FRIDA

hack-a-ton

for DEVREVs

No cON Name

at Barcelona / NoConName
on December 2015
by pancake / oleavr

Who am I?

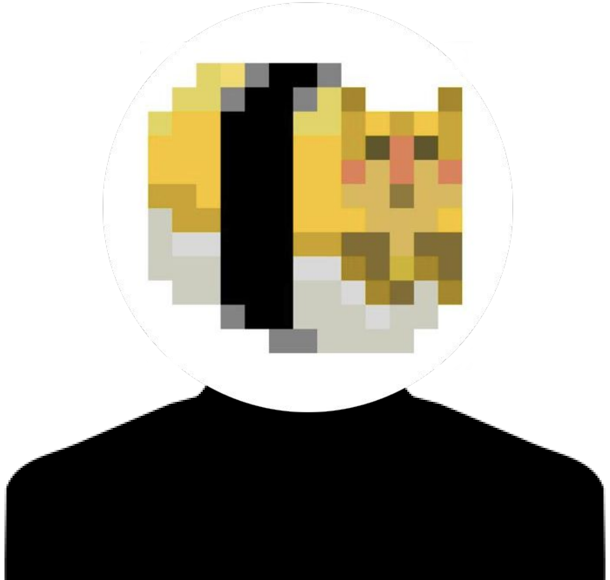
My name is Sergi (also known as pancake/trufae)

Author of Radare, Radare2, Valabind, Acr, sdb...

Developer, Hacker and Reverse Engineer

Currently working at NowSecure

Doing R+D on mobile platforms



History & Today

The radare project started in 2006 as a simple 64bit hexadecimal editor to do forensics from commandline. But quickly evolved to support disassembler, code analyzer, debugger, bindings for many scripting languages, GUI and much more!

Very active project, used by individuals, organizations, companies, CTF teams…

Receiving contributions from a wide range of users every day.

Looking forward r2-1.0 to be released next year (2016)

Who am I with?

My name is Ole André

Author of Frida, CryptoShark, oSpy, libmimic...

Developer, Hacker and Reverse Engineer

Currently working at NowSecure

Doing R+D on mobile platforms

Contact Info

multi-platform hex editor, disassembler, static code analyzer, emulator and webserver

IRC FreeNode/Telegram **#radare**

Twitter **@trufae @radareorg**

FRIDA

multi-platform in-process debugger for scriptable dynamic instrumentation

IRC FreeNode **#frida**

Twitter **@oleavr @fridadotre**

Who Are The DevRevs?

Developers and Reverse Engineers.

Understanding an algorithm requires writing tools to extract information or transform the representation into other forms for proper verification and analysis.

That’s where radare2 and Frida come into play.

License

When you get into programming, choosing a license tends to be the more painful thing to do right after choosing the project name.

r2 is LGPL3

- Some plugins are GPL

Frida is wxWindows License

- aka GPLv2 + static linking exception

- GPL
- LGPL
- BSD
- Apache
- MIT
- PD
- …

See COPYING[.txt] and COPYING-LESSER in case of LGPL.

Linking with GPL variants

GPL has some good ideas, but also some problems.

Defining a new license is possible, but confusing.

Combining multiple licenses is also possible, even with exceptions, but it’s just confusing and you can’t use stickers.

More liberal licenses can be dangerous for free software.

GPL enforces that all derived code must also be published as open source under the same license.

This means that if somebody publishes a tool based on a GPL library or program, the source must also be provided so others can recompile for their platform.

This has nothing to do with privative or private software.

Restriction comes to redistributing binary builds that statically link to GPL code.

How r2 solves the GPL restrictions?

Radare2 was born to be free.

Reverse Engineering is a collaborative task, having **open-source tools** for this enables everyone to contribute with their findings, preferences, needs or ideas.

Most people in RE write their scripts or tools in a standalone way, without maintaining, or publishing it properly, so lot of **work is re-done** again in again to solve the same problems.

Free Software can fix that, but having open tools and algorithms doesn't mean you can't implement private or commercial services on top of it.

r2pipe bypasses any license restriction because those binding APIs use r2 as an external process, only cmd() interface is available.

Scripting can be done in **several ways**, from RLang, r2pipe, using the native bindings generated by Valabind or the dynamic bindings like Ctypes.

LGPL allows dynamic linking of **closed source** apps.

Static linking requires object files to be released. Which doesn't seem like a bad approach to me because r2 remains as an external tool.

Versioning and Releases

Releasing is a process that tends to be more heavy than just placing a tag in a specific commit.

Frida tends to release as soon as breaking changes or useful features come in. Build process may involve code signing for system-level trust, so most users use the latest released binaries.

radare2 always releases late, git is always recommended, stable releases get outdated pretty quickly.

- Rolling Releases
- Periodic Releases
- Feature Releases
- Bugfix Releases
- ...

- Mark blocker issues
- Feature-Freeze
- Reduce changelog for humans
- Update AUTHORS
- Fuzz and claim for testing
- Contact all packagers and distros

Choosing the OS

You can build and run r2 or Frida on any major OS:

- **Linux**
- **OS X**
- **QNX**
- **Windows**

Personally I prefer Linux or OS X, mainly because they are true UNIX systems and build faster than Windows.

Cross-compilation is easy and it is always possible to share a Windows folder to a Linux VM to get better build times.

In addition, r2 runs on any BSD, Haiku and there are plans for both projects to support kernel-land and boot-land.

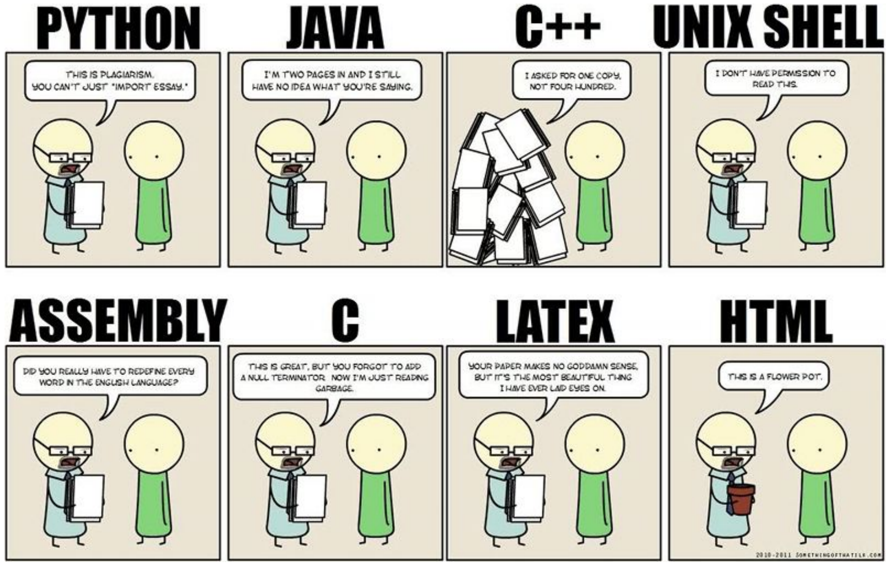
Choosing the Language

Each programming language has its pros and cons, and it's good to evaluate many variables before deciding which one to choose for your project.

- **r2** is pure C, with webui in JS
- **Frida** core is C (Gum), Vala and C++ (V8)

Bindings for Node.js and Python.

- C
- C++
- Vala
- Rust
- Perl
- Node.js
- Go
- Python


PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
C++
I ASKED FOR ONE COPY, NOT FOUR HUNDRED.
UNIX SHELL
I DON'T HAVE PERMISSION TO READ THIS.
ASSEMBLY
DID YOU REALLY HAVE TO REDEFINE EVERY WORD IN THE ENGLISH LANGUAGE?
C
THIS IS GREAT, BUT YOU FORGOT TO ADD A NULL TERMINATOR. NOW I'M JUST READING GARBAGE.
LATEX
YOUR PAPER MAKES NO GODDAMN SENSE, BUT IT'S THE MOST BEAUTIFUL THING I HAVE EVER LAID EYES ON.
HTML
THIS IS A FLOWER POT.

Pros and Cons of C

PRO

- legacy (C99) and portable
- small footprint
- syntax is simple (imperative)
- lots of compilers to choose from
- easy to find developers
- low level and easy to integrate with system
- many debugging and profiling tools
- almost auto-indentable
- what-you-write-is-what-you-get

CONs

- not memory safe
- not thread safe
- easy to write buggy code
- memory leaks
- macros are hacky
- no module/dependency system
- messy build systems
- no integrated documentation/testsuite

Secure Coding assistants

So.. the only way to fix all those CONs from C is by using external tools that point us to the errors in the code..

- Compiler **Warnings**
- Learn from **CERT secure coding style**
- **Defensive Coding** from Fedora Security Team
- **CLANG-Analyzer**
- SCAN.**Coverity**
- **Valgrind** and **MALLOC_OPTIONS** on OpenBSD
- **ASAN** (AddressSanitizer, LeakSanitizer, MemorySanitizer), **UBSan**, **SyzyASan**

Continuous Integration and Build Farms

Build farms compile every single commit performed on a specific repo, it then runs the battery of regression tests and reports the results on irc/telegram/..

Radare2 is currently using those three:

- **Travis**
- **Jenkins**
- **AppVeyor** (native windows builds)

Provide per-commit binaries for all platforms*

***platforms**: Linux32/64, OS X, Android-ARM/x864/ARM64/MIPS

Coding Style

Keeping the same coding style for the whole projects makes the code more readable, consistent and easy to work with.

- **Frida** uses GNOME coding styles (like GStreamer, GTK+, GLib, ..)

- **Radare2** uses sys/indent.sh :D
  - Only tabs for indentation
  - Spaces before parenthesis in function calls or keywords
  - Cases at the same indent level as the Switch

What is Fuzzing

Fuzzing is the art of generating tons of samples that are then loaded by a program in order to catch which inputs produce **unexpected** outputs or **crashes**.

Fuzzing can be done, not only by generating new samples, but also, by injecting code and emulating/running a specific function with different parameters or sending network packets for finding bugs in network stacks or remote services.

Fuzzing

Fuzzing is one of the most important steps when testing software, as long as testsuites can't really cover all the possible control flow paths, fuzzing helps us to find some really smart bugs in a reasonable amount of time.

Some of the most known/used fuzzers are

- **afl** (american fuzzy lop)
- **libFuzzer** LLVM's SanitizerCoverage
- **KLEE** LLVM Execution Engine
- zzuf, **radamsa**, surku ...

TDD: Test Driven Development

Perl is probably the most successful case of TDD.

- Write Tests for Everything
- Write them before writing the implementation.
- Use Coverage tools like kcov, gcov, .. to understand how much tested your code is
- Create as many small modules as possible
- C is not good at this (++ for node or rust)
- Writing tests is boring. Really boring.
- Tests require a lot of time to run (r2 have more than 2000 tests)

RDD: Regression Driven Development

When you have more users than developers, TDD becomes hard to achieve, because features and releases are more important than writing 100% correct and verified code.

- **radare2** uses mainly RDD
- **Frida** uses hybrid TDD/RDD

RDD happens when you write tests after implementing the code or right when someone spots a bug. So, it enforces that every single reported issue should be associated with one or more tests.

Those tests are executed on every commit by the build farm in order to identify when something got broken, this is handy to catch bugs before it’s too late.

Git

Git is the version control system of choice for r2 and frida.

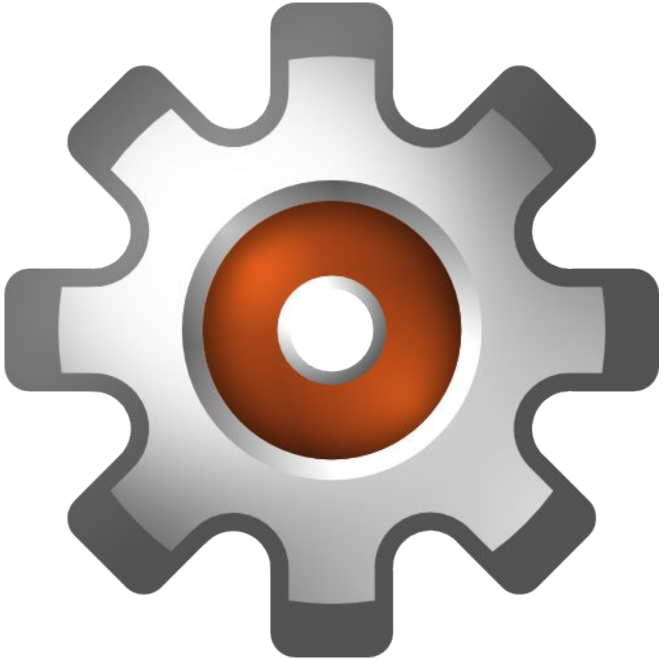
- How to use it
  - `git clone https://github.com/radare/radare2`
- How to fork
  - (in github website click fork button) This will clone the repo into your account.
- How to send a pull request
  - *Create a branch, rebase your changes to master and click the PULLREQ button*
- How to rebase
  - `git cherry-pick, reset --hard, rebase -i @~X`
- How to manage branches
  - `git checkout -b newbranch # create`
  - `git checkout master  # switch`
  - `git branch -D newbranch   # delete`

Issues

Bug trackers are useful tools for developers and users, it’s important to use them properly. Avoiding dupped issues, defining priorities, group them by tags, etc..

- **Milestones**
- **Tags**
- **Assign**
- **Priorities**

For starters see the #easy or #hackaton tags.



Reporting

Reporting issues is also an important part of software development. In many cases, after reading the crashlog the reason and fix for the crash becomes obvious.

- Provide **reproducer** (minimized if possible)
- Use **gdb/lldb** and **valgrind/asan**
- Paste **backtrace** in case of crash
- Register state (values and contents of **memory** if references)
- Version used (always test **latest from git**) including arch/os info
- A test for **r2r** is welcome!
- Avoid duplicates, issues surfing, **referencing**
- Propose **patch** if possible
- Feedback and **fix times** matter

Code Organization

r2 is a rewrite of the monolithic r1, main purpose was to modularize the code for proper reusability, clarify the api dependencies, support plugins and bindings.

- Split the problem into **modules**
- Use same code **style** everywhere
- Create **tests** and verify
- Continuous **refactoring**
- Split implementations into **plugins**

Radare2 Code Organization

- **libr/**
  - asm, anal, core, flags, debug, reg, lang, io, ...
- **libr/*/p**
  - plugins for each library
- **binr/**
  - rax2, radare2, rabin2, r2pm, radiff2, ...
- **shlr/**
  - grub, capstone, rar, windbg, udis86, sdb, ...
- **radare2-regressions/**
  - after `make tests`

Radare2 installation

- `sys/install.sh`
- `sys/user.sh` (home installations)
- `sys/asan.sh` (ASAN builds, env ASAN='leak memory')
- `sys/afl.sh` (american fuzzy lop)
- `sys/build-m32.sh`
- `sys/clang-analyzer.sh`
- `..`

Frida Code Organization

- frida-gum - low level APIs for hooking, code manipulation, etc.
- frida-core - server, agent, gadget, binding APIs, etc.
- frida-node - host Node.js API
- frida-python - host Python API

Frida Installation

Command-line Tools

```
    $ sudo pip install frida
```

Node.js APIs

```
    $ npm install frida
```

Hackatons

Hackatons are competitions between individuals or groups in order to achieve some specific **features** implemented or **bugs** fixed.

As long as time and resources are limited, the tasks should be simple or reasonable to be done. We have selected some of them ready for you to pick them up.

We will be attending all your **questions** and discuss or recommend the details needed to perform the tasks.

Tasks for Radare2

- **Assemblers**
  - Improve x86, ARM and ARM64
- **Disassemblers**
  - Flash (rbin.swf and asm.swf)
  - .NET (rbin.pe.msil and r2e/libr/asm/msil)
  - WebAssembly
- **Enhance the WebUI**
  - Focus on /p and /m (desktop + mobile)
  - Expose more info
  - Add more interaction
  - New widgets
- **RBin file formats**
  - Optimize fatmach0 and dyldcache (rabin2 -x)
  - Full support for OAT/ART android binaries
  - Crashdump support (linux/osx core)
  - Minidump support (already in r2e)

Participants must be fluent in C or JS depending on the task.

Tasks for Frida

- **Documentation**
  - API reference for Node.js and Python bindings
  - Translations to other languages
- **Instrumentation Engine**
  - Fix thread enumeration on Android
  - Expose profiler API to the JS runtime
- **Node.JS**
  - Implement npm modules for Frida
  - Implement npm-like tool only for Frida packages
- **Python**
  - Integrate already existing py2js to allow running python code in target
  - Enhance REPL to show inline documentation
- **JavaScript**
  - Add support for creating Java classes at runtime
  - Read-write Objective-C fields at runtime
  - Add support for Swift introspection

Participants must be fluent in C or JS depending on the task.

FRIDA

Demo Required?

Quick Radare2 / Frida introductory demo

Questions?

Yes / No

Repositories

frida.re

- https://github.com/frida/frida
- https://github.com/frida/frida-core
- https://github.com/frida/frida-gum
- https://github.com/frida/frida-python
- https://github.com/frida/frida-node

radare.org

- https://github.com/radare/radare2
- https://github.com/radare/radare2-extras
- https://github.com/radare/radare2-regressions
- https://github.com/radare/radare2-bindings

References

- https://fuzzing-project.org/
- http://lcamtuf.coredump.cx/afl/
- https://klee.github.io/
- http://llvm.org/docs/LibFuzzer.html
- https://github.com/google/sanitizers
- https://scan.coverity.com/
- http://clang-analyzer.llvm.org/
- https://docs.fedoraproject.org/en-US/Fedora_Security_Team/1/html/Defensive_Coding/index.html
- https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards