6a ; there are 3 0's at the end of a basic file. when this routine hits
those, then the end of the file was found, needed to set variables below
inc $fb
bne 1cc55
inc $fc
jmp 1cc55
1cc6a lda $fb ; load the lo byte (a5)
clc
adc #$03 ; add 3 more (to get to real end of the file, oda8)
sta $fb
bcc 1cc75
inc $fc
1cc75 lda $fb ; set pointer to beginning of variable area and pointer to beginning
of array variable area.
sta $2d
sta $2f
sta $31 ; Pointer to end of array variable area, all to $0da8
lda $fc
sta $2e
sta $30
sta $32
jmp ($0300) ; Execution address of warm reset, displaying optional BASIC error
message and entering BASIC idle loop. Default: $E38B.
;-----; "RUN"
??? ;%01010010 'r'
eor $4e,x

;-----;"START2" or other filename to load
lda $5453
eor ($52,x)
??? ;%01010100 't'
??? ;%00110010 '2'
brk
asl a
jsr $ffe7
lda #$35
sta $01
jmp ($f1ff)
jsr $ffe7
;-----
1cd00 lda #$60 ; set 7fff to RTS
sta $7fff
jsr $7fff ; jsr there (and back)
tsx ; store current stack pointer to x
lda $0100,x ; load that pointer ($cd) there
sta $fc ; and make fb/fc a pointer to $cd00
ldx #$00
stx $fb
ldy #$56 ; set y to $56
sta ($fb),y ; set $cd56 to $cd
ldy #$6b
sta ($fb),y ; set $cd6b to $cd
ldy #$7d
sta ($fb),y ; set $cd7d to $cd
ldy #$82
sta ($fb),y ; set $cd82 to $cd
ldy #$85
sta ($fb),y ; set $cd85 to $cd
ldy #$9c
sta ($fb),y ; set $cd9c to $cd
ldy #$b9

```

```

        sta ($fb),y    ; set $cdb9 to $cd
        ldy $9000      ; get y from $9000
        iny           ; y=y+1
        sty $9000      ; store y at $9000
        cpy $9000      ; compare y with $9000
        beq lcd49      ; if the same then go on down
lcd3a    ldy #$3b       ; set cd3b downward to 0000 to cd
lcd3c    sta ($fb),y
        dey
        bne lcd3c
        dec $fc
        bne lcd3c
        dec $fc
        bne lcd3c
lcd49    jsr $fff90     ; control kernal messages
        lda #$00
        jsr $fffc      ; close input and output channels
        lda #$03       ; set the filename --> 3 bytes --> "I/O"
        ldx $dd        ; get it from $cddd
        ldy #$65       ; will change to $cd by top -> cddd
        jsr $ffbd      ; set filename to "I/O"
        lda #$0f       ; command port
        ldx #$08       ; device
        ldy #$0f       ; command 15
        jsr $ffba      ; set logical, first and second addresses
        jsr $ffc0      ; open the file
        lda #$01       ; new filename -> 1 byte "#"
        ldx #$dc
        ldy #$65       ; will change to $cd by top -> cddd
        jsr $ffbd      ; set filename/byte out to 1541 to "#"
        lda #$03       ; logical, first and second to 3, 8 and 3
        ldx #$08       ; device 8
        ldy #$03
        jsr $ffba      ; send to 1541
        jsr $ffc0      ; open the file
        jsr $65b0      ; will change to $cd by top -> cdb0 -> jsr $cdb0 --> Try to load
track 1, sector 2. This should give an error
        bne lcd3a      ; if this anything other than 0 then crash
        dec $65eb      ; this will change to $cdeb and lower than ("2" to "1") sector 2 to
sector 1
        jsr $65b0      ; this will change to $cdb0
        beq lcd3a      ; we should get an error on this track (23: "Checksum error in
data"),if not we crash
        jsr $fffc      ; close input and output channels
        ldx #$0f       ; open channel for output
        jsr $ffc9
        lda #$49       ; send "I" , which is a "reset drive"
        jsr $ffd2
        lda #$0d
        jsr $ffd2
        jsr $65c2      ; will change to $cdc2 - ensure reset was done
        lda #$0f       ; close logical file f
        jsr $ffc3
        lda #$03       ; close logical file 3
        jsr $ffc3
        jsr $ff8a      ; restore default I/O vectors
        lda #$ee
        ldy $fc        ; load y with $cd --> cdee
        bne lcdf3      ; auto-jump o cdf3 -> and return
;-----
; Try to load track 1, sector at cdeb (starts off as "2", then is
lowered to "1". This should give no error on sector 2, but an error on sector 1)
        ldx #$0f       ; open channel for output
        jsr $ffc9
        ldy #$00       ; prepare to send "U1:3 0 01 02"
lcdb7    lda $65e0,y    ; will change to cde0 --> "U1:3 0 01 02" (null-terminated string,
with CR at the end before 0)
        beq lcdc2      ; if 0 then end reached , go on down
        jsr $ffd2      ; send byte to 1541
        iny
        bne lcdb7      ; keep doing it until 0 in string reached
lcdc2    jsr $fffc      ; close input and output channels. The command sent will try to load
sector 2 at track 1 to buffer 3
        ldx #$0f       ; open channel for input
        jsr $ffc6
        jsr $ffcf      ; input character from channel
        and #$0f       ; isolate lo nibble
        pha           ; push it
lcd0     jsr $ffcf      ; input character from channel
        cmp #$0d       ; wait until CR gotten
        bne lcd0       ; not yet CR? Get another character
        jsr $fffc      ; close input and output channels
        pla           ; pull error message
        rts           ; return
        ???           ;%00100011 '#'
;-----
        eor #$2f       ; "I/O"
        ???           ;%01001111 'o'
        eor $31,x
        ???           ;%00111010 ':'
        ???           ;%00110011 '3'
        jsr $2030
        bmi $ce1a
        jsr $3230

```

```
;-----;
    lda #$ff
    sta $91
    rts
;-----;
1cdf3    rts
         nop
         sta $0328
         sty $0329
         cli
         nop
         nop
         rts
         nop
```