

RE-Learning w/R2

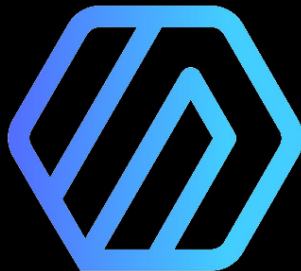
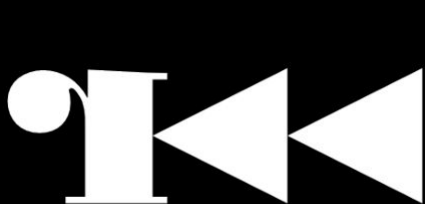
@pancake@infosec.exchange // NN2025



Who Am I?

Sergi Àlvarez aka **pancake**

- Mobile Security Research Engineer at NowSecure
- Author and leader of the Radare project
- Free Software enthusiast and developer



Target Audience

- Newcomers to the Low Level Reigns
- Commandline cowboys
- Malware analysts
- Unix Enthusiasts

Poll

First of all, let's understand better the audience:

- Do you know and use **radare2**?
- Can you read **assembly**?
- What about **Reverse** Engineering?
- Toolkit overview

Contents

- Setup r2 and get comfortable in the shell
- Analyzing binaries, from headers to the code
- Scripting tasks in Python and Javascript
- Popular extensions and plugins
- Dynamic Instrumentation Debugging / Tracing
- Learn more (chats + books)

Disclaimer

Take these slides as a reference!

- **Focus on practical examples**
- Get **comfortable** in the shell

What's radare2

The
Libre Software
Reversing Framework

History

Back in 2006..

- I was a forensic analyst
 - And had to recover some deleted files from a mac
 - I was not allowed to use company software
- So I wrote my own thing
 - A portable unix-centric hexeditor for 64bit seeks
- 18 years after that it's still kicking

In short

Libre RE Framework with **UNIX** philosophy in mind.

- Purely written in **C**, portability and control matters
 - Very extensible through plugins and scripts
- Added disassembler, binary parser, analyser
 - Debugger, Emulator, Scripting, GUI
- One-man project most of its lifetime

Installation

- Always from git or latest release.

```
$ git clone https://github.com/radareorg/radare2
```

```
$ radare2/sys/install.sh
```

<https://www.radare.org>

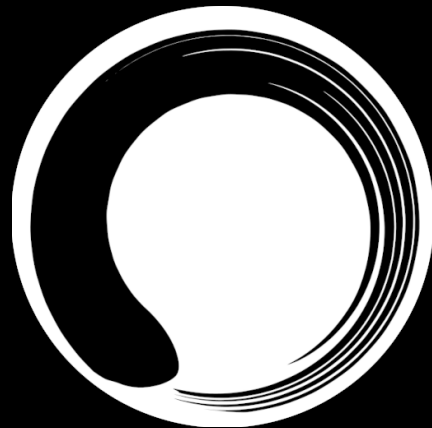
Iaito

The official GUI (but there are more)

- Runs on BSD, Haiku, Linux, macOS, Windows

Check the release page or use flatpak

<https://github.com/radareorg/iaito>



Exercise

Get your workshop environment ready!

- **Install radare2 from git**

Introduction To The Shell

The Magic Of The
Commandline

Commandline

Main way to interact with **radare2** is through the **shell**

- Basic posix shell commands (ls, cd, rm, cat, ..)
- Learning the commands and syntax matters!
- Subcommands just add a letter after the root one
- Useful for **scripting** and automation
- Doing things faster than using the mouse

Basic Commands

s = seek (s 0x/s..)

p = print (px / pd)

f = flags

i = info

w = write

q = quit

a = analysis

v = visual/panels

e = eval config

? = help/math

! = system shell

d = debugger

Command Operators

| = redirect to process (like in posix shell)

> = redirect to file (\$file are internal)

~ = internal grep (indent json, xml, code, filter words)

= comment

; = command separator

? = show command help

Command Suffixes

- **?** = help message
- **j** = json
- ***** = r2 commands (scripts)
- **q** = quiet (less verbose)
- **,** = comma separated values (table)
- **k** = key-value

Command Prefixes

- (**number**) = repeat a command N times
- ' = single quote, to avoid parsing special characters
- **?t** = calculate execution time
- : = io command
- `` = replace command output inline
- . = run script

Iterator Operator

Useful to run commands in different items

- Functions, flags, registers, symbols, basic blocks, ..
- @ - temporal seek
- @@ - repeat command on different places
- @@@ - advanced repeat actions

See @? @@? @@@? for help

Useful Commands

Combine and learn new commands every day!

- Recursive Help: **?***
- JSON indent (json path queries like jq): **~{ }**
- HUD filtering: **~...**
- Analyse all symbols: **af @@ sym***
- Set, list flags: **f**
- Comments: **CC**

Commandline Exercise

- Open a file (`/bin/lis ;D`)
- Dump bytes and disassemble some instructions
- Seek to different addresses
- Analyze code / list and count functions
- Use tab to autocomplete flags
- Enter visual mode (v, V)
 - Bonus: Try the new Visual Matrix (V#)

IO

—

The lowest layer in r2,
where everything looks
like a file.

IO Plugins

List of uri handlers exposed by the IO plugins:

```
$ r2 -L
```

You can find more plugins if you need them

```
$ r2pm -s ...
```

IO Primitives

The plugins expose:

- **Open/Close** = handle uri **://** to select plugin
- **Seek** = used to move around, 64 bit offsets, getsize
- **Read/Write** = basic IO operations
- **System** = run a command return string with result

Maps and File Descriptors

Use **o** and **om** commands to list files and their maps

- Necessary to configure the memory layout

```
[0x100003a84]> o
```

```
3 - r-x 0x00025af0 /bin/ls
```

```
4 * r-- 0x00002510 null://9488
```

```
[0x100003a84]> om
```

```
* 5 fd: 3 +0x00010000 0x100000000 - 0x100007fff r-x fmap.__TEXT
```

```
- 4 fd: 3 +0x00018000 0x100008000 - 0x10000bfff r-- fmap.__DATA_CONST
```

```
- 3 fd: 3 +0x0001c000 0x10000c000 - 0x10000ffff r-- fmap.__DATA
```

```
- 2 fd: 3 +0x00020000 0x100010000 - 0x100015aef r-- fmap.__LINKEDIT
```

```
- 1 fd: 4 +0x00000000 0x100015af0 - 0x100017fff r-- mmap.__LINKEDIT
```

```
[0x100003a84]> █
```

Run IO Commands

IO Plugins expose an optional callback to run commands through the `:` prefix from the core shell.

- **Expose custom functionality**
 - FileSystem (when using the [fs.io](#) plugin)
 - Debugger (when using the [debug.io](#))
 - Binary parsing (show symbols, ..)
 - Custom IO configurations (VM Layout/..)

Searching Patterns

The **/** command is used to search stuff

- **/** - text
- **/x** - byte patterns (with binary mask?)
- **/a** - assembly code
- **/c** - cryptographic materials
- **/m** - magic headers
- **/z** - find strings

Exercise

Create your custom configuration file in your home!

- **r2 -H R2_RCFILE**
- Select a theme with **eco**
- Change **scr.** and **asm.** options

Structured Binary Data

Files with Executable
Code Structured with
Headers and Metadata

Binary Formats

The list of file formats supported is very large:

```
$ rabin2 -L
```

- ELF, MACHO, PE, COFF, NE
- DYLDCACHE, KERNELCACHE
- CLASS, DEX, LUA, PYC
- GB, NES, 3DS, SMS, SMD, XBE, Z64, NSO
- ..

Parsing Headers

Executables and libraries store information needed by the operating system to load and execute them.

- Sections and segments
- Symbols, imports and exports
- Entrypoints, constructors / destructors
- Strings, Libraries / Dependencies

```
$ rabin2 -H (ih)
```

Libraries

In GNU/Linux we use the **ldd** program to list the libraries a program is using.

In r2 we can use **il**, which is portable and doesn't have code execution risks.

- Same functionality is available in **rabin2 -l**

Sections vs Segments

Rabin2 unifies concepts for simplicity

iS vs **iSS**

- Sections are only needed for static analysis tools
- Segments is what the runtime linker needs.
- Check with ***** and **om**.

Strings

Plain text stored in the read-only sections of the binary

- Sometimes compilers put code in rw sections
- Eventually they are inside unmapped headers
- Sometimes the text is generated with code
- Or maybe it is encoded (base64, rot13, ..)

```
$ rabin2 -z /bin/ls (see -zz and -zzz)
```

Exporting binary details as script

Using the **-r** flag to create an r2 script

- This **-r** works across all tools in r2land
- r2 commands use the ***** suffix

Commands inside r2

R2 is the tool that unifies all the other tools.

- Use RCore which links against RBin, RArch,...

The rabin2 functionalities are implemented under the **i** command.

```
$ rabin2 -z == iz, -zz = izz, ...
```

- Check the shell

Decoding Instructions

Analyzing program code,
control flow graphs,
string references, ..

Decoding

Different representations of the same

- Zeros and Ones (Machine Code)
- Bytes in Hexadecimal (octal was more readable)
- Plaintext Assembly
- Pseudo Disassembly
- Intermediate Representation (ESIL for r2)

Supported Architectures

```
$ rasm2 -L
```

Note that arch plugins can optionally provide

- **ESIL** representation for emulation
- Encoding (assembler) support
- Different **CPU** models

Visual Instruction Decoding

- Vd1

```
r2's bit editor: (=pfb 3b4b formatting)
```

```
adr: 0x00006db4
```

```
hex: 488b0525c20100
```

```
len: 7
```

```
shf: >> 0 << 55
```

```
asm: mov rax, qword [rip + 0x1c225]
```

```
esl: 0x1c225,rip,+, [8],rax,=
```

```
dec:      72      139      5      37 |      194      1      0      72
```

```
hex: 0x48 0x8b 0x05 0x25 | 0xc2 0x01 0x00 0x48
```

```
bit:  ·1·1·· 1··1·11  ····1·1  ··1··1·1 | 11···1·  ······1  ········ ·1··1··
```

```
bit: 0-00-000  -000-0.. 00000-0. 00-00-0. | ..0000-0 00000000. 000000000 0-00-000
```

00 mov = 02042

```
1r rax = 0130
```

[illegible]

```
\\\\\\\\\\\\\\\\\\\\_\\\\\\\\\\\\\\\\\\\\_\\\\\\\\\\\\\\\\\\\\_\\\\\\\\\\\\\\\\\\\\_ 5i 0x25] = 05
```

$$4r + = 00$$

Evaluable Strings Intermediate Language

- Designed by and for radare2
- Like FORTH, but using commas instead of spaces
- Expresses the instruction behaviour
- Simple to parse, fast to execute

mov eax, 33 => 33,eax, :=

Disassembler Options

- > **e arch.** # architecture options
- > **e asm.** # display disasm options

Enable emulation for computed references

- asm.describe
- asm.emu/emu.str

Graphs

Control Flow Graph, for function basic blocks

- **agf**

Formats: ascii art, graphviz, mermaid, ..

- **agfd, agfm, ...**

Exercise

- Open **/bin/sleep** with **-n** and **-w**
- Use **o** and **om** to see the differences
- Patch the entrypoint with a ret
 - Advanced: modify the default behaviour
- radiff2 to understand the patch we did
- Execute the patched program

Uplifting To High Level Languages

Retrieving a high level
representation of the
underlying assembly
code

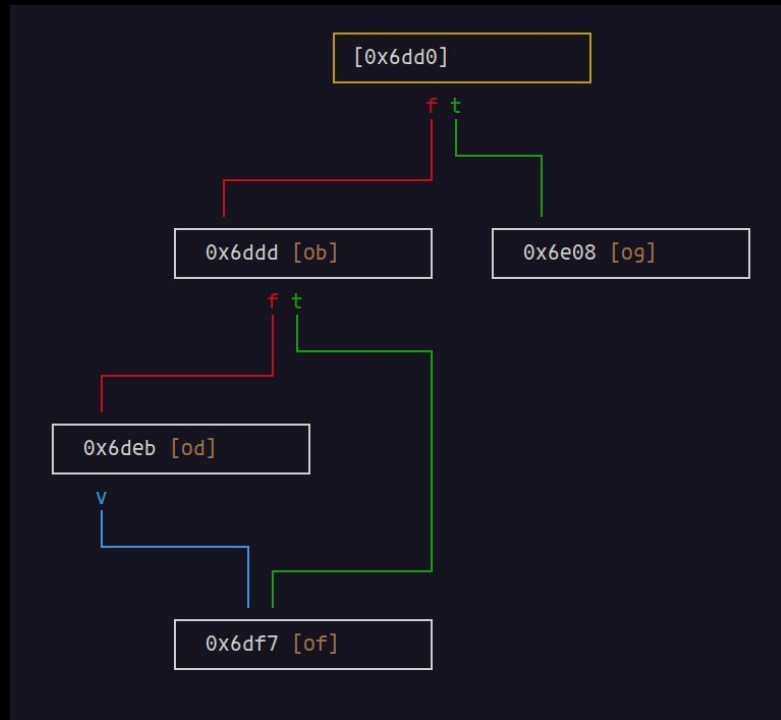
Analysis

r2 -A = aa / r2 -AA = aaa

- Functions / BB / Ops
 - **afl, afb, ao**
- Different analysis: **aa?**
- Options: **e anal.**

Use them wisely

- Default is not always the best



Decompilation

The art of creating high level representations of the assembly code, aka, the inverse step of compilation.

- Assumptions to fill the gaps with the info we miss

Pseudo Decompile with pdc

Native to r2, works on all archs

- Enables **asm.pseudo**
- Prints each basic block with labels and gotos
- Uses ESIL to reference ALL strings
- Very verbose, but useful when others fail
- Very fast, perfect for grepping around

Decompiler for r2 written in Javascript

- Quite correct, few optimization passes
- Supports most common archs
- Actively maintained and developed
- Available in the **pdd** command
- By **@wargio/deroad**

6502
8051
ARM Thumb
ARM 32/64
AVR
Dalvik
m68k
MIPS
PowerPC 32/64
SuperH
SPARC
v850
WebAssembly
x86/x64
RISC-V

r2ghidra

Native plugin linking to the ghidra-native fork of ghidra's decompiler (only c++ code, no java)

- Not aligned with r2 analysis
- Good results sometimes, but misses lot of info

Decompiler based on R2AI:

- Takes N decompilations as input
- Generates better output combining them
- Guess variable names and arguments
 - Best Local: **gpt-oss:20b**
 - Best Remote: **Claude**

Exercise

- Install r2dec, r2ghidra, decai
- Try them on different functions of different binaries
- Understand the differences

Debugging And Tracing

Manipulating program
execution at runtime

Low Level Debugging

R2 is a tool for reversing, not for developers

- No plan to replace gdb/lldb
- It's not a source debugger.

But it's great when you don't have the source

- Easy to script and automate

```
$ r2 -d [program|pid]
```

Backends

The native backend works on all major platforms!

- Linux, macOS, iOS, Android, Windows, *BSD, !!

But sometimes we need to do remote debugging

- Over TCP / JTAG, use the gdb:// protocol

Windbg / gdbio / qemu / bochs support..

Registers

Showing and changing register values

> **dr, dr=, dr 32, dr rax**

- Telescoping with **drr**

We can also telescope memory with **pxr@r : SP**

- Register profiles with **drp**

Breakpoints

Use the **db** command for that..

No need to use a temporal breakpoint. You can continue until address with dcu

With some archs sometimes you may need to use:

- **e dbg.hwbp**

Memory Maps

At runtime, the address space is not fully mapped

- Use **dm** and **dmm** to understand the layout

Identify regions by permissions and name

- Where's the stack, inspect it with **pxr**

Heap Structures

Heap memory is structured in a way that can be parsed and detect corruptions, which is useful for analyzing and exploiting buffer overflow vulnerabilities.

- Check the **dmh** command

Exercise

Start debugging a program, change control flow by changing the program counter.

- Manipulate register values: **dr**, **dr=**
- Identify location with maps: **dm**
- Continue execution: **db**, **db-***, **ds**, **dc**

Scripting

Automate actions,
Solve boring tasks
Quickly

The Basics

We know how to use the shell.

- `r2 -i` or the `.` command.

What about running a command and capturing the output displayed in return?

- That's called `r2pipe`

We can also use bindings to the native API (`rlang`)

Supported Languages

For r2pipe you can literally use ANY language

- Python, JavaScript, Ruby, Nim, Scheme, ...

Even native!

- C, Vala, Rust, Swift, Zig, D, ...

Why Javascript

Is the only scripting language that is widely available, uses no setjmp and it's very easy to use and many languages have it as a target for transpilation.

- Nim, TypeScript, V, Scala, Dart, LUA, Scheme,...

We ship quickjs, scripts must be named **.r2.js**

R2Pipe

Example using the basic r2pipe api

```
import r2pipe

r2 = r2pipe.open("/bin/ls")
out = r2.cmd("?E Hello World")
print(out)
r2.quit()
```

R2Pipe Backends

R2Pipe can be used in different environments:

- Spawn + pipes
- Spawn + stdio
- Fork current session + pipes (#!pipe)
- Talking to an HTTP webserver /cmd
- Dlopen RCore API

R2Pipe JSON (cmdj)

Appending **j** to any command in r2 shows JSON.

Using the **cmdj** methods returns an object.

We can autogenerate object schemas and have autocompletion in our favourite editor!

```
cmdj(command: string) : string {  
    return JSON.parse(this.cmd(command));  
}
```

R2Pipe cmd vs call

Running a command implies too much internal work sometimes that we can bypass with `.call()`

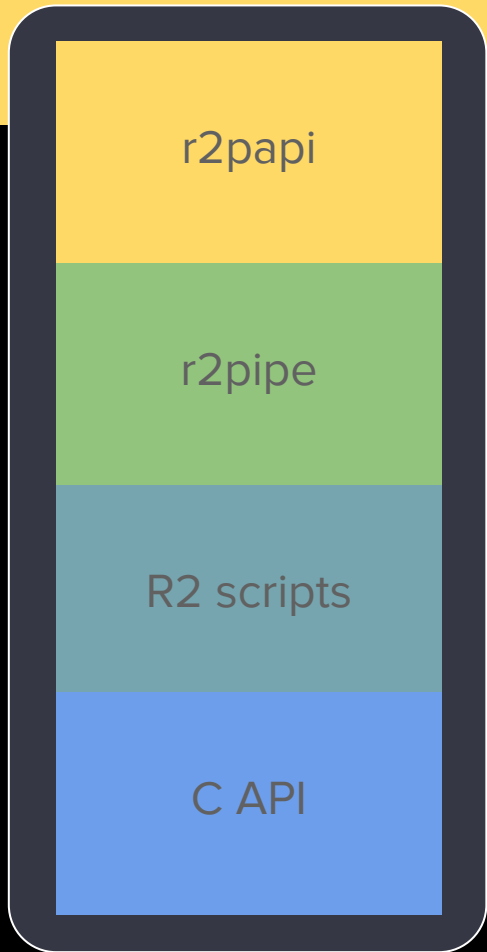
- Don't parse special characters
- Avoid command injection
- Support temporal seek `.callAt()`
- Faster execution for large scripts

Performance

Who said speed?

Sometimes we don't need the output

- Use **cmd0** or **call0** commands



R2Pipe2

Introduced in r2-5.9.x, still under development and not fully handled; needs more testing, feedback and contributions.

- Protocol is there
- Fully compatible with r2pipe
- Uses the `{` command from r2
- Captures stderr and return code and value

R2Papi

What about having an idiomatic and high level API on top of the r2pipe primitive?

- Similar to the Frida API (NativePointer, ..)

```
}  
/**  
 * Copy N bytes from current pointer to the destination  
 *  
 * @param {string|NativePointer|number} destination address  
 * @param {string|number} amount of bytes  
 */  
async copyTo(addr: string|NativePointer|number, size: string|number) : Promise<void> {  
    this.api.call(`wf ${this.addr} ${size} @ ${addr}`)  
}  
/**
```

r2skel

This project is a collection of template source codes in different languages for starting new plugins or scripts for radare2.

```
$ r2pm -ci r2skel
```

```
$ r2pm -r r2skel ..
```

Exercise

Install r2skel and write a core plugin in your language of choice to add a new command in the r2 shell.

Choose wisely!

- C, Python, R2JS

Plugins

Using the r2pm package manager, installing plugins to add new features.

Extensibility

We are about to reach the end of this training, but we won't be over without having a look at all the awesome tools that can be integrated with radare2!

- Use **r2pm** to search and install them!

r2frida

The best way to combine dynamic instrumentation with static analysis, a powerful shell on top of the tracing capabilities of Frida.

```
$ r2 frida://0
```

r2(ida|ghidra|bn)

Exchange analysis info from/to r2

- Get an r2 shell inside those tools
- Using r2pipe
- The r4ghidra is a full rewrite of the r2 shell in Java

- Looking for contributors!

Useful for crypto constant and malware analysis

- Create Rules with patterns
 - Integrated with r2 analysis and metadata
- Load them into memory
- Scan for patterns in memory or file

Symbolic Execution Solver (in Rust) on top of ESIL.

- Remake of esilsolver (z3py)
- Can resolve conditions that must be matched to reach a specific address
- Resolve passwords from crackmes, ..

r2poke

GNU/POKE is a programming language for describing binary files. Exposes a shell with powerful scripting capabilities.

- Integrates well with radare2
- Can run r2 commands from POKE
- Run POKE expressions in the R2 shell

Integrate Angr decompiler with radare2

- Looking for contributors and better integration
- A bit slow the first run needs to analyze the whole binary

Integrating language model capabilities within r2.

- Supports any local or remote LLM provider
- Easily scriptable with r2pipe
- Integrated with the analysis context

I'll be speaking on Saturday about it!

The standard file format to exchange findings from different source and binary analysis tools.

- Uses JSON format
- Well structured and extensible
- Inspect vulnerabilities found by other tools

r2diaphora

A fork of the Diaphora tool from Joxean for IDA, but maintained to work with radare2.

- Designed to work on scale
- Battle tested on fuzzed and malware binaries
- Looking for contributors!
- No SQLITE backend
- Needs a GUI

Exercise

Choose your favourite plugins and install them!

```
$ r2pm -ci r2ghidra r2dec r2yara r2svd
```

Continue Learning

Reference Books, Chats,
Blogs, Manpages,
Videos, Conferences..

Crackmes vs Projects

Do you have a project in mind?

- **Go for it!**

If you are used to other tools, make them play well with r2. It's easy and gives you lots of capabilities

Source

As I use to say, the best documentation is the source!

- Read as much code as you can, and when you are tired write more or refactor it!
- Coding plugins for r2, programming tools on top of it, or adding new commands are great ways to learn more about r2
- Fix bugs, add tests, open tickets!

r2book

Yep! We have an official book and it's opensource

<https://github.com/radareorg/radare2-book>

Feel free to contribute and make it better!

- It's also available as an r2 plugin

Chats!

Join our Discord, Telegram or Matrix chats!

- We have pancakes .. i mean cookies!

The Fediverse!

Follow [**@radareorg@infosec.exchange**](mailto:@radareorg@infosec.exchange)

(if you're still not in the fediverse, it's never too late!)

Exercise: Attend r2con!

Our periodic conference where r2 users, developers and hackers around the world meet in person!

- **This 2025 it's Only Online**
- **November 24th, 25th**

<https://www.radare.org/con>

Radare2 TV

- An index to find videos to learn to use radare2

<https://www.radare.org/tv>

Questions?