Radare from A to Z

pancake // NN2015
@trufae

Introduction

What Am I Doing Here?

- What is r2?
- How to use the shell
- Analyzing
- Debugging
- Patching
- Scripting

Why Radare2?

- It’s free and opensource
- Runs everywhere (Windows, Mac, Linux, QNX, iOS, ..)
- Easy to script and extend with plugins
- Embeddable
- Grows fast
- Supports tons of file-formats
- Handles gazillions of architectures
- Easy to hack
- Commandline cowboy-friendly
- Great community and even better leader
- Collaborative

What's Radare2?

- Reverse Engineering
  - Analyze Code/Data/..
  - Understanding Programs
- Low Level Debugging
  - Similar to olly
  - Multi-platform, and support for remote
- Forensics
  - File Systems
  - Memory Dumps
- Assembler/Disassembler
  - Several architectures
  - Multiplatform

Tools

Radare2 is composed by some core libraries and a set of tools that use those libraries and plugins.

radare2	r2pm	rarun2	ragg2
rabin2	radiff2	rax2	rahash2
rasm2	rafind2	r2agent	rasign2

History

Radare was born in 2006 as a forensic tool for performing manual and interactive carving to recover some files from disk or ram.

It grew quickly adding support for disassembler, debugger, code analyzer, scripting, …

And then I decided to completely rewrite it to fix the maintainance and monolithic design problems.

But First.. A Poll!

(who are you?)

Which is your main OS?

Do you know assembly?

How’s your UNIX foo?

Did you used r2 before?

Installation

(always use git)

PROTIP: Installing radare2 is recommended method to use it.

How To Install Radare2

There are several binary distributions of radare2

- LiveCD
- OSX package.
- Windows Installer (and nightly builds)
- BSD || GNU/Linux (Gentoo, ArchLinux, Void, ..)
- Use the Cloud Web user interface
- Chat with the @r2bot on Telegram

Coming soon: PPA/Windows from Travis/AppVeyour

Installing from Git

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

$ cd radare2

$ sys/install.sh

   or

$ sys/user.sh
```

Package Management

```
$ r2pm -i radare2

$ rm -rf radare2
```

You can also install other programs, plugins and scripts with it. It aims to ease the identify

Package Management

Some of the most interesting packages:

- Yara (2 / 3)
- RetDec decompiler (@nighterman)
- Unicorn - code emulator
- Native Python bindings
- Duktape (Embedded javascript)
- Radeco decompiler (@sushant94)
- Baleful (SkUaTeR)
- r2pipe apis for NodeJS, Python and Ruby
- Vala/Vapi/Valabind/Swig/Bokken/...

Basic Commands

Seeking

Change current position, accepts flags, relative offsets, math ops. Use @ for temporal seeks.

Printing

Show current block (b) bytes, instructions, metadata, analysis, ...

Writing

Write string, hexpairs, file contents, instructions, etc..

Spawning an R2 Shell

The `r2` command is a symlink for `radare2`.

```
$ r2 -          # alias for `radare2 malloc://1024`
$ r2 --         # open r2 without any file opened
$ r2 /bin/ls # open this file in r2
$ r2 -d ls      # start debugging
```

Other Useful Command Line Flags

```
-h           # get halp message
-a <arch>    # specify architecture (RAsm Plugin name)
-b <bits>    # specify 8, 16, 32, 64 register size in bits
-c <cmd>     # run command
-i <script>  # include/interpret script
-n           # do not load rbin info
-L           # list io plugins
```

In The Shell

Syntax of the commands:

```
> [repeat][command] [args] [@ tmpseek] [; ...]


> 3x                # perform 3 hexdumps

> pd 3 @ entry0     # disasm 3 instructions at entrypoint

> x@rsp;pd@rip      # show stack and code
```

The Internal Grep

As long as r2 is portable, it doesn’t depends on other programs, so there are some basic unix commands, as well as an internal grep/less.

```
> pd~call

> is~test
```

Flags and Calculations

Flags are used to specify a name for an offset.

Math expressions evaluate those names to retrieve the number.

```
> ? 1+1

> f foo = 1024

> ? foo+123
```

Printing Bytes

R2 is an block-based hexadecimal editor. Change the block size with the ‘b’ command.

p8	print hexpairs
px	print hexdump
pxw/pxq	dword/qword dump
pxr	print references

Structures

pf - define function signatures

Can load include files with the t command.

010 templates can be loaded using 010 python script.

Load the bin with r2 -nn to load the struct/headers definitions of the target bin file.

Use pxa to visualize them in colorized hexdump.

Disassembling

(and printing bytes)

Disassembling is the “art” of translating bytes into meaningful instructions.

Disassembling Code

```
pd/pD - disassemble N bytes/instructions.

pi/pI - just print the instructions

pid   - print address, bytes and instruction

pad   - disassemble given hexpairs

pa    - assemble instruction
```

Disassembling Code

```
> e asm.emu=true    - emulates the code with esil and
> agv/agf.          - render ascii art or graphviz graph
```

Seek History   s- (undo)   s+ (redo)

Use u and U keys to go back/forward in the visual seek history.

rasm2

Disassembling and assembling code can be done with pa/pad or using the rasm2 commandline tool.

```
$ rasm2 -a x86 -b 32 nop

$ rasm2 -a x86 -b 64 nop
```

(demo)

Binary Info

(parsing fileformats)

RBin detects file type and parses the internal structures to provide symbolic and other information.

RBin Information

```
$ rabin2 -s
> is
> fs symbols;f
```

Symbols	Relocs	Classes	Entrypoints
Imports	Strings	Demangling	Exports
Sections	Libraries	SourceLines	ExtraInfo

RBin Information

All this info can be exported in JSON by appending a ‘j’.

(DEMO)

Scripting

(automation)

The art of automating actions in r2 using your favourite programming language (or not).

Scripting

- Shellscript (batch mode)
  - Use ‘jq’ to parse json output
  - Send commands via stdin
- Bindings (full api)
  - Also supports Python, Java, ...
- Plugins
  - Loaded from home and system directories
- R2Pipe scripts
  - spawn/pipe/http/…
  - NodeJS / Python / Perl / Ruby / Rust / Go / Swift / …
  - Interpreted with ‘.’ command

Using R2Pipe For Automation

R2 providws a very basic interface to use it based on the cmd() api call which accepta a steing with the command and returns the output string.

```
$ pip install r2pipe

$ r2 -qi names.py /bin/ls

$ cat names.py
```

Analyzing Code

(and graphing)

Analyzing is the “art” of understanding the purpose of a sequence of instructions.

Analyzing From The Metal

R2 provides tools for analyzing code at different levels.

ae - emulates the instruction (microinstructions)

ao - provides information about the current opcode

afb - analyze the basic blocks

af - analyzes the function (or a2f)

ax - code/data references/calls

Analyzing the Whole Thing

Many people is used to the IDA way: load the bin, expect all xrefs, functions and strings to magically appear in there.

R2 will not do this by default because it can be slow, tedious, and 99% of the time we can solve the problem quicker with direct and manual analysis.

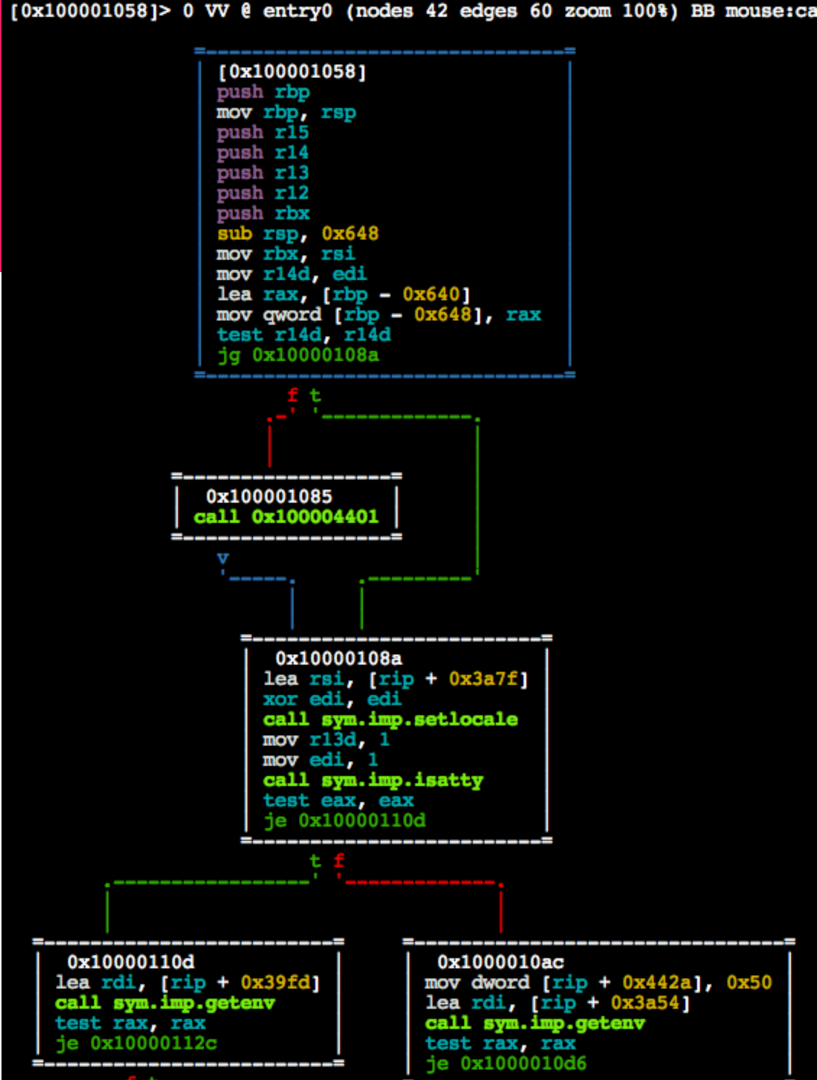
Run `r2 -A` or use the ‘aa’ subcommands to achieve this.

Graphing Code

Functions can be rendered as an ascii-art graph using the ‘ag’.

Enter visual mode using the V key

Then press V again to get the graph view.



Signatures

(and graphing)

Signatures is the "art" of identifying functions by looking at byte patterns.

Signatures

```
aap - function preludes

z*  - Zignatures!  (supports FLIRT and r2's own format)


$ r2 -A static-bin

> zg lebin > lebin.r2
```

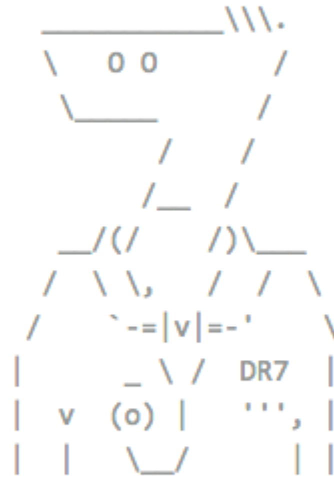
BinDiffing

(and graphing)

Finding differences between two binaries looking for bugfixes.

Finding the Bugfix

(DEMO)



https://www.nowsecure.com/blog/2015/09/30/doctor-seven-osx-vulnerability/

Debugging

(and emulation)

R2 supports native debugger for Linux, BSD, XNU and Windows.

But there’s more!

First Steps

R2 is a low level debugger (not a source debugger).

It provides much more low level information than source debuggers use to provide. Doesn't competes with GDB/LLDB.

Basic Actions for a debugger are:

ds	step	db	breakpoint	dr	show regs
dso	step over	dcu	continue-until	dx	code-inject
dc	continue	dm	memory-maps	dd	file-desc

Remote Debugging

R2 supports WINDBG, GDB and native remote protocols. But, as long as r2 runs everywhere it is recommended to use it in place.

ESIL

ESIL stands for Evaluable Strings Intermediate Language.

A forth-like language (stack based language) using comma as a tokenizer and used for emulating and analyzing code.

Widely used for decrypting malware routines and analyzing shellcodes and other payloads.

```
mov eax, 33        =>        33,eax,=
```

User Interface

- WebUI
- Bokken
- Visual Mode
- Visual Panels
- Commandline
- R2Pipe
- Colors!

Colors!

```
> e scr.color=true

> e scr.rgb=true

> e scr.truecolor=true

> e scr.utf8=true


> ecr    # Random colors

> eco X  # Select color palette
```

```
[0x100001058]> ecr
[0x100001058]> pd 20
            ;-- main:
            ;-- entry0:
            0x100001058      55              push rbp
            0x100001059      4889e5          mov rbp, rsp
            0x10000105c      4157            push r15
            0x10000105e      4156            push r14
            0x100001060      4155            push r13
            0x100001062      4154            push r12
            0x100001064      53              push rbx
            0x100001065      4881ec480600.   sub rsp, 0x648
            0x10000106c      4889f3          mov rbx, rsi
            0x10000106f      4189fe          mov r14d, edi
            0x100001072      488d85c0f9ff.   lea rax, [rbp - 0x640]
            0x100001079      488985b8f9ff.   mov qword [rbp - 0x648], rax
            0x100001080      4585f6          test r14d, r14d
        ,=< 0x100001083      7f05            jg 0x10000108a
        |   0x100001085      e877330000      call 0x100004401
        |      ^- 0x100004401() ; main
        `-> 0x10000108a      488d357f3a00.   lea rsi, [rip + 0x3a7f]
ng ; section.4.__cstring
            0x100001091      31ff            xor edi, edi
            0x100001093      e806350000      call sym.imp.setlocale
               ^- 0x10000459e() ; sym.imp.setlocale
            0x100001098      41bd01000000    mov r13d, 1
            0x10000109e      bf01000000      mov edi, 1
[0x100001058]>
```

Visual Mode

Type V and then change the view with ‘p’ and ‘P’

```
[0x100001072 0% 125 /bin/ls]> f tmp;sr s.. @ main+26 # 0x100001072
0x7fff5fbff8b0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x7fff5fbff8c0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x7fff5fbff8d0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x7fff5fbff8e0  0x00000000 0x00000000 0x00000000 0x00000000  ................
 rax 0x7fff5fbff8e0      rbx 0x7fff5fbfff48      rcx 0x7fff5fbfff60
 rdx 0x7fff5fbfff58      rdi 0x00000001          rsi 0x7fff5fbfff48
 rbp 0x7fff5fbfff20      rsp 0x7fff5fbff8b0       r8 0x00000000
  r9 0x7fff5fbfefd0      r10 0x00000032          r11 0x00000246
 r12 0x00000000          r13 0x00000000          r14 0x00000001
 r15 0x00000000          rip 0x100001079         rflags = 1TI
            0x100001072    488d85c0f9ff.   lea rax, [rbp - 0x640]
            ;-- rip:
            0x100001079    488985b8f9ff.   mov qword [rbp - 0x648], rax
            0x100001080    4585f6          test r14d, r14d
        ,=< 0x100001083    7f05            jg 0x10000108a              ;[1]
        |   0x100001085    e877330000      call 0x100004401            ;[2]
        |      ^- 0x100004401() ; rip
        `-> 0x10000108a    488d357f3a00.   lea rsi, [rip + 0x3a7f]     ; 0x1000
            0x100001091    31ff            xor edi, edi
            0x100001093    e806350000      call sym.imp.setlocale      ;[3]
               ^- 0x10000459e() ; sym.imp.setlocale
            0x100001098    41bd01000000    mov r13d, 1
            0x10000109e    bf01000000      mov edi, 1
            0x1000010a3    e89c340000      call sym.imp.isatty         ;[4]
```

Visual Panels

Press ‘!’ in the Visual mode

```
>  File  Edit  View  Tools  Search  Debug [Analyze] Help             [0x100001060]

   Disassembly                                  > Function          00 0 __mh_execute_he
       0x100001060   push r13                     Program           2 0 radr:__5614542
       0x100001062   push r12                     Calls             4e 0 imp.__assert_rt
       0x100001064   push rbx                     References         4 0 imp.__bzero
       0x100001065   sub rsp, 0x648                                 5a 0 imp.__error
       0x10000106c   mov rbx, rsi
       0x10000106f   mov r14d, edi
       0x100001072   lea rax, [rbp-local_200]             Stack
       0x100001079   mov qword [rbp-local_201],   - offset -    0 1  2 3  4 5  6
       0x100001080   test r14d, r14d              0x00000000   cffa edfe 0700 00
   ,=< 0x100001083   jg 0x10000108a               0x00000010   1300 0000 1807 00
   |   0x100001085   call 0x100004401             0x00000020   1900 0000 4800 00
   `-> 0x10000108a   lea rsi, [rip + 0x3a7f]      0x00000030   524f 0000 0000 00
       0x100001091   xor edi, edi                 0x00000040   0000 0000 0100 00
       0x100001093   call sym.imp.setlocale
       0x100001098   mov r13d, 1                    Registers
       0x10000109e   mov edi, 1                   rax 0x00000000         rbx 0
       0x1000010a3   call sym.imp.isatty          rdx 0x00000000         rsi 0
       0x1000010a8   test eax, eax                 r8 0x00000000          r9 0
   ,=< 0x1000010aa   je 0x10000110d               r11 0x00000000         r12 0
   |   0x1000010ac   mov dword [rip + 0x442a],    r14 0x00000000         r15 0
   |   0x1000010b6   lea rdi, [rip + 0x3a54]      rbp 0x00000000         rflag
```

Web User Interface

Start the webserver with =h

Launch the browser with =H

See /m /p /t and /enyo

Disassembly

ANALYZE COMMENT INFO RENAME WRITE

```
|       0x100001091    xor edi, edi
|       0x100001093    call sym.imp.setlocale
|          ^- 0x10000459e () ; sym.imp.setlocale
|       0x100001098    mov r13d, 1
|       0x10000109e    mov edi, 1
|       0x1000010a3    call sym.imp.isatty
|          ^- 0x100004544 () ; sym.imp.isatty
|       0x1000010a8    test eax, eax
|  ,=< 0x1000010aa    je 0x10000110d
|  |   0x1000010ac    mov dword [rip + 0x442a], 0x50 ; [0x1000054e0:4]
|  |      ^- ; "P" @ 0x1000054e0
|  |   0x1000010b6    lea rdi, [rip + 0x3a54]
|  |      ^- ; 0x100004b11 ; str.COLUMNS
|  |      ^- ; "COLUMNS" @ 0x100004b11
```

SEEK

Bokken

Native Python/Gtk GUI

Binaries for Windows

Runs on OSX/Linux too

Author: Hugo Teso

Questions?

\o.

Thanks For Watching!