

C64 cracking

"Slug"

(c) 1988 Alternative Software

Things you will need:

1. The tape original of "Slug" (c) 1988 Alternative Software.
2. A C64, or a copy of VICE emulator.
3. C64 Action Replay cart (link: https://rr.pokefinder.org/rrwiki/images/7/78/Action_Replay_binaries_rr.c64.org_2013-01.rar)
4. All_About_Your_64 by Ninja/The Dreams (reference for all things C64).
5. East Linker (link: <https://csdb.dk/release/?id=27297>)
6. Cruel Cruncher (link: <https://csdb.dk/release/?id=47768>)

Before we begin, please read the following disclaimer:

I am a novice at cracking C64 software. Everything I say on the subject should be ignored, or at the very least questioned :P

With that out of the way, for our first foray into the wonderful world of C64 cracking, lets start with the simple tape loader on "Slug".

Tape? Yes, the biggest difference between the Amiga and the C64 was that 95% of games were released on tape in order to save money.

Of course some games were also released (at a slightly higher price) on disk, but due to the huge number of budget titles on 8-bit, most games were released on tape only.

In practical terms, for a cracker, there isn't a lot of difference - you still have to understand a loader, rip the files, replace the loader (if it's a multi-load game, which this is *NOT*), etc.

Insert the "Slug" tape original, and lets get started!

Because if we start the tape in the usual way ('LOAD ""',1') it will auto-start, we need to find a way to load that lets us inspect it before any code executes.

The answer is found in the KERNAL section of AAY64 (see "Things you will need #4" above):-

```
$F72C/63276:    Find Any Tape Header
```

In order to call a C64 KERNAL function, we use the SYS command with the decimal address:-

SYS 63276

Press play and the tape will load for a few seconds then return control to us. Now it's time to investigate a particular area of memory:-

\$033C-\$03FB/828-1019: Tape I/O Buffer

We'll do this using the Action Replay, so press the magic button (Alt-Z in VICE) and enter the monitor:-

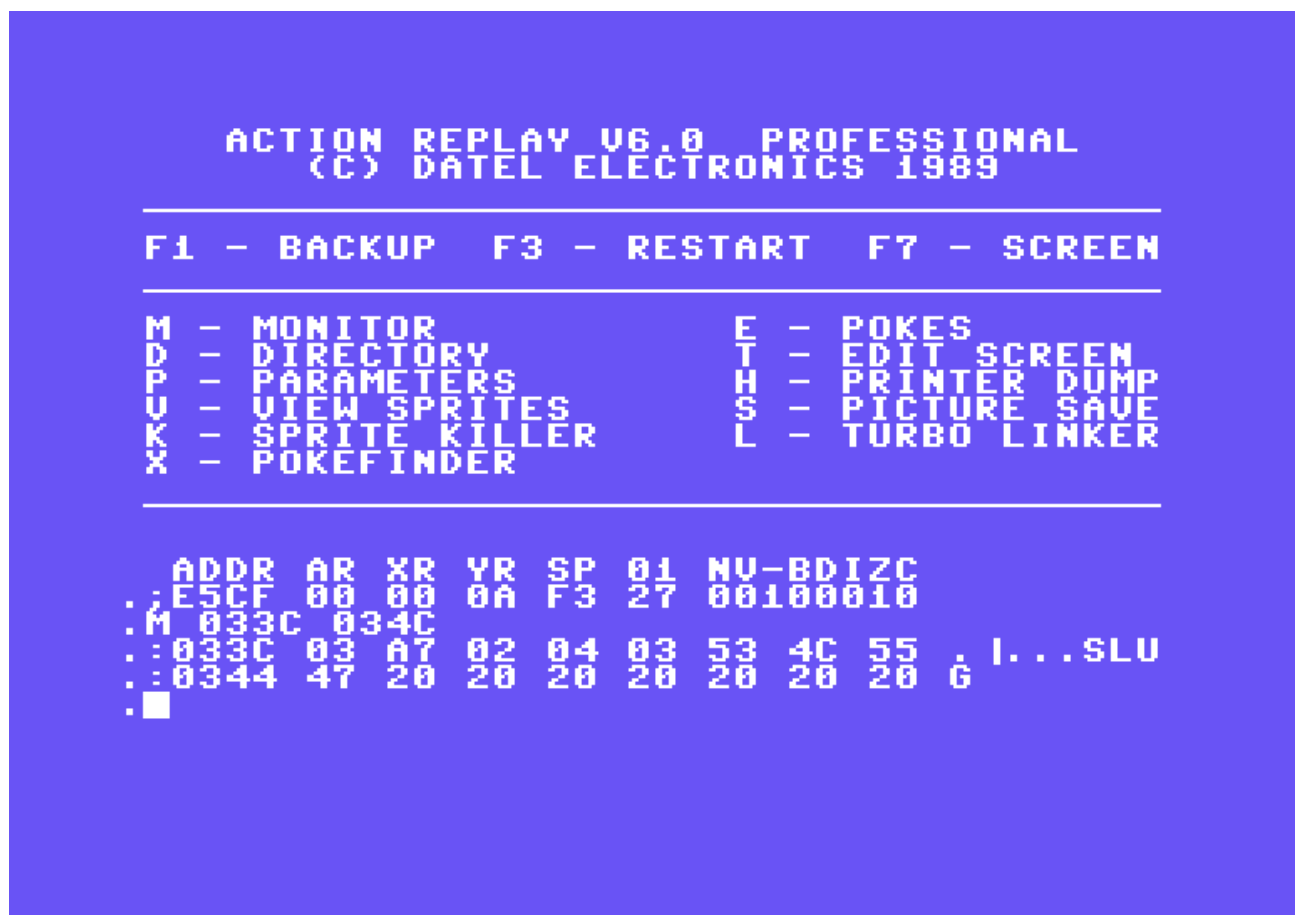


Illustration 1: modify tape load address

The bytes at \$033C are important to us...

\$033C: program type, \$03 = machine code

\$033D: load address start low, \$A7

\$033E: load address start high, \$02

\$033F: load address end low, \$04

\$0340: load address end high, \$03

So it loads code from \$02A7 to \$0304, we need to modify this address before resuming loading. Simply move the cursor using the arrow keys to the byte @ \$033E and change it to \$12 instead of \$02, then do the same for the byte @ \$0340 but change it to \$13 from \$03. Now when we resume, the tape loader will place the code \$1000 bytes higher in memory, and we can take a look at it before execution starts!

BTW: 16-bit values are stored as low,high on C64, keep this in mind as we disassemble the loader!

Exit AR (with X then F3) and get back to C64 basic - now it's time to load the file "SLUG" we found in the tape header that we modified.

Time for another SYS command:-

```
SYS 62828
```

This value is \$F56C, which is midway through the KERNAL "load file from tape" routine - skipping over all the tape-header loading etc. which we have already done (+ modified!).

Let the short load complete, and now we can disassemble the code and see what it does - back into the AR monitor:-

```

.D 12A7 12D7 JSR $03C4
.> 12A7 20 C4 03 LDA #$00
.> 12AA A9 00 STA $60
.> 12AC 85 60 LDY #$0C
.> 12AE A0 0C STY $61
.> 12B0 84 61 INY
.> 12B2 C8 INY
.> 12B3 C8 JSR $02BA
.> 12B4 20 BA 02 JMP ($0C00)
.> 12B7 6C 00 0C STA $62
.> 12BA 85 62 STY $63
.> 12BC 84 63 LDA #$05
.> 12BE A9 05 STA $01
.> 12C0 85 01 LDA #$02
.> 12C2 A9 02 STA $0365
.> 12C4 8D 65 03 LDA #$1F
.> 12C7 A9 1F STA $DC0D
.> 12C9 8D 0D DC LDA $DC0D
.> 12CC AD 0D DC LDA #$68
.> 12CF A9 68 STA $DC04
.> 12D1 8D 04 DC LDA #$03
.> 12D4 A9 03 STA $DC05
.> 12D6 8D 05 DC
.■

```

Illustration 2: Disassemble tape loader

```

12A7 20 C4 03 JSR $03C4
12AA A9 00 LDA #$00
12AC 85 60 STA $60 ;#$00 to ($60)
12AE A0 0C LDY #$0C
12B0 84 61 STY $61 ;#$0C to ($61)
12B2 C8 INY
12B3 C8 INY ;Y = $0E, A = $00
12B4 20 BA 02 JSR $02BA ;this must be loader...
12B7 6C 00 0C JMP ($0C00) ;...since this instruction
references memory which is currently empty!

```

Note that the final JMP instruction is an indirect jump to the address stored @ (\$0C00), and not a jump directly to code @ \$0C00.

Obviously we don't want it to execute that JMP (\$0C00), because the game will continue to load and we'll lose control - let's patch it!

```

.D 12A7 12BA JSR $03C4
.V 12A7 20 C4 03 LDA #$00
.V 12AA A9 00 STA $60
.V 12AC 85 60 LDY #$0C
.V 12AE A0 0C STY $61
.V 12B0 84 61 INY
.V 12B2 C8 INY
.V 12B3 C8 JSR $02BA
.V 12B4 20 BA 02 JMP ($0C00)
.V 12B7 6C 00 0C JMP $02B7
.V 12B7 4C B7 02
.A 12BA
.D 12A7 12BA JSR $03C4
.V 12A7 20 C4 03 LDA #$00
.V 12AA A9 00 STA $60
.V 12AC 85 60 LDY #$0C
.V 12AE A0 0C STY $61
.V 12B0 84 61 INY
.V 12B2 C8 INY
.V 12B3 C8 JSR $02BA
.V 12B4 20 BA 02 JMP $02B7
.V 12B7 4C B7 02
.T 12A7 1304 02A7
.G 02A7

```

Illustration 3: Disassemble tape loader (2)

I'll just recap what we did in the previous picture - we replaced the JMP (\$0C00) instruction with a JMP \$02B7 endless loop, then used the "T" command to transfer memory down to the original load address (where it expects to be executed from, remember this code is full of absolute addresses, nothing pc-relative!), before using the "G" command to GO \$02A7 and start the modified loader!

After some border flashing, the loader will have finished and reached our endless JMP, so we can stop the tape and examine what was loaded @ \$0C00.

Please note the tape counter (hereafter referred to as TC) when the load has completed (border stops flashing), you should be at TC=0008.

(\$0C00) simply points to \$0C02, so the code we want to examine immediately follows:-

```

.D 0C02 0C32
..> 0C02 A9 00 LDA #$00
..> 0C04 85 60 STA $60 LOAD START LOW
..> 0C06 A9 12 LDA #$12
..> 0C08 85 61 STA $61 LOAD START HIGH
..> 0C0A A9 00 LDA #$00 LOAD END LOW
..> 0C0C A0 D0 LDY #$D0 LOAD END HIGH
..> 0C0E 20 BA 02 JSR $02BA CALL LOADER
..> 0C11 A9 00 LDA #$00
..> 0C13 A2 E0 LDX #$E0
..> 0C15 85 60 STA $60 LOAD START LOW
..> 0C17 86 61 STX $61 LOAD START HIGH
..> 0C19 A9 1E LDA #$1E LOAD END LOW
..> 0C1B A0 F4 LDY #$F4 LOAD END HIGH
..> 0C1D 20 BA 02 JSR $02BA CALL LOADER
..> 0C20 A9 00 LDA #$00
..> 0C22 A2 D0 LDX #$D0
..> 0C24 85 60 STA $60 LOAD START LOW
..> 0C26 86 61 STX $61 LOAD START HIGH
..> 0C28 A9 3D LDA #$3D LOAD END LOW
..> 0C2A A0 D8 LDY #$D8 LOAD END HIGH
..> 0C2C 20 BA 02 JSR $02BA CALL LOADER
..> 0C2F A9 7F LDA #$7F
..> 0C31 8D 0D DC STA $DC0D
.
```

Illustration 4: Disassemble tape loader (3)

Now we're getting somewhere! It's easy to understand the arguments to the loader now that we can see multiple calls to it...

The loader arguments are simply:-

(\$60) = load start low

(\$61) = load start high

A = load end low

Y = load end high

If we go back and apply our newfound knowledge to the first load (where we modified a JMP (\$0C00) to JMP FOREVER), we can see the load went from \$0C00 to \$0E00. This matches what we can observe in memory now, so we can be certain we have correctly identified the arguments.

Lets make a quick list of where files are loaded:-

start=\$1200, end=\$D000

start=\$E000, end=\$F41E

start=\$D000, end=\$D83D

Nothing is executed between these loads, so we can safely let the game load these files into memory for us, we just need to find a suitable place to hook.
Let's continue disassembling the code following the 3 loads:-

```

.> 0C0E 20 BA 02 JSR $02BA
.> 0C11 A9 00 LDA #$00
.> 0C13 A2 E0 LDX #$E0
.> 0C15 85 60 STA $60
.> 0C17 86 61 STX $61
.> 0C19 A9 1E LDA #$1E
.> 0C1B A0 F4 LDY #$F4
.> 0C1D 20 BA 02 JSR $02BA
.> 0C20 A9 00 LDA #$00
.> 0C22 A2 D0 LDX #$D0
.> 0C24 85 60 STA $60
.> 0C26 86 61 STX $61
.> 0C28 A9 3D LDA #$3D
.> 0C2A A0 D8 LDY #$D8
.> 0C2C 20 BA 02 JSR $02BA
.> 0C2F A9 7F LDA #$7F
.> 0C31 8D 0D DC STA $DC0D
.> 0C34 AD 0D DC LDA $DC0D
.> 0C37 A9 00 LDA #$00
.> 0C39 8D 1A D0 STA $D01A
.> 0C3C 78 SEI
.> 0C3D A9 30 LDA #$30
.> 0C3F 85 01 STA $01
.> 0C41 4C 00 D8 JMP $D800

```

PATCH TO JMP 0C41

Illustration 5: Disassemble tape loader (4)

We can clearly see a suitable candidate, the JMP \$D800 which would jump into code at the end of the 3rd file loaded here (from \$D000 to \$D83D).

Let's patch it in the same way as before, simply with an infinite jump ("A 0C41 JMP \$0C41"), before we continue loading by using "G 0C02" in the AR monitor.

Please make sure your tape is at TC=0008 before you resume execution.

After some minutes of loading, you should end up hitting our infinite jump at TC=0068. It's time to get back into the AR monitor and save our files!

But just before we save, let's check out the code we would have jumped into (at the end of file 3):-

```

.D D800 D83D
..> D800 A2 00 LDX #$00
..> D802 BD 00 D0 LDA $D000,X
..> D805 9D 00 08 STA $0800,X
..> D808 BD 00 D1 LDA $D100,X
..> D80B 9D 00 09 STA $0900,X
..> D80E BD 00 D2 LDA $D200,X
..> D811 9D 00 0A STA $0A00,X
..> D814 BD 00 D3 LDA $D300,X
..> D817 9D 00 0B STA $0B00,X
..> D81A BD 00 D4 LDA $D400,X
..> D81D 9D 00 0C STA $0C00,X
..> D820 BD 00 D5 LDA $D500,X
..> D823 9D 00 0D STA $0D00,X
..> D826 BD 00 D6 LDA $D600,X
..> D829 9D 00 0E STA $0E00,X
..> D82C BD 00 D7 LDA $D700,X
..> D82F 9D 00 0F STA $0F00,X
..> D832 E8 INX
..> D833 D0 CD BNE $D802
..> D835 4C 63 2D JMP $2D63
..> D838 01 4C QRA ($4C,X)
..> D83A 63 ???
..> D83B 2D 00 00 AND $0000
.■

```

Illustration 6: Transfer data from high -> low

This is a very standard piece of code found in many games (demos, everything really!) - it simply moves code from where it was loaded (\$D000 onwards) to another location (\$0800 onwards).

When this routine finishes, the bytes from \$D000-\$D800 will have been copied down to \$0800-\$1000, before the JMP \$2D63 returns execution to the first of the 3 loaded files (\$1200-\$D000 remember?).

Since we are going to combine the 3 files we're ripping to make a single-file game, we have no need for this copy routine, thankfully AR has an extra argument to the save routine.

Now would be a good time to insert a blank disk into the drive (device 8):-

```

.> D80E BD 00 D2 LDA $D200,X
.> D811 9D 00 0A STA $0A00,X
.> D814 BD 00 D3 LDA $D300,X
.> D817 9D 00 0B STA $0B00,X
.> D81A BD 00 D4 LDA $D400,X
.> D81D 9D 00 0C STA $0C00,X
.> D820 BD 00 D5 LDA $D500,X
.> D823 9D 00 0D STA $0D00,X
.> D826 BD 00 D6 LDA $D600,X
.> D829 9D 00 0E STA $0E00,X
.> D82C BD 00 D7 LDA $D700,X
.> D82F 9D 00 0F STA $0F00,X
.> D832 E8 INX
.> D833 D0 CD BNE $D802
.> D835 4C 63 2D JMP $2D63
.> D838 01 4C ORA ($4C,X)
.> D83A 63 ???
.> D83B 2D 00 00 AND $0000
$"$S0",8,1200,D000
$AVING $0
$"$S1",8,E000,F41E
$AVING $1
$"$S2",8,D000,D800,0800
$AVING $2
.■

```

Illustration 7: Save ripped files

By default, the files are saved as ".PRG" files (program) - these have a 2-byte header which is the load address in low,high format.

If you don't supply this when saving, the start address is used. This is correct for our first 2 files, but for the 3rd save we specify a load address of \$0800.

This way when the data is reloaded it will go to that address rather than the memory we saved it from (\$D000 onwards), which will therefore be exactly as if it had been relocated by the routine we just discarded!

Obviously we can discard the code loaded to \$0C00 to \$0E00, it is only used to load (and eventually jump to) the 3 files we just ripped.

This code would be obliterated when the data from \$D000 was relocated to \$0800 onwards, since it writes from \$0800 to \$1000.

Now that we have our files ripped, we need some way to link them together into a single file. If only there was a tool for that on C64...

It turns out there are dozens, possibly hundreds, of linkers designed for exactly this task. As you might expect, many also pack the files as they link.

However unlike most other systems, the C64 scene historically had 2 types of packing: char-packing, and crunching!

Think of char-packing as a simple RLE-like compression (fast, but low compression), and crunching more like the LZ-based packers everyone is used to.

On C64 it is extremely common to char-pack a game first (which often happens for us during linking), then crunch the resulting file with a cruncher!

Reset and load "eastlinker 1_0 .prg", remembering to insert your workdisk in device 8 (first connected drive).

Select F1 for LINKER, and choose the files as instructed:-

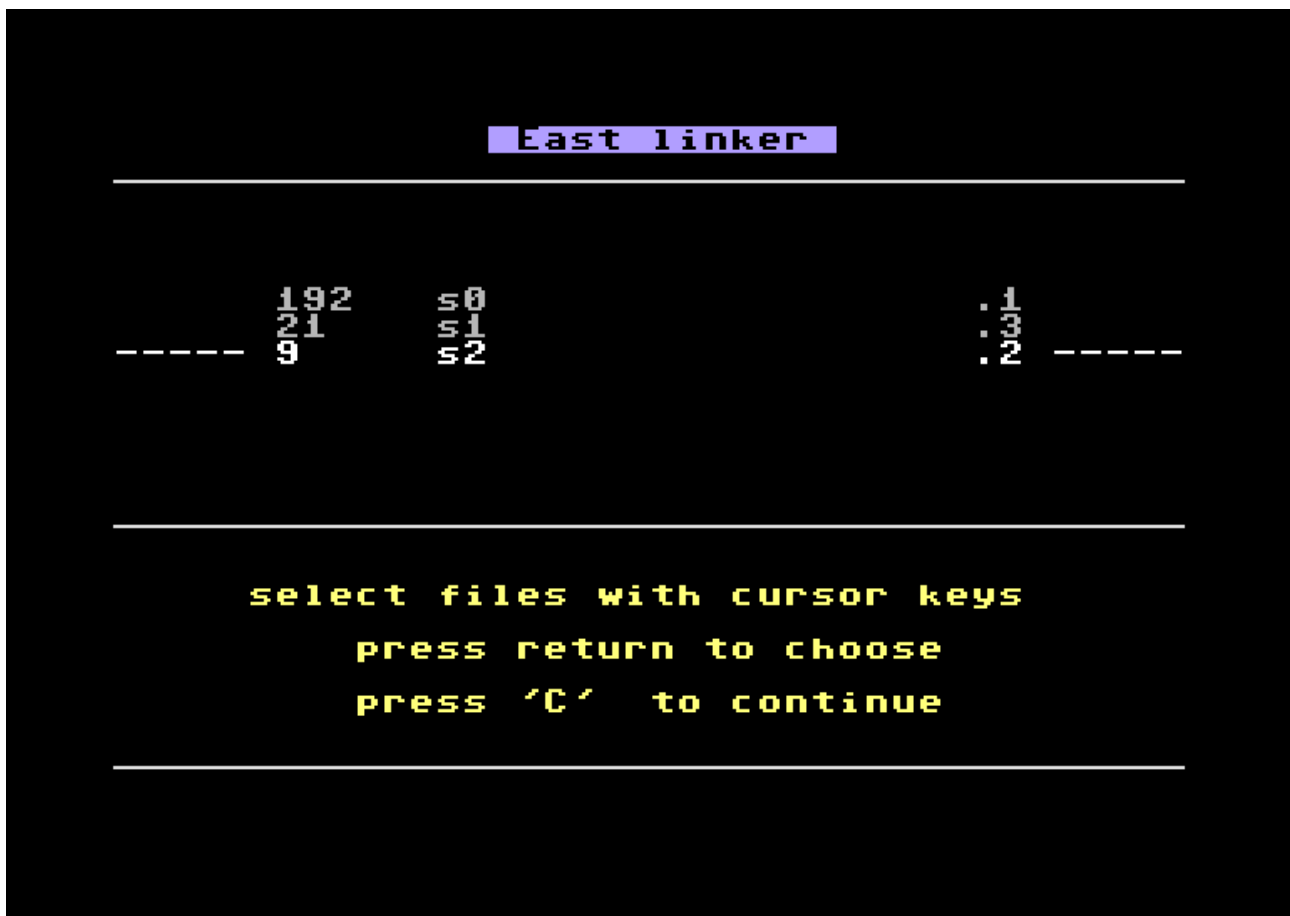


Illustration 8: Select files to link (Eastlinker)

When asked if you want to change the load address, simply press enter for "NO" (the default), and Eastlink will scan the selected files.

Once scanning has completed, you will be asked to provide some more information:-

```
SAVE NAME      :LINKTEST
START ADDRESS  $2D63
$01 STATUS     $35
SEI/CLI
```

Illustration 9: Answer the important questions (Eastlinker)

For the last option choose "SEI" (disable interrupts).

When asked to insert the target disk, you can just press space since we have enough room on our workdisk to save the linked file.

After a couple of minutes (or less, if you use VICE emulator's "warp mode" which is ALT-W) you should have a single-file game!

At this point you've successfully cracked & single-filed the game, but the resulting char-packed file is still quite large (186 blocks, a block is 256 bytes, which means it's about 46k).

Since loading this would take approximately a week, we should crunch the linked file next...

Reset + load "Best Cruelcruncher Collection.XR" and select "E" at the menu to load Chromance Cruel v3.6. Select your preferred decrunch effect from the list, then answer the cruncher's questions like so:-

```
CRUEL CRUNCHER V3.6
LOAD NAME ->LINKTEST
SAVE NAME ->CRUTEST
JUMP (HEX) :$080E
$01 VALUE :$35
SEI OR CLI? S
SPEED? HIT 1-7 OR H FOR HYPERSPEED.
```

Illustration 10: Packing the linked file (Cruelcruncher)

Choose "SEI" and crunch speed 4, then insert our "Slug" workdisk when requested. Now go and do something else for at least 45 minutes (really, crunching files on the C64 is a slow process). If you're working in VICE emulator, warp mode is your friend (Alt-W remember) - although it still takes a couple of minutes on my ancient PC...

When the crunching is complete and it asks for a target disk, you can just press space since there is sufficient free space available on our workdisk.

That's it - test the resulting file by loading "crutest.prg" from our workdisk, and bask in the glory of your first C64 tape crack!

Of course you can't release it like this, people would point at you in the street and laugh... "He released Slug with no trainers or docs, it didn't even have an intro!".

But since we're 31 years late with our crack anyway, and there are several 100% cracks of it already, we'll just leave it as is :)

This was a very gentle introduction to the subject of C64 cracking, next time (possibly) we'll tackle something a bit harder...

-WK/Flashtro, August 2019