Radare2 Developers Training

r2con 2021

Introduction

In this session we will learn about the internals of radare2 and how can we use them for extending its capabilities or fixing a bug.

Who Am I?

Author of radare, radare2.

- Formerly known as pancake, but also Sergi Àlvarez i Capilla in real life
- 38yo Mobile Security Researcher working at NowSecure
- Wrote other free software projects like 0xFFFF, ACR, Valabind, ..
- Optimized codecs in assembly for mips, arm, sparc and x86

Who Are You?

- Hopefully someone willing to learn
- Or maybe contribute

Contents

- Understand the directory and source structure of the project
- Basic syntax rules and data structures
- Using git and gh
- Learn how commands are created
- Extend with new plugins
- Scripting with r2pipe
- Questions? Chat debate

Requirements

- Any modern system would work, preferably:
  - macOS, *BSD, Haiku, GNU/Linux or Windows
- Your favourite editor and C skills:
  - I like vim, but vscode or visual studio also work quite well
- Compiler and tooling:
  - A C compiler (tcc, clang or gcc)
  - Git, make, pkg-config
  - Tig, Zfz, Valgrind and Code Sanitizers
- Other packages:
  - r2frida

Wanna help?

There are many ways to contribute!

- Add new packages for r2pm
- Implement new features for radare2
- Fix a bug/crash/error
- Report a problem or request a feature
- Improve the documentation (help messages, fortunes, book, ...)
- Join the Discord/Telegram/Matrix/IRC channels
- Write articles, record videos or podcasts solving problems with r2
- Watch this presentation for more ideas!

Exploring the sources

Directory Structure

One common concern for beginners is the way the code is structured, it aims to be simple:

- Libr : directory containing all the libraries of r2 (one per directory)
- Binr: main executables that will be installed later
- Shlr: shared / external / 3rd party libraries
    - shlr/capstone

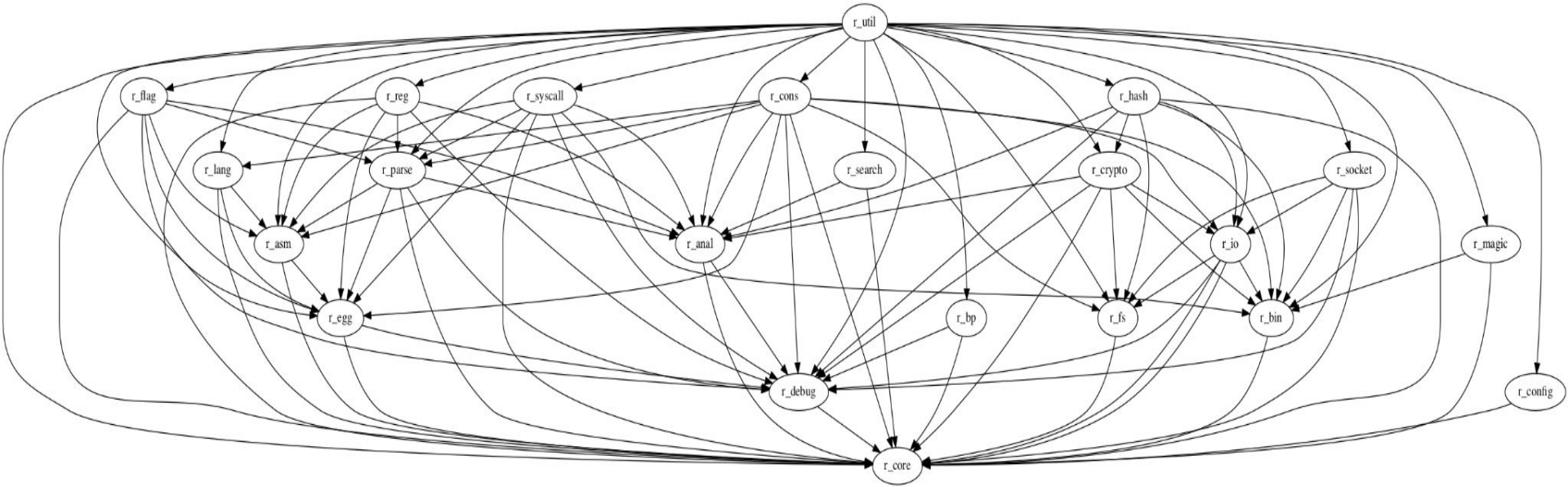
Each directory under libr can have none or some of the following:

- D: data associated with the module
- P: plugins to be loaded when initializing the library
- T: test files and scripts

Code Structure

- Public APIs are prefixed with R_API, you can grep for that
- Public function names start with r_modulename, for example: r_util or r_core
- Lifecycle involves _new() and _free() functions
  - When plugins are in need, the _use() function should be called
- RUtil is the standard library for radare2, that way we abstract from the OS
- RMain provides the main functions of all the r2 tools, busybox style!
  - So you can call rabin2 from r2
- Tight dependency between modules
- SDB - external project sync with r2 for source compat reasons
- Capstone and vector35

$ make depgraph.png



Data Structures

Many data structures are implemented in r2

- Hashtables, trees, lists, sets, etc

Pick the one that fits better for your needs, sometimes its not clear

- RList is implemented in libr/util/list.c
- Git grep is your friend. Check shlr/sdb and libr/include

- sdb
- Rlist
- Rht
- Rset
- Rbtree

Common Data Structures

C doesn't provide any standard implementation for any of those, we use the ones in sdb and r_util.

- RList - double linked list, can be used as a stack using the .push() and .pop() methods
  - RSList - single linked list
- RVector - single fixed size linear allocation containing inlined structs or pointers
- RHash - hash table, have variants for ut64 and string as key or value
- RBTree - tree structure

We need to know when and how to use each data structure, or implement a new one custom for our needs, in case of doubt, is sometimes useful to evaluate the use of `sdb` (in process key-value on disk and memory database).

SDB

The k command

Check the shlr/sdb directory (or the radareorg/sdb repo)

- Provides a key-value string database with disk serialization and disk-based hashtables
- Supports binary, plaintext and precompiled-C databases
- GPERF - the magic
- Provides linked list, hashtables, and other simple data structures
- Attic contains old/unused but still useful and curious data structures like julian arrays

Compilation

Fetching the Source

You can fetch the source of radare2 from different sources:

- Code from git
- Tarball snapshot from github
- Release source tarballs or builds compiled in the CI

Offline builds require ./preconfigure to be executed beforehand

Building

The recommended way to install r2 from git:

- Git clone https://github.com/radareorg/radare2 && cd radare2
- sys/install.sh or sys/user.sh

Support for both build systems

- ACR (./configure, make)
- Meson (ninja, visual studio)

Compilers supported

- Tcc, Clang, Gcc, MSVC, and probably more if they support C90

The CI

R2 is built in many different ways in the CI, this ensures portability across different systems.

For more details see .github/workflows/ci.yml

ACR

The ‘acr’ build refers to ‘autoconf-replacement’, which is the tool used to generate the `configure` script.

- Recently the ./preconfigure script has been introduced
  - Fetches the external repositories and prepares and checks the environment before running configure
- Run ./configure
  - See --report and --help for more details
  - Usually you may want to pass --prefix=
  - Use --with-rpath to enable
- Run make
  - This will build the whole project, but you can compile only parts when running in a subdirectory
  - Make install DESTDIR=/tmp/prefix for chroots/packaging/..
- Install
  - Default installation with sys/install.sh is done with symlinks, this allows to get instant system wide changes without having to run make install again when editing the code.

ACR on Windows?

The preconfigure script was needed initially for supporting the same compilation workflow on Windows systems. This is, the root directory contains a preconfigure.bat, configure.bat and make.bat scripts that will do exactly what you expect them to do, but on windows.

- Configure a clean python virtual environment
- Setup meson and ninja
- Find and load compiler environment for Visual Studio
- Runs git clone to download capstone and vector35 disassemblers

Now you can run configure, which under the hood calls meson

- It will also create a `vs` folder with the project to load in Visual Studio

And finally: make will execute ninja

Meson

With meson we have some benefits over acr/make:

- Supports visual studio
- Supports building on paths with spaces

But the build options and supported targets differ from acr/make. So both build systems are maintained

- meson —backend=vs build
- ninja -C build

The preconfigure.bat and configure/make batch scripts will do the dirty work for you

Coding!

Common Developer Workflows

That's up to you, pick the workflow thats better

- Use eprintf for debugging messages
- Use the r_return apis as preconditions reduce checks on release builds
- Use r_sys_backtrace and r_sys_breakpoint
- Install with symlinks, so just running make is enough to get the build working
- Run make only on the directories you are working on
- Use sys/rebuild.sh to recompile few specific parts of r2 like gdb+io or cs+asm

Coding Style

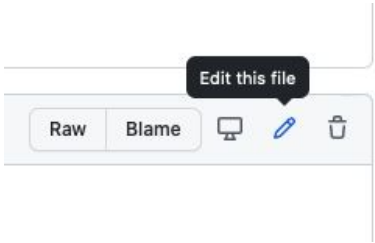
- Check out the sources as example
- No tool can indent C perfectly
- But following few basic rules is easy to do by hand
  - Just try to be educated
  - Use tabs
  - Add a space before the parenthesis when calling a function (like you do already for keywords)
  - Open braces at right
  - Cases at the same level as Switch

Pull Requests

The best way to contribute with code is via pull requests. This is done in GitHub, after forking the repository.

- Create a branch: git checkout -b changes
- Do your changes you want
- Git commit -a
- Git push -u origin master changes
- Open github in your browser

It’s also possible to edit the files directly in the web by clicking the pencil button.



From Commands to APIs

Where is a command implemented?

As easy as typing the following command:

- Git grep '// "command"' libr/core

By convention all commands have the '// "cmd"' comment, so for example let's find `"px"`:

- Git grep '// "px"' libr/core

Let's dig in there!

Core and IO Plugins

Those two types of plugins have a `system/command` interface to execute a command represented as a string and get back the result as a string (or int, and then you must pick the data from RCons)

You can load custom plugins from r2:

- Placing them in the plugins directory (See r2 -H)
- Loading them at startup time (r2 -l (lowercase L))
- Loading at runtime (L command)

List them with r2tool* -L

Write your own command

...

...

Other types of plugins

Check in the radare2-extras repo for more examples (all of them available via r2pm)

- RAnal plugins can now implement disassemblers (no need for duplications!)
- RAsm/RIO can be implemented in js or python
- RBin plugins to load more filetypes

Scripting with R2pipe

RCore.cmdStr()

That’s the only function needed to do anything with r2, by following this concept, the r2pipe libraries implement ways to interact with the shell to run a command and get the output as a string or JSON object.

Communication Channels

The script and the r2 instance can communicate using different ways:

- Process spawn + 2 pipes
- Environment R2PIPE_IN and R2PIPE_OUT variables
- Talk to the embedded webserver /cmd/ endpoint
- Comunicate via TCP/ rap protocol
- Dynamically load the r_core library and use r_core_cmd_str() natively

RLang

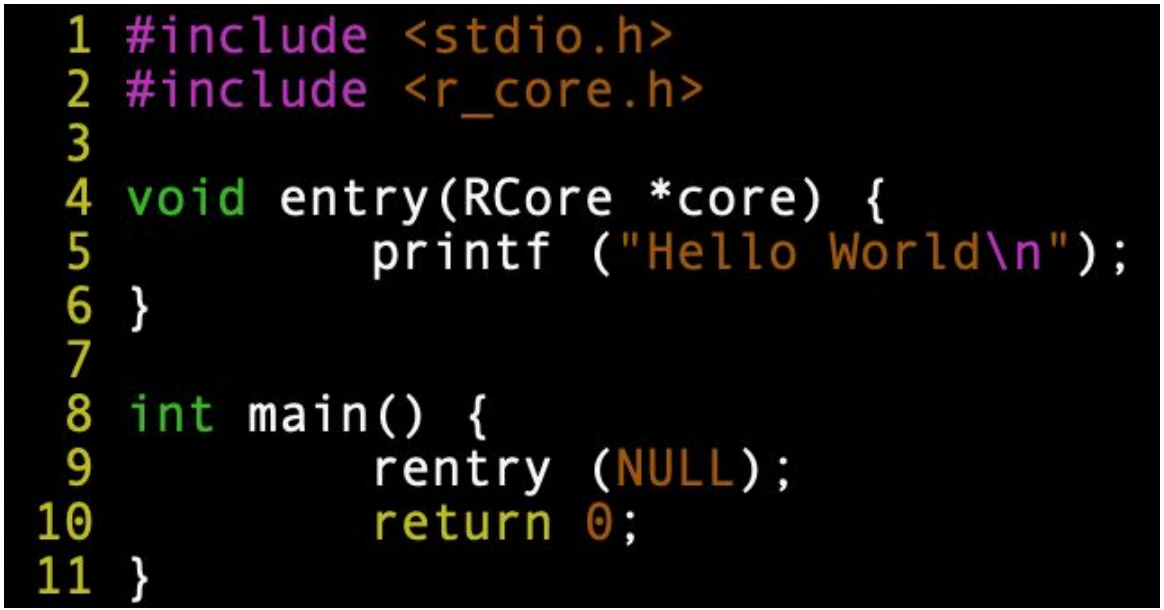
Scripting is also possible with statically compiled languages like C, this allows faster

- But also with Vala, V, Go, Rust, Zig, Nim and many other

This is done via the r2pipe interface #!pipe (automatically detected by extension when the ‘.’ command)

- Compiles the code
- Loads the library
- Runs the entrypoint
- Frees the thing

Scripting in C!



Bindings

The R2 api can be used directly by calling the c apis instead of the r2 commands.

There are autogenerated bindings for many programmings language

- valabind
- Bindgen
- Swig

Updating bindings

Stabilize/document apis using vapidoc

- Use the vapi files

Not all commands support the `j` suffix

- Help identifying them using the `?*` command and trying them in the shell.
- Extend the commands

JSON stuff

There's maybe some JSONs created manually, or calling `pj_new()` instead of `r_core_pj_new()` inside RCore

- Proper encoding, maybe evaluate base64 for binary data.
- The RCore.pjNew() call honors the user settings

Not all commands support the `j` suffix

- Help identifying them using the `?*` command and trying them in the shell.
- Extend the commands

RTable stuff

Associated to the comma command suffix

Permits to filter/reorder/join all the data

Exports in CSV, JSON, ...

Kind of Excel/SQL but in 1000LOC

- More commands need to be integrated with RTable
- Reduces code duplication for listings
- Allows user queries

WebServer

The Web

The portability and flexibility of the web platform is quite extense:

- Easy to tweak, modify and deploy

But also negative points:

- Feels non-native / laggy
- Requires a big runtime (web browser or electron)
- Single threaded r2 could make the web unresponsive

R2 ships its own webserver implementation, no need for 3rd party sw.

- When compiled with libuv it may be more responsive

Tasks and background server

Its possible to run a tcp or http server in background using rap or r2pipe protocols

- use the & command to manage tasks
- Run the webserver with =h& or & =h

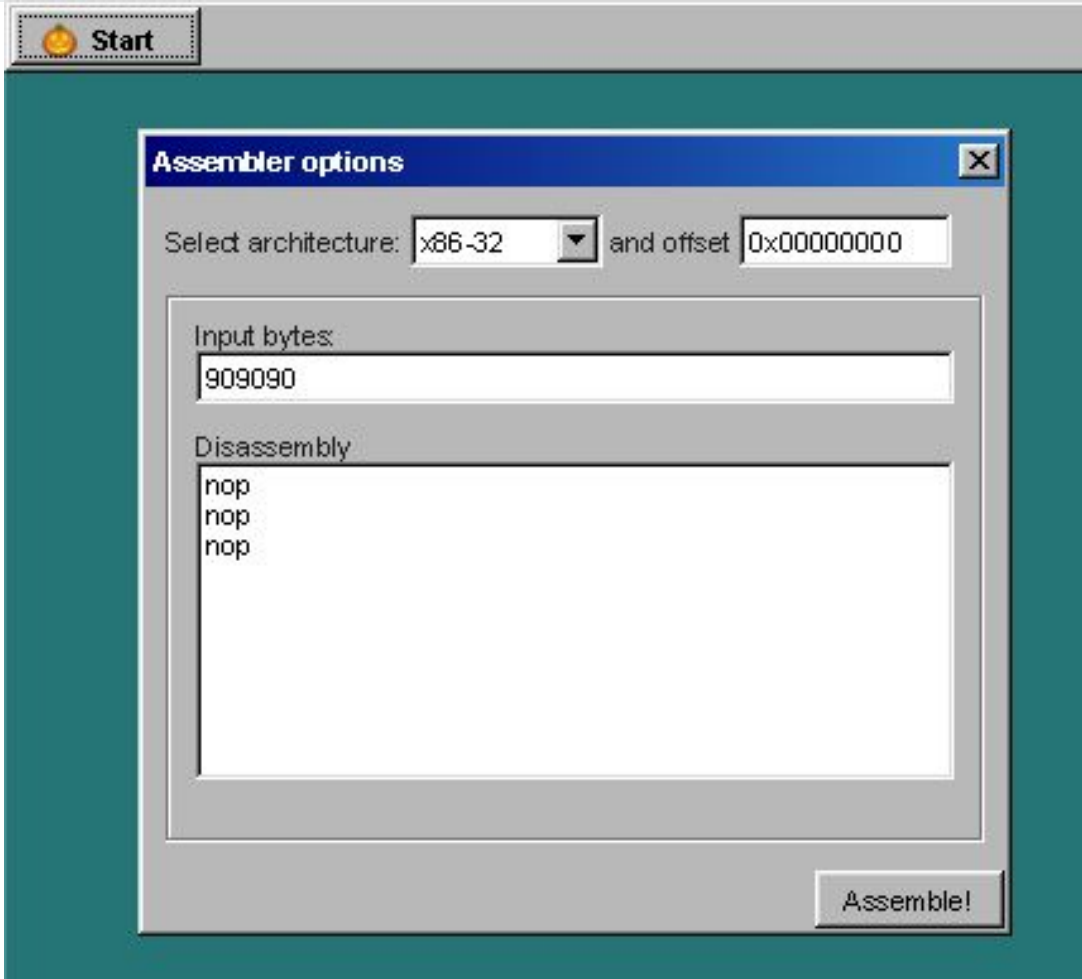
Allows to use the webserver while keeping the local shell accessible

- It’s also easy to implement a native webserver in nodejs, V or Rust, that uses r2pipe to achieve the same thing
- R2 webserver is also used by r2agent

radare2-webui

R2 -c=H /bin/ls

$ make -C www/w



Create your own web interface for r2

R2 -q -e http.webui=my -e http.root=$PWD -c=H -

Include the r2.js shipped with radare2 in your index.html

ESIL

Evaluable String Intermediate Language

The analysis plugins provide an interface to retrieve the ESIL expression of any instruction for any architecture.

These expressions represent the microcode needed to be executed to perform the same action in a generalistic way.

Designed to be a single chunk of memory, easy to parse, very similar to FORTH.

Can be used for many things

- Searching data patterns in memory that match an esil expression
- Emulating code
- Assisted debugging to predict jumps, values, conditions, ..
- r2wars!

Fix ESIL expressions

Show the ESIL debugger

- What are the ESIL macros?

Side Projects

Other projects tied to r2

- R2env / r2pm
  - Follow up on the talk under this name in the same conference.
- R2frida
  - Checkout the workshop from Murphy!
- R2ghidra
  - Doesnt support callables and pdgo is broken
- Iaito
  - No Qt6 support yet
- R2dec
  - Actively maintained by wargio, in sync with jsdec
- EsilSolver
  - Follow up in the talk in this same conference!

Iaito: The Official GUI?

Iaito was initially written by Hugo Teso, forked by the Cutter team, and then forked back as Iaito to keep supporting r2, looking for maintainers and developers.

- Should support Qt6 (6.2 is LTS already)
- Lacks many widgets and features (asm/disasm, diffing, forensics, r2frida, signatures, r2pm, ..)
- Debugger is pretty slow, unusable and unstable
- Decompilers lock the interface
- C++ is not an ideal language

R2Ghidra

- Supports acr, cmake and meson builds
- Doesnt support function pointers yet, as r2 is not yet handling this atm
- pdgo is broken (regression)
- The ghidra-decompiler fork adds more archs than the ones shipped by ghidra
- Sleigh files are built and released separately in the CI
- Support meson, acr and cmake build systems
- Ghidra C++ code is pretty bad and full of bugs, patches needed and sometimes segfaults appear

Other Decompilers

R2 integrates with other decompilers:

- Jadx
- R2dec
- Retdec
- Snowman

R2 ships its own pseudo-code, that cant be considered a decompiler.

- Aim to improve it and make it the default for the upcoming r2-6.x
- Esil2c allows to convert any function into re-compilable C, useful for retargeting binaries
- Some people used asm.pseudo output as almost-python for emulating algorithms
- Written in C, tightly integrated with the internal structures and fast

Closing

Final Words

Hope you all learned new stuff about r2, if you have any question, please drop them in the chat.

Lots of stuff covered and much more to dig in!

This project is free software, there's no company or


Questions?