

# PRISM, a light BEAM disassembler

Pass The Salt 2024, Lille, France

---

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 !



# Table of Contents

## Introduction

- Introducing the BEAM virtual machine
- State of the Art: a tour of available disassemblers
- Why another disassembler ?

## Writing a disassembler in 3 days

- BEAM and EZ file formats
- Instruction and operands encoding
- Accessing literals, atoms and functions information
- Building our disassembler
- Source code release

## Conclusion



# Table of Contents

## Introduction

- Introducing the BEAM virtual machine

- State of the Art: a tour of available disassemblers

- Why another disassembler ?

## Writing a disassembler in 3 days

## Conclusion

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**)
  - ▶ Hybrid 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**
  - ▶ **X<sub>n</sub>** registers are used for passing args
  - ▶ **Y<sub>n</sub>** 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)



# Table of Contents

## Introduction

Introducing the BEAM virtual machine

State of the Art: a tour of available disassemblers

Why another disassembler ?

Writing a disassembler in 3 days

Conclusion





# Off-the-shelf BEAM disassemblers

- ▶ beam\_disassemble<sup>1</sup>:
  - ▶ An Erlang module to disassemble BEAM bytecode (Erlang-ception !)
  - ▶ Based on Erlang/OTP compiler module
  
- ▶ BEAMdasm (original)<sup>2</sup>:
  - ▶ *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)<sup>3</sup>:
  - ▶ Works a bit better than the previous BEAMdasm
  - ▶ Still unable to disassemble some assemblies !

---

<sup>1</sup>[https://github.com/sg2342/beam\\_disassemble](https://github.com/sg2342/beam_disassemble)

<sup>2</sup><https://github.com/scout119/beamdasm>

<sup>3</sup><https://github.com/doorgan/beamdasm>



# Table of Contents

## Introduction

Introducing the BEAM virtual machine

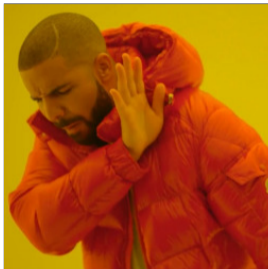
State of the Art: a tour of available disassemblers

Why another disassembler ?

Writing a disassembler in 3 days

Conclusion

# Why another disassembler ?



Fixing  
these VSCode  
extensions



Writing  
a new BEAM  
disassembler



# Table of Contents

## Introduction

- Introducing the BEAM virtual machine
- State of the Art: a tour of available disassemblers
- Why another disassembler ?

## Writing a disassembler in 3 days

- BEAM and EZ file formats
- Instruction and operands encoding
- Accessing literals, atoms and functions information
- Building our disassembler
- Source code release

## Conclusion

# Remember, I love challenges !



- ▶ 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





# 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)
- ▶ **Keep it simple stupid !**
- ▶ **Don't waste time**, really.



# Table of Contents

Introduction

Writing a disassembler in 3 days

- BEAM and EZ file formats

- Instruction and operands encoding

- Accessing literals, atoms and functions information

- Building our disassembler

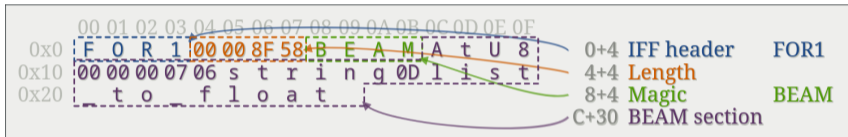
- Source code release

Conclusion





# BEAM file format



- ▶ 12-byte header with **IFF** and **BEAM** magic values
- ▶ **Section-based** file format
- ▶ All values are **big-endian**

# BEAM sections

	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	
0x0	A t U 8 00 00 00 15 06 s t r i n g 0D	0+4 Marker
0x10	l i s t _ t o _ f l o a t	4+4 Section length 0x15
		8+1 Atom[0] length 0x06
		9+6 Atom[0]
		F+1 Atom[1] length 0x0D
		10+13 Atom[1]

- ▶ 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

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



# Table of Contents

Introduction

Writing a disassembler in 3 days

- BEAM and EZ file formats

- Instruction and operands encoding**

- Accessing literals, atoms and functions information

- Building our disassembler

- Source code release

Conclusion



# 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<sup>4</sup>)

---

<sup>4</sup><https://github.com/erlang/otp/blob/OTP-26.2.5/lib/compiler/src/genop.tab>



# Code section

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
0x0	C	o	d	e	00	00	80	18	00	00	00	10	00	00	00	00	0+4 Marker
0x10	00	00	00	B6	00	00	04	83	00	00	00	99	01	10	? ?	4+4 Section size 0x8018	
0x20	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	8+4 Code version 0x10	
0x30	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	C+4 Instruction set 0x00	
																10+4 Max opcode 0xB6	
																14+4 Label count 0x483	
																18+4 Function count 0x99	
																1C+2 Instruction	

# 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
```





# Instruction decoding

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F			
0x0	C	o	d	e	00	00	80	18	00	00	00	10	00	00	00	00	0+4	Marker	
0x10	00	00	00	B6	00	00	04	83	00	00	00	99	01	10	?	?	4+4	Section size	0x8018
0x20	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	8+4	Code version	0x10
0x30	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	C+4	Instruction set	0x00
																	10+4	Max opcode	0xB6
																	14+4	Label count	0x483
																	18+4	Function count	0x99
																	1C+2	Instruction	

- ▶ Instruction opcode stored on 1 byte
- ▶ First instruction opcode is 0x01
- ▶ Instruction **0x01** is **label** of *arity* 1 (1 operand)
- ▶ Operands are encoded as **compact terms**

# 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

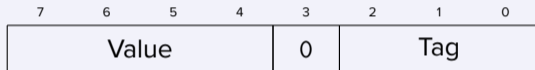
```

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



# Encoding a value (basic type)

## Value below 16

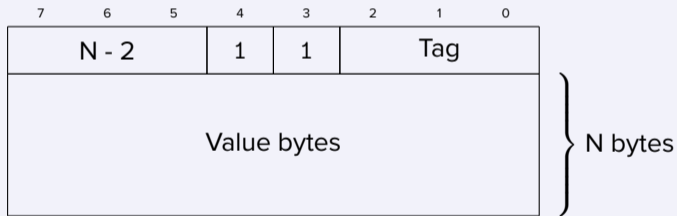


## Value below 2048



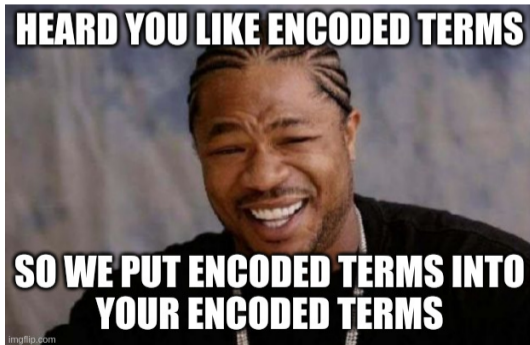
# Encoding larger values (basic type)

Values stored on 2-8 bytes

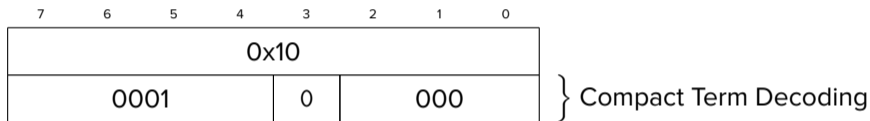


# Extended Term Encoding

- ▶ 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 can be interpreted as `label #1`



# Table of Contents

Introduction

Writing a disassembler in 3 days

BEAM and EZ file formats

Instruction and operands encoding

**Accessing literals, atoms and functions information**

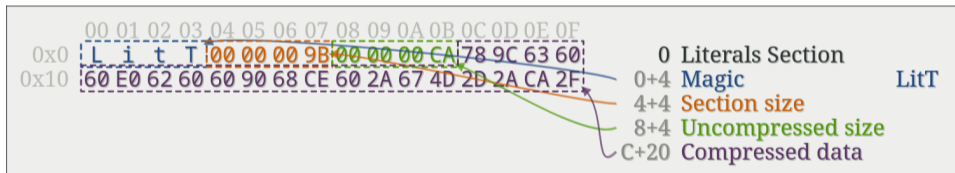
Building our disassembler

Source code release

Conclusion

# 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**

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
0x0	00	00	00	02	00	00	00	00	?	?	?	?	?	?	?	?	0 Uncompressed Literals
0x10	00	00	00	00	?	?	?	?	?	?	?	?					0+4 Number of entries
																	4+4 Reserved
																	8+8 Encoded term #1
																	10+4 Reserved 2
																	14+8 Encoded term #2



# Atoms section

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F		
0x0	A	t	o	m	00	00	00	20	00	00	00	03	07	e	x	a	0	Atoms Section
0x10	m	p	l	e	05	H	e	l	l	o	05	W	o	r	l	d	0+4	Marker
																	4+4	Section size
																	8+4	Number of atoms
																	C+1	Atom length
																	D+7	Atom
																	14+1	Atom length
																	15+5	Atom
																	1A+1	Atom length
																	1B+5	Atom

# Functions information section

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0x0	F	u	n	T	00	00	00	3B	00	00	00	02	00	00	00	01
0x10	00	00	00	01	00	00	00	28	00	00	00	00	00	00	00	02
0x20	00	00	00	00	00	00	00	03	00	00	00	02	00	00	00	55
0x30	00	00	00	00	00	00	00	05	00	00	00	00	00	00	00	00

0	Functions Section
0+4	Marker
4+4	Section size
8+4	Number of functions
C+4	Fun#1 atom
10+4	Fun#1 arity
14+4	Fun#1 offset
18+4	Fun#1 index
1C+4	Fun#1 nfree
20+4	Fun#1 ouniq
24+4	Fun#2 atom
28+4	Fun#2 arity
2C+4	Fun#2 offset
30+4	Fun#2 index
34+4	Fun#2 nfree
38+4	Fun#2 ouniq



# Table of Contents

Introduction

Writing a disassembler in 3 days

- BEAM and EZ file formats

- Instruction and operands encoding

- Accessing literals, atoms and functions information

- Building our disassembler**

- Source code release

Conclusion



# 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

# 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
```



```
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
```



# Table of Contents

Introduction

Writing a disassembler in 3 days

- BEAM and EZ file formats

- Instruction and operands encoding

- Accessing literals, atoms and functions information

- Building our disassembler

- Source code release

Conclusion

# Source code release



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

**Clone it, fork it, and send pull requests !**



# Table of Contents

## Introduction

- Introducing the BEAM virtual machine
- State of the Art: a tour of available disassemblers
- Why another disassembler ?

## Writing a disassembler in 3 days

- BEAM and EZ file formats
- Instruction and operands encoding
- Accessing literals, atoms and functions information
- Building our disassembler
- Source code release

## 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 🍷 **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



# Thanks



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

# Thank you

Contact information:

Email: [contact@quarkslab.com](mailto:contact@quarkslab.com)

Phone: +33 1 58 30 81 51

Website: <https://www.quarkslab.com>



# 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](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](https://github.com/erlang/otp/blob/cf8c4267657bf3bce5847bb44793f6ff092fea39/lib/compiler/src/beam_disasm.erl)