

```
-----,
      \-----
      -----) GNU poke
      --)
      --)
-----.)
```

The extensible editor
for structured binary data

Jose E. Marchesi

jemarch@gnu.org

Pass The Salt 2022



Contents

- 1 Introduction
- 2 Demo!
- 3 poke goes to the mountain
- 4 Epilogue



Motivation

Hmmm gotta to...

```
# Figure out the file offset of the text
# section in the object file.
text_off=0x$(objdump -j .text -h $objfile \
    | grep \.text | $TR -s ' ' \
    | $CUT -d' ' -f 7)
```

...

```
func_off=$(printf %s $fun | $CUT -d: -f1)
base=$(($EXPR $func_off + 0))
probe_off=$((text_off + base + offset))
...
byte=$(dd if=$objfile count=1 ibs=1 bs=1 \
    skip=$probe_off 2> /dev/null)
```



Simple Binary Editors

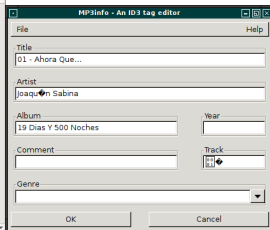
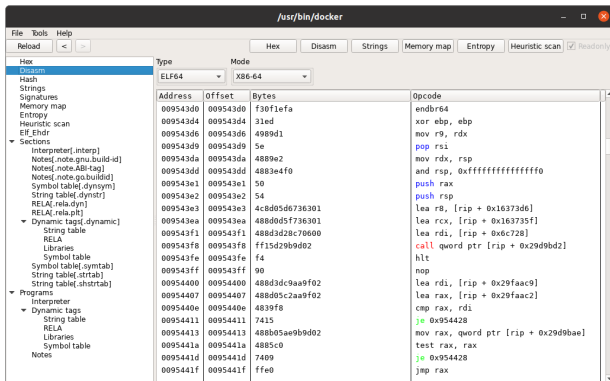
The screenshot displays the Simple Binary Editor window. The main area is divided into two panes. The left pane shows a hex dump of memory, with each line containing a hexadecimal address followed by two columns of hex bytes. The right pane shows the corresponding ASCII characters, with some non-printable characters represented by dots. Below the panes is a menu bar with the following items: Help, Save, Open, Goto, Find, Hex Addr, Hex Edit, and Quit.

```
00000000  C0 07 8E D8 B8 00 90 BE C0 89 00 01 29 F6 29 FF FC F3 A5 EA 19
00000016  00 90 90 8C C8 BA F4 3F BE D8 8E C0 8E D0 89 D4 6A 00 0F A1 BB 78
0000002C  00 64 0F B5 37 89 D7 B9 06 00 FC 65 F3 A5 89 D7 C6 45 04 12 64 89
00000042  3F 64 8C 47 02 8C C8 8E 8E E8 30 E4 30 D2 CD 13 31 D2 B9 02 00
00000058  B8 00 02 88 04 02 CD 13 73 12 50 E8 2F 01 89 E5 E8 34 01 58 30 D2
0000006E  30 E4 CD 13 EB DF 31 D2 B9 12 00 88 00 0A B8 01 02 CD 13 73 0B B1
00000084  0F 88 01 02 CD 13 73 02 B1 09 2E 89 0E CE 01 B8 00 90 8E C0 B4 03
0000009A  30 FF CD 10 B9 09 00 BB E7 00 80 D0 01 B8 01 13 CD 10 B8 00 10 8E
000000B0  C8 E8 14 00 E8 04 01 88 0A 01 E8 D8 00 EA 00 00 20 90 95 00 00 00
000000C6  09 00 8C C0 A9 FF 0F 75 FE 31 D8 8C C0 2D 00 10 3B 06 F4 01 76 01
000000DC  C3 2E A1 CE 01 28 06 C2 00 89 C1 C1 E1 09 01 D9 73 09 74 07 31 C0
000000F2  29 D8 C1 E8 09 E8 34 00 89 C1 03 06 C2 00 2E 3B 06 CE 01 75 12 B8
00000108  01 00 2B 06 C4 00 75 04 FF 06 C6 00 A3 C4 00 31 C0 A3 C2 00 C1 E1
0000011E  09 01 CB 73 AE 8C C0 80 C4 10 8E C0 31 D8 EB A3 60 60 B8 2E 0E BB
00000134  07 00 CD 10 61 8B 16 C6 00 88 0E C2 00 41 88 D5 8B 16 C4 00 88 D6
0000014A  81 E2 00 01 B4 02 52 51 53 50 CD 13 72 05 83 C4 08 61 C3 50 E8 0C
00000160  00 30 E4 30 D2 CD 13 83 C4 0A 61 EB C1 B9 05 00 89 E5 51 EB 1F 00
00000176  3A 0E 05 00 73 0F B8 45 9E 28 C8 CD 10 B0 58 CD 10 B0 3A CD 10 83
0000018C  C5 02 E8 0E 00 59 E2 DE C3 B8 00 0E CD 10 B0 0A CD 10 C3 B9 04 00
000001A2  8B 56 00 C1 C2 04 B4 0E B8 D0 24 0F 04 30 3C 39 76 02 04 07 CD 10
000001B8  E2 EB C3 52 BA F2 03 30 C0 EE 5A C3 B4 01 B7 00 B9 00 28 CD 10 C3
000001CE  00 00 0D 0A 4C 6F 61 64 69 6E 67 00 00 00 00 00 00 00 00 00 00 00
000001E4  00 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 FB 28 00 00 00 00
000001FA  00 00 00 05 5A FA B0 00 E6 70 B8 00 90 8E D8 8E C0 8E E0 8E D0
00000210  89 D4 0E 1F 0F 01 1E A2 00 0F 01 16 A8 00 BA 92 00 EC 3C FF 74 12
00000226  67 8A 64 24 04 84 E4 74 04 0C 02 EB 02 24 FD 24 FE EE E8 2B 00 80
0000023C  D1 E6 64 E8 24 00 B0 DF E6 60 E8 1D 00 B8 01 00 0F 01 F0 EB 00 88
00000252  18 00 8E D8 8E C0 8E D0 8E 00 8E 8E 66 EA 00 00 01 00 10 00 E8 16
00000268  00 E4 64 3C FF 74 0F AB 01 74 07 E8 09 00 E4 60 EB EC A8 02 75 E8
0000027E  C3 EB 00 C3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 7F
```

- Cute, easy to use, useful.
- Interactive, **immediate** editing.
- Bytes, strings, sometimes bits.
- Byte dump and ASCII views.
- Fixed capabilities: search, byte diff/compare, entropy, etc.



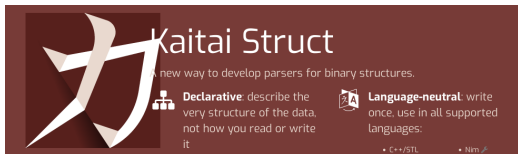
Specialized Binary Editors



- Pre-defined and fixed data structures: ELF editor, MP3 editor, foo editor, ...
- Not extensible.
- Choke on incorrect data.
- They implement everything but not what **you** need (doh!)



Encoders/Decoders Generators

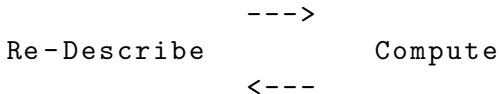


Decode ---> Compute [---> Encode]

- User-defined complex structures.
- XML, JSON, ... “declarative”
- Generate code in several target languages: C, Java, ...
- Non interactive.
- Can't deal with incorrect data.
- Either bit-oriented (uncommon) or byte-oriented.
- Often no encoders.



GNU poke



- Interactive, **immediate** editing with data integrity.
- User-defined complex structures.
- Powerful and to the point DSL: **Poke**.
 - Interactive, **statically** typed, garbage collected
 - **pickle**: file containing definitions of types, variables, functions, etc, that conceptually apply to some definite domain.
 - No bit-oriented, no byte-oriented: arbitrary **units**
- bit-addressable IO spaces.
- Can edit incorrect/guessed data (non-strict mapping)
- Extensible: write your own editor, commands and tools.



...but poke is not just about editing

- **Precise** and functional documentation of formats/protocols.
- Progressive investigation of a format/protocol, reverse engineering.
- Prototyping.
- Binary utilities (`#!/usr/bin/env poke -L`).
- Filters.
- Integration in other programs (e.g. GDB).



IO Spaces

- Abstract **bit-addressable** space of IO objects:
 - ints - signed integers from 1 to 64 bits wide
 - uints - unsigned integers from 1 to 64 bits wide
 - strings - sequences of NULL-terminated bytes
- Backed by some **byte-addressable** IO device:
 - Files (ios-dev-file)
 - Memory buffers (ios-dev-mem)
 - NBD servers (ios-dev-nbd)
 - Memory of a running process (ios-dev-proc)
 - Standard input/output/error (ios-dev-stream)
 - Zero (ios-dev-zero)
 - Sub IO space (ios-dev-sub)
 - Client application provided IO space (like GDB inferior memory.)



Bytes

- Data stored in modern computers is fundamentally a sequence of entities called “bytes”:

```
+-----+-----+-----+ ... +-----+  
| byte 0 | byte 1 | byte 2 |     | byte N |  
+-----+-----+-----+ ... +-----+
```

- Each byte stores a little unsigned integer in the range 0..255
- In Poke parlance, that is an `uint<8>`, aka byte
- `byte @ 4#B`
- `uint<8> @ 4#B = 66`



From Bytes to Integers

- `uint<16> @ 1#B`

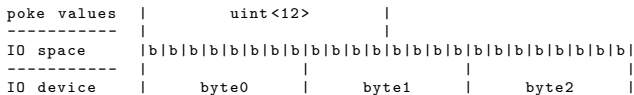
```
poke values                |                uint<16> @ 1#B                |
-----                    |                    |                    |
IO space                   |b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|
-----                    |                    |                    |
IO device                   |   byte0           |   byte1           |   byte2           |
```

- From 1 to 64 bits, signed and unsigned.
- Bit ordering is always big-endian.
- Byte ordering may be either big-endian or little-endian.

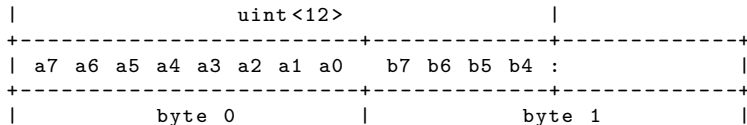


Weird Integers - Incomplete Bytes

- `uint<12> @ 0#B`



- But what is “the first half of the second byte”?



- In big-endian: a7 a6 a5 a4 a3 a2 a1 a0 b7 b6 b5 b4
- In little-endian: b7 b6 b5 b4 a7 a6 a5 a4 a3 a2 a1 a0



Weird Integers - Quantum Bytenics

- `uint<5> @ 0#B`

```
poke values | uint<5> |
-----|-----|
IO space   |b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|
-----|-----|
IO device  |      byte0      |      byte1      |      byte2      |
```

- Let's view the byte as a sequence of bits:

```
|      uint<5>      |
+-----+-----+
| b7 b6 b5 b4 b3 |   |
+-----+-----+
|               byte0               |
```

- The value is `b7 b6 b5 b4 b3` regardless of byte endianness.



Unaligned Integers

- `uint<16> @ 2#b`

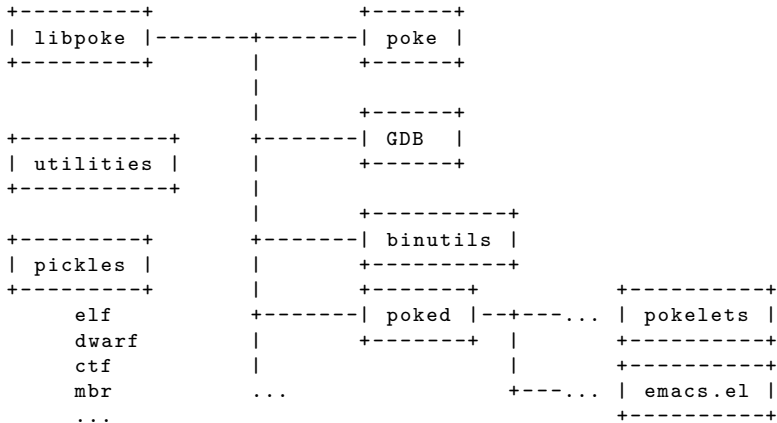
```
poke values      |          uint<16> @ 2#b          |
-----
IO space         |0|1|1|1|1|1|1|1|1|0|1|0|0|0|1|1|0|1|1|0|1|1|0|0|1|1|1|0|0|
-----
IO device        |    0x7f    |    0x45    |    0x4c    |
```

- This helps to visualize it:

```
poke values      |          uint<16> @ 2#b          |
-----
Virtual bytes    | virt. byte1 | virt. byte2 |
-----
IO space         |0|1|1|1|1|1|1|1|1|0|1|0|0|0|1|1|0|1|1|0|1|1|0|0|1|1|1|0|0|
-----
IO device        |    0x7f    |    0x45    |    0x4c    |
```



The pokesphere



Segmentation fault time
(demo)



“If the **mountain** will not come to **poke**
poke will go to the **mountain**”



GDB meets poke

```
+-----+                               +-----+
|      | <-- terminal if ----- |      |
| GDB  |                               | libpoke |
|      | <-- foreign IOS if --> |      |
+-----+                               +-----+
```

- GDB is good at debugging processes
- poke is good at poking at data
- ... guess what, GDB+poke is good at both!



The GNU assembler meets poke (WIP)

- Current data directives

```
.section .text
.set softfloat
.ascii "PS-X EXE"
.byte 0, 0, 0, 0, 0, 0, 0, 0
.word main
.word 0
.word 0x80010000
.word image_size
.word 0,0,0,0
.word 0x8001FFF0
.word 0,0,0,0,0,0
.ascii "Sony Computer Entertainment Inc. for zid"
```

- New .poke directive

```
.poke load psxexe
.poke var s = "Sony Computer"
.poke PS_X_EXE { start = $main, size = $image_size, vendor = s }
```



The Project

- Licensed under GPLv3+, both applications and libraries
- Homepage: <https://www.jemarch.net/poke>
- Savannah: <https://savannah.gnu.org/p/poke>
- Mailing list: poke-devel@gnu.org
- IRC channel: **#poke** in **irc.libera.chat**
- Pokology: <https://pokology.org>



Join the fun!

See file **HACKING** in the source tree.

- Current released poke version is 2.3
- poke 2.x is in maintenance mode
- poke 3.x is under development in the master branch.
- The poke daemon, TUIs, GUIs.
- poking at the kernel.

