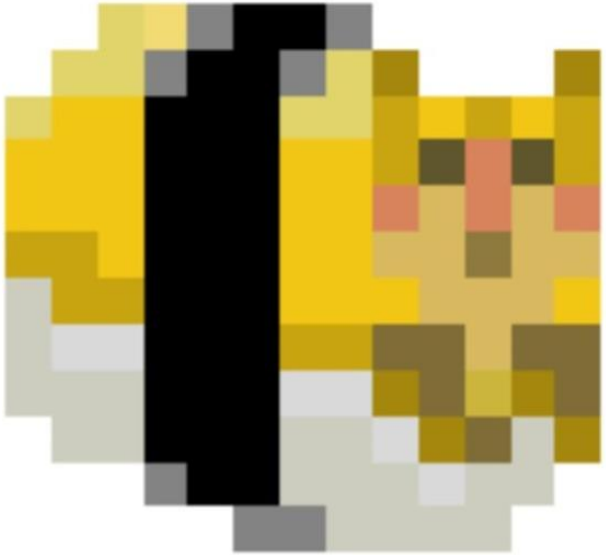
Reversing with Radare2

pancake@OverdriveCon2016

Who am I?



pancake aka Sergi Alvarez i Capilla

Twitter: @trufae @radareorg

Web: http://rada.re

Currently working as a Mobile Security Analyst at NowSecure, author of **radare** and many other open-source tools, also worked as a Forensic Analyst, full-stack developer, embedded firmware hacker, teacher and eventual reverse engineer.

What is Reversing?

Understanding the internal mechanisms in a piece of software or hardware in order to:

- Find vulnerabilities
- Bypass security protections (cracks/exploits)
- Extend its functionalities
- Understand how it works
- Find hidden features
- Fix bugs

What is Radare2?

Free and open-source **hexadecimal editor**, disassembler and debugger created by me in 2006 aiming to be modular, pluggable and orthogonal. Major rewrite of radare. (10yo)

Follow some of the UNIX design principles, written in C, portable, scriptable, orthogonal, flexible and very active project with a great community.

Release every 6 weeks. About 50 contributors on each release.

r2con is the congress around radare2. 120 attendees in the first edition (2016).

What is exactly Radare2?

Framework to ease several reverse enginering and other low level tasks.

- Composed by a bunch of libraries written in C
  - Automatic bindings generation with valabind
  - Extensible via plugins
- Provides different tools that make use of them
  - ls /usrbin/r*2
  - Extensible via scripts
- Portable as hell (stick to posix and requires at least 1MB of disk)
  - Supports native and remote targets without needing recompilation

What can r2 do for me?

- Better ask yourself about what it can’t do

What Can It Do?

- Disassemble binaries of several architectures, operating systems.
- Analyze code, data, references, structures, …
- Debugging, tracing, exploiting, …
- Binary manipulation, code injection, patching, “optimizing”, …
- Mount filesystems, detect partitions, carve for known file formats, …
- Extract information and metrics from binaries for classification
- Find differences between two files
- Compute checksums of the blocks in a file
- Kernel analysis and debugging
- Play 2048 and even Order pizzas online

Plugins

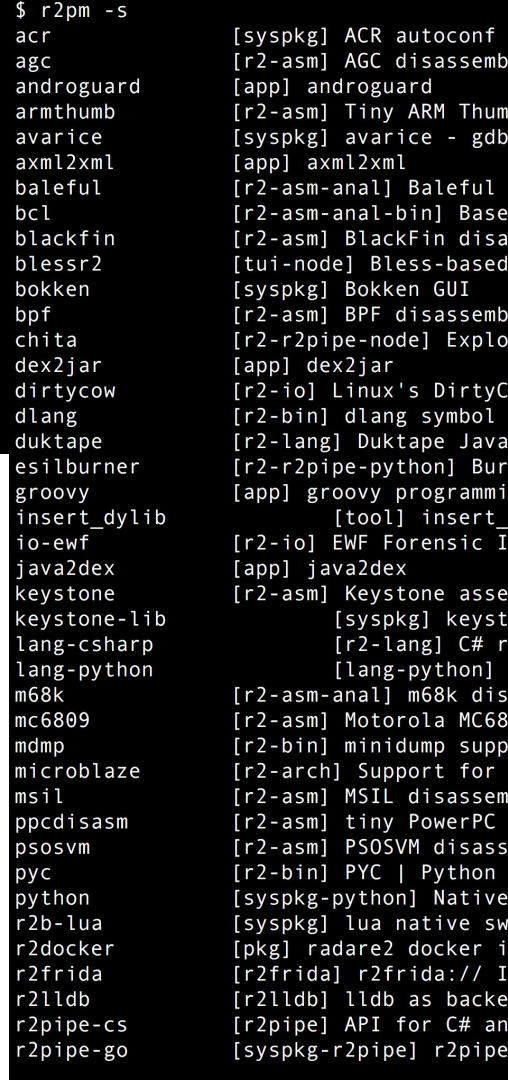
- Understand a lot of file formats (rabin2 -L)
  - Even corrupted ones!
- Assemble/Disasm many CPUs (rasm2 -L)
  - Tune it via asm.arch, asm.bits and asm.cpu
- IO plugins abstract filesystem access (r2 -L)
  - Handle ptrace/remoting/kernel/sockets/…
- Debugger plugins (r2 -qcdh --)
  - Bochs, GDB, Native, Remote, …
- Crypto / Checksums (rahash2 -L)

```
$ rasm2 -L
_dAe  8 16       6502        LGPL3   6502/NES/C64/Tamagotchi/T-1000
_dA_  8          8051        PD      8051 Intel CPU
_dA_  16 32      arc         GPL3    Argonaut RISC Core
a___  16 32 64   arm.as       LGPL3   as ARM Assembler (use ARM_AS en
adAe  16 32 64   arm         BSD     Capstone ARM disassembler
_dA_  16 32 64   arm.gnu     GPL3    Acorn RISC Machine CPU
_d__  16 32      arm.winedbg LGPL2   WineDBG's ARM disassembler
adAe  8 16       avr         GPL     AVR Atmel
adAe  16 32 64   bf          LGPL3   Brainfuck
_dA_  16         cr16        LGPL3   cr16 disassembly plugin
_dA_  32         cris        GPL3    Axis Communications 32-bit embe
adA_  32 64      dalvik      LGPL3   AndroidVM Dalvik
ad__  16         dcpu16      PD      Mojang's DCPU-16
_dA_  32 64      ebc         LGPL3   EFI Bytecode
adAe  16         gb          LGPL3   GameBoy(TM) (z80-like)
_dAe  16         h8300       LGPL3   H8/300 disassembly plugin
_d__  32         hppa        GPL3    HP PA-RISC
_dAe             i4004       LGPL3   Intel 4004 microprocessor
_dA_  8          i8080       BSD     Intel 8080 CPU
adA_  32         java        Apache  Java bytecode
_d__  32         lanai       GPL3    LANAI
_d__  8          lh5801      LGPL3   SHARP LH5801 disassembler
_d__  32         lm32        BSD     disassembly plugin for Lattice
_d__  32         m68k        BSD     Capstone M68K disassembler
_dA_  32         malbolge    LGPL3   Malbolge Ternary VM
_d__  16         mcs96       LGPL3   condrets car
adAe  16 32 64   mips        BSD     Capstone MIPS disassembler
adAe  32 64      mips.gnu    GPL3    MIPS CPU
_dA_  16         msp430      LGPL3   msp430 disassembly plugin
_dA_  32         nios2       GPL3    NIOS II Embedded Processor
_dAe  8          pic18c      LGPL3   pic18c disassembler
_dA_  32 64      ppc         BSD     Capstone PowerPC disassembler
_dA_  32 64      ppc.gnu     GPL3    PowerPC
ad__             rar         LGPL3   RAR VM
_dA_  32 64      riscv       GPL     RISC-V
_dAe  32         rsp         LGPL3   Reality Signal Processor
_dA_  32         sh          GPL3    SuperH-4 CPU
_dA_  8 16       snes        LGPL3   SuperNES CPU
_dAe  32 64      sparc       BSD     Capstone SPARC disassembler
_dA_  32 64      sparc.gnu   GPL3    Scalable Processor Architecture
_d__  16         spc700      LGPL3   spc700, snes' sound-chip
```

r2pm: Package Manager

Provides an easy way to install dependencies and plugins for r2 in the user home directory or system wide.

- KeyStone assembler
- RetDec decompiler
- Unicorn emulator
- Disassemblers for more architectures
- R2 api bindings
- And more!...



Introducing the Shell

The main interaction is happening in the shell. R2 offers a powerful and expressive (but sometimes confusing) way to run commands.

The user usually needs to learn less than 10 commands to do most of the common tasks, so it’s not really an excuse to not learn it.

Let’s see some very basic introduction before going into the practice.

Basic Commands

- **Move**: ‘s’ stands for seek, use @ for temporal seeks
- **Hexdump:** x
- **Disasm**: pd
- **Write Hexpairs**: wx
- **Write Assembly**: wa
- **Analyze All Code**: aa
- **Help**: append ‘?’ to any command
- **Quit**: q

Solving a Crackme

(demo)

- Explain basic commands
- How to get help
- Explain visual mode
- Strings with rabin2 -qz
- Extract the password
- Patch to make it always accept the password

Extract Information

Rabin2 and the i command

- Entrypoint (rabin2 -e)
- Symbols (-s)
- Imports (-i)
- Libraries (-l)
- Strings (-qz)
- Relocs (-r)

```
$ rabin2 -I /bin/ls
havecode true
pic      true
canary   true
nx       false
crypto   false
va       true
intrp    /usr/lib/dyld
bintype  mach0
class    MACH064
lang     c
arch     x86
bits     64
machine  x86 64 all
os       osx
minopsz  1
maxopsz  16
pcalign  0
subsys   darwin
endian   little
stripped false
static   false
linenum  false
lsyms    false
relocs   false
binsz    38512
```

Forensics

The original objective of this tool was to serve as a computer forensics tool to search for patterns in a hard disk or memory dump and recover information from there.

- Support partitions and several filesystems (GRUB)
- File magic functionality integrated
- Parse file format headers and data structures
- Print data in different formats, raw, base64, hex
- Compute and compare per block checksums.
- Binary diffing and entropy calculations

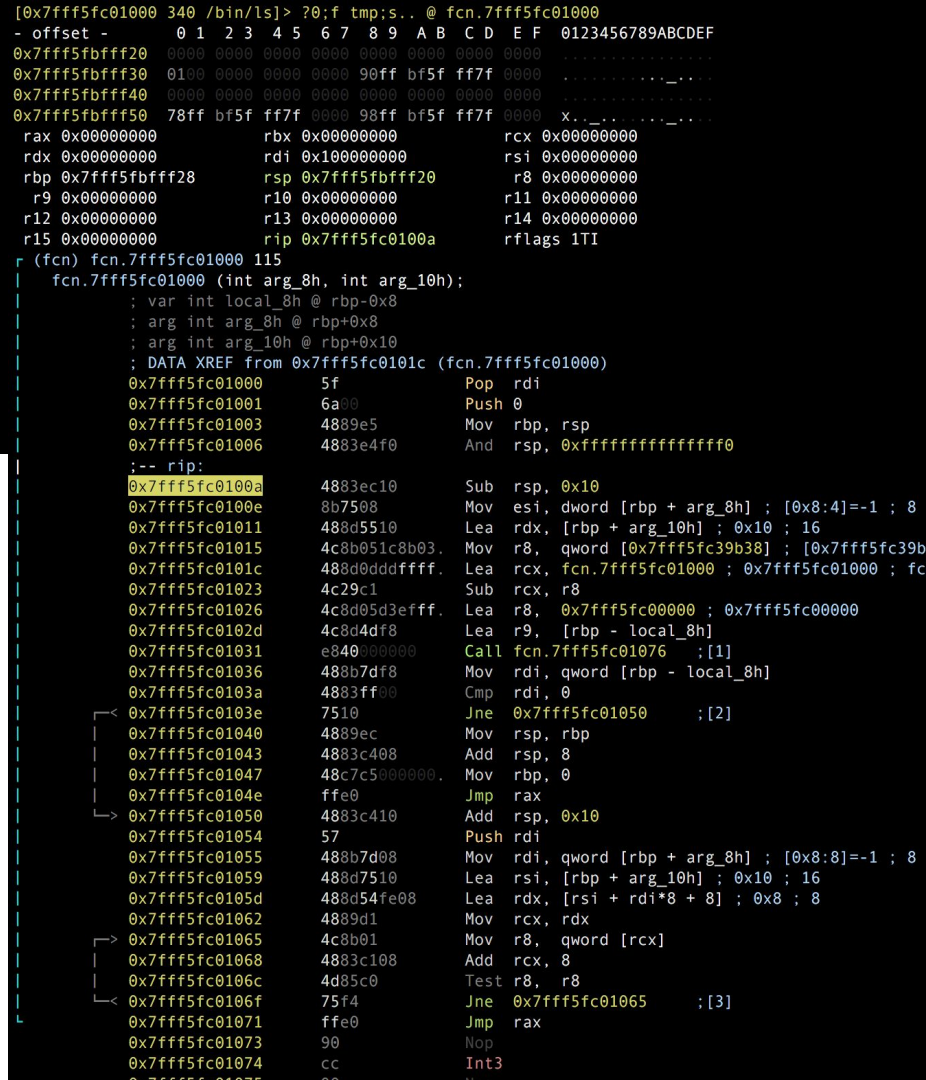
Graphing

- Graph Basic blocks
- Branch Lines
- Graph Calls / Refs
- Color Schemes
- Entropy
- Section Ranges
- Exploration Bar

Debugging

Running a program or attaching to a process

- read/write registers
- read/write memory and list maps
- step/breakpoints/continue
- stack telescoping,
- heap analysis
- code injection
- file descriptor manipulations



Frida, LLDB, Bochs, WinDBG, …

Also an option as debuggers backends for radare2.

- Frida is a dynamic programmable tracer and code injection framework
  - More expressive shell (not just js)
  - Static analysis and
  - Low level code patching and injection
- LLDB is de-facto debugger in the Apple ecosystem
  - Debug iWatch, OSX or iOS apps without jailbreak via r2lldb
  - Much better disassembly
- Bochs/GDBServer/WinDBG… just as remote debuggers

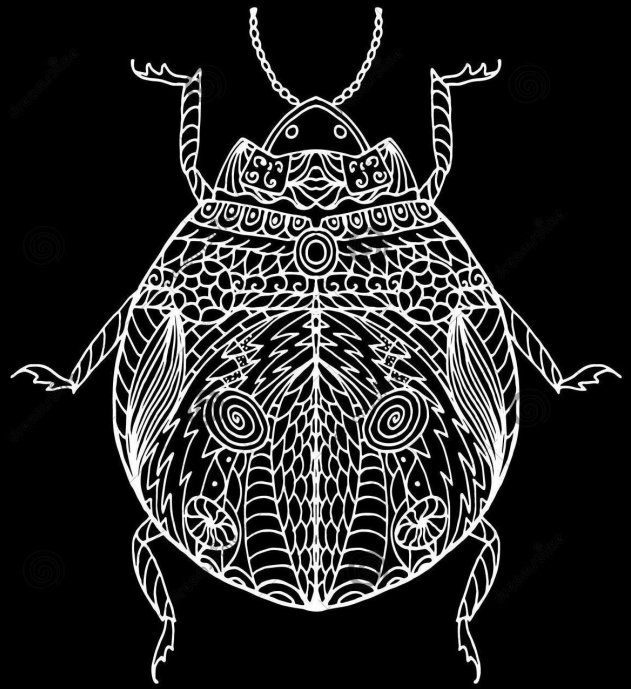
r2frida Demo

Disabling features in Twitter at runtime

```
[0x100e957fd]> =!ic TwitterAPI~00e957fd

0x0000000100e957fd - didGetSearchResults:info:

[0x100e957fd]> "wa xor rax,rax;ret"

[0x100e957fd]> wx 554889e5
```

Debugging Demo

- Step Into/Over
- Change program counter
- Visualize the stack contents
- Tracing
- Using breakpoints



rarun2

Tool to define execution profiles to specify program environment, arguments, permissions, directories, input/output, etc. This tool and APIs are used by the debugger.

```
$ man rarun2
```

```
$ rarun2 > target.rr2
```

```
$ r2 -R target.rr2
```

Exploiting The Dirty COW (CVE 2016-5195)

```
$ r2pm -i dirtycow

$ r2 dcow:///etc/services

> 30w you are vulnerable

$ head /etc/services
```

(demo with an old livecd)

Analysis

- Identify functions
- Function signatures (zignatures)
- List them
- Find references
- Detect local variables
- Stack accesses
- Resolve function signatures
- Resolve syscalls

```
[0x100001174]> ao 10
address: 0x100001174
opcode: push rbp
prefix: 0
id: 588
bytes: 55
refptr: 0
size: 1
type: upush
esil: rbp,8,rsp,-=,rsp,=[8]
stack: inc
family: cpu
,address: 0x100001175
opcode: mov rbp, rsp
prefix: 0
id: 449
bytes: 4889e5
refptr: 0
size: 3
type: mov
esil: rsp,rbp,=
stack: null
family: cpu
,address: 0x100001178
opcode: push r15
prefix: 0
id: 588
bytes: 4157
refptr: 0
size: 2
type: upush
esil: r15,8,rsp,-=,rsp,=[8]
stack: inc
family: cpu
,address: 0x10000117a
opcode: push r14
prefix: 0
id: 588
bytes: 4156
refptr: 0
size: 2
```

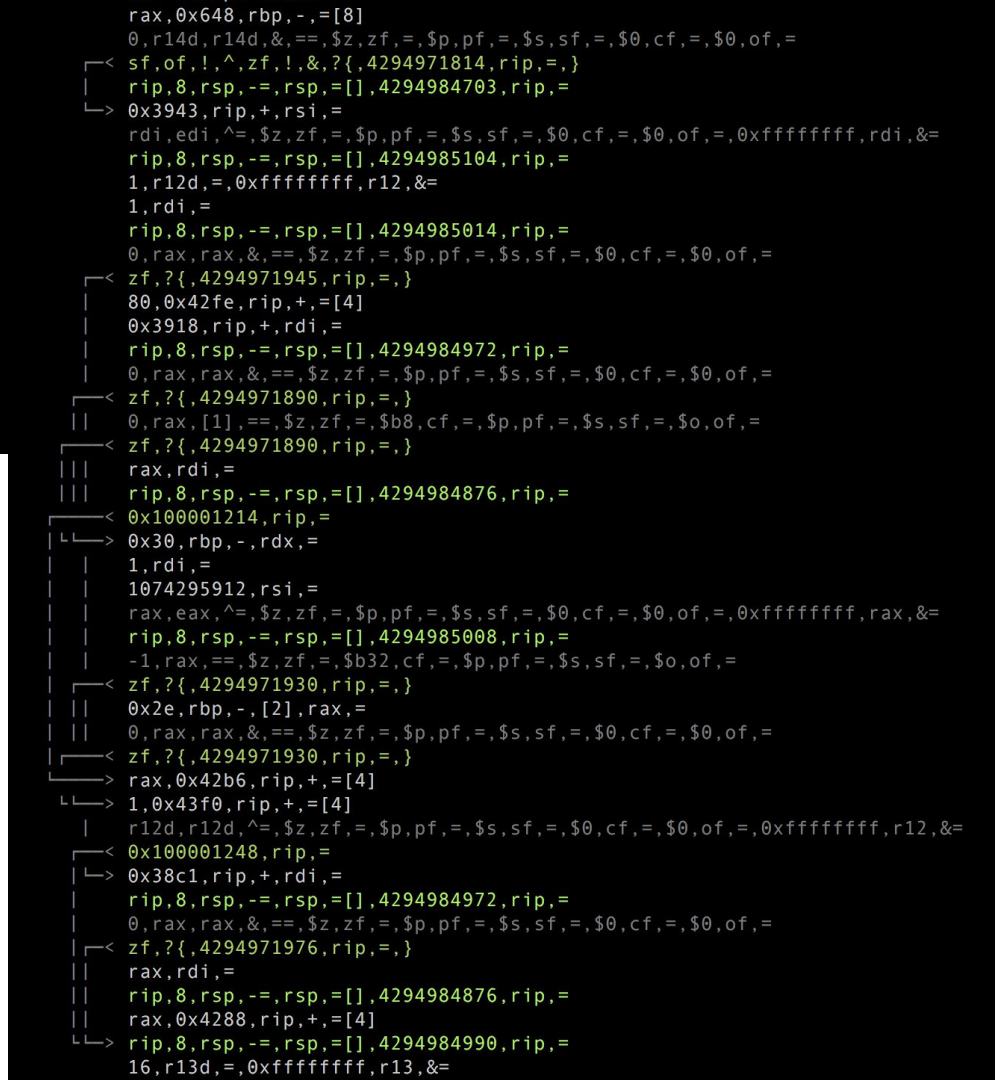
Analysis Demo

- Analyze binary, find references to strings
- Listing functions
- Cyclomatic complexity
- Enumerate syscalls in Go binary
- Resolve strings (objc, or Go)

ESIL

Evaluable Strings Intermediate Language

- Code emulation
- Branch prediction
- Find read/write register in functions
- Resolve Syscalls
- Assisted Debugging
- Complex search queries



ESIL demo

Use ESIL to resolve a strings that are computed in more than one instruction.

Resolve crackme using ESIL.

- Initialize stack with aeim, Visualize stack
- Set program counter to sym._checkPassword
- Step into the decrypt loop until the string is clear

Scripting

The ability to automate a sequence of actions:

- Scripting using r2 commands
- Using r2pipe (available for lot of languages)
- Using Native bindings (not recommended)
- Using RLang (#! hashbang)

```
var r2pipe = require ("../");

function doSomeStuff(err, r2) {

  r2.cmdj ("aij entry0+2", function(err, o) {
    console.log (o);
  });

  r2.cmd ('af @ entry0', function(err, o) {
    r2.cmd ("pdf @ entry0", function(err, o) {
      console.log (o);
      r2.quit ()
    });
  });

}


r2pipe.pipe ("/bin/ls", doSomeStuff);
r2pipe.launch ("/bin/ls", doSomeStuff);
r2pipe.connect ("http://cloud.rada.re/cmd/", doSomeStuff);
```

Interpreting r2 scripts

Using the -i flag and the . command

- Conditionals
- Macros
- Quoted commands
- Comments

Python, Node, C#, Ruby..

The most recommended command bindings for scripting.

Wraps access to r_core_cmd_str(), provides JSON helpers

- Native Backend
- Remote via HTTP
- Pipes
- Sockets

Also possible to write asm and io plugins as well as interacting with the debugger

r2pipe demo

Using Python and NodeJS to interact with r2.

- Run commands
- Parse output JSON
- Sync/Async
- Pipe/TCP/HTTP/RAP/Native

Other languages:

- C#, Go, Vala, Java, Rust Ruby, Lisp, Erlang, Swift, Ocaml, ..

```perl
// Perl
use Radare::r2pipe;
my $r2 = Radare::r2pipe->new("/bin/ls");
print $r2->cmd("?V");
$r2->quit();
```

```lisp
// Lisp
(setf r2 (r2pipe "/bin/ls"))
(format t "~s~%" (r2-cmd r2 "?V")
(r2-quit r2)
```

```nodejs
// NodeJS-Async
const r2pipe = require('r2pipe');
r2pipe.open('/bin/ls', (err, r2) => {
  r2.cmd('?V', console.log);
});
```

```python
// Python-Sync
import r2pipe
r2 = r2pipe.open("/bin/ls")
print r2.cmd("?V")
```

```swift
// Swift-Sync
if let r2p = R2Pipe(url:"/bin/ls") {
  if let str = r2p.cmdSync("?V") {
    print("\(str)");
  }
}
```

```java
// Java
import org.radare.r2pipe.R2Pipe;
R2Pipe r2p = new R2Pipe("/bin/ls");
System.out.println(R2p.cmd("?V");
r2p.quit();
```

```groovy
// Groovy
def r2 = new R2(r2: new R2Pipe("/bin/ls"))
println r2.cmd("?V");
r2.quit();
```

```golang
import r2pipe
r2p, err := NewPipe("/bin/ls")
defer r2p.Close()
buf, err := r2p.Cmd("?V")
fmt.Printf("%s\n", buf);
```

User Interfaces

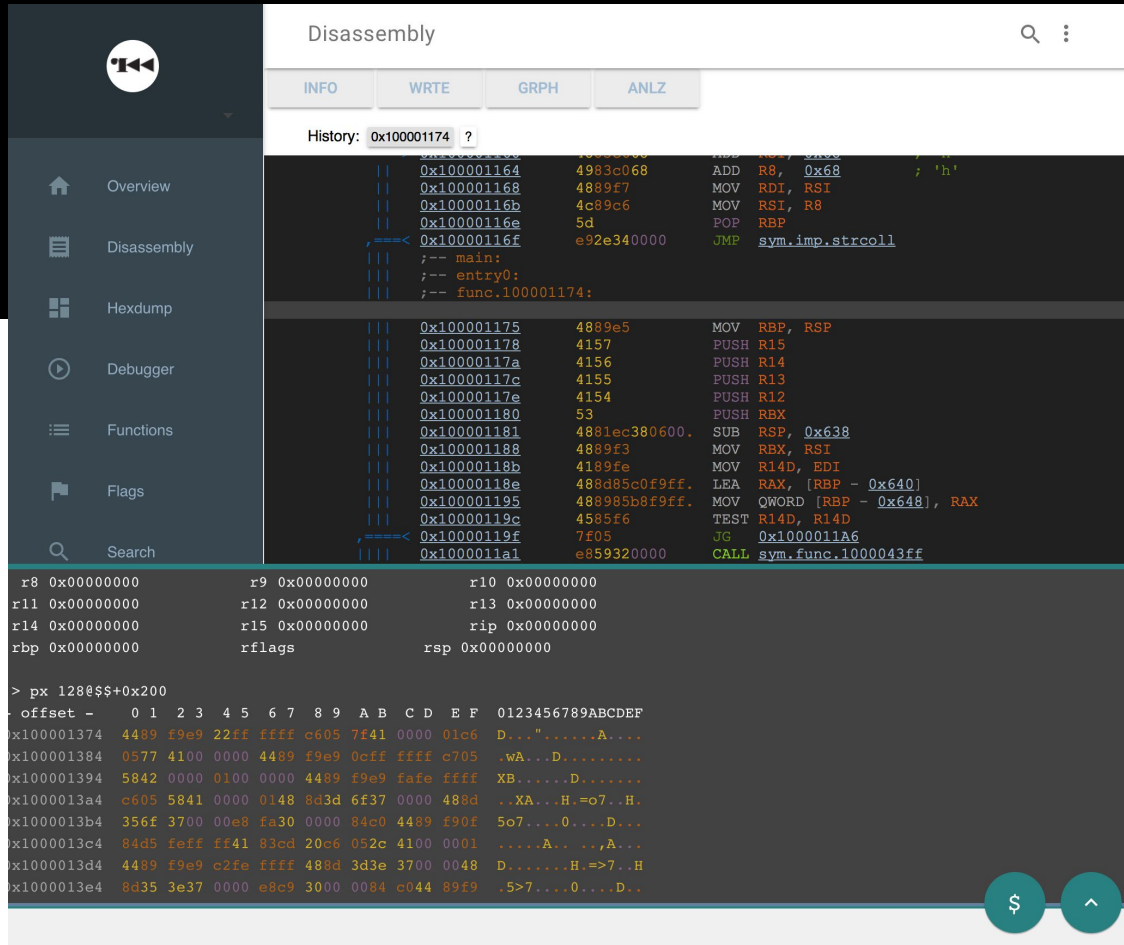
- Console modes (Prompt, Visual, Graph, Panels, Columns)
- WebUI (r2 -c=H) Android Material Design
- Native User Interfaces (c#-mfc, qt, gtk2/3, ..)
  - Most of them unreleased, unstable, limited or unmaintained
  - Gradare, Ragui, Bokken, ..

Console will always be more complete than any GUI.

Looking for web developers!

WebUI demo

(demo)



Questions?

EOF