radare2

Radare2 - a framework for reverse engineering

Maxime Morin (@Maijin212), Julien Voisin, Jeffrey Crowell (@jeffreycrowell), Anton Kochkov (@akochkov)
October 22, 2015

Hack.lu 10-2015
• 22 y/o french expat @ Luxembourg
• Food, Travel and Languages <3
• I hate Bullshit
• Malware.lu CERT team leader (2days/week) and incident response @ European Commission CSIRC (3days/week)
• User of radare2 (impossibru!)
• I’m creating tests + documentation
• Living in Moscow, Russia
• Reverse Engineering, Languages and Travel
• Reverse engineer, firmware security analyst at SecurityCode Ltd.
• Member of r2 crew
• Living in Paris
• I like to reverse/pwn things
• Mostly bugfixer and warning silencer
• Boston, MA, USA
• Shellphish CTF
• r1 2006, r2 2009
• Multi-(OSes—Archs—Bindings—FileFormats—...)
• 10 tools based on the framework
• Around 149 contributors from various fields
• GSOC + RSOC
• CLI/VisualMode/GUI/WebGUI
• around 350K LOC
installation
installation

- Always use git version!
- Use the provided VM on SSH (radare:radare / root:radare)
- `git clone http://github.com/radare/radare2 && cd radare2 && ./sys/install.sh`
- Use the Windows installer `http://bin.rada.re/radare2.exe`
utilities
utilities

- rax2
- rabin2
- rasm2
- radiff2
- rafind2
- rahash2
- radare2
- r2pm
- rarun2/ragg2/ragg2-cc
utilities

- rax2
- rabin2
- rasm2
- radiff2
- rafind2
- rahash2
- radare2
- r2pm
- rarun2/ragg2/ragg2-cc
utilities: rax2

rax2 — Base converter

$ rax2 10

0xa

$ rax2 33 0x41 0101b

0x21 65 0x5

$ rax2 -s 4142434445

ABCDE

$ rax2 0x5*101b+5

30
utilities

- rax2
- rabin2
- rasm2
- radiff2
- rafind2
- rahash2
- radare2
- r2pm
- rarun2/ragg2/ragg2-cc
$ rabin2 -e

Entrypoints

$ rabin2 -i

Shows imports

$ rabin2 -zz

Shows strings

$ rabin2 -g

Show all possible information
utilities

- rax2
- rabin2
- rasm2
- radiff2
- rafind2
- rahash2
- radare2
- r2pm
- rarun2/ragg2/ragg2-cc
utilities: rasm2

**rasm2** — assembler and disassembler tool

$ rasm2 -a x86 -b 32 'mov eax, 33'

Assemble

$ rasm2 -d 9090

Disassemble

$ rasm2 -L

List supported asm plugins

$ rasm2 -a x86 -b 32 'mov eax, 33' -C

Output in C format
utilities

- rax2
- rabin2
- rasm2
- radiff2
- rafind2
- rahash2
- radare2
- r2pm
- rarun2/ragg2/ragg2-cc
utilities: radiff2

radiff2 — unified binary diffing utility

$ radiff2 original patched

Code diffing

$ radiff2 -C original patched

Code diffing using graphdiff algorithm

$ radiff2 -g main -a x86 -b32 original patched

Graph diff output of given symbol, or between two functions, at given offsets: one for each binary.
utilities: radiff2 — graph example

/bin/true

/bin/false
utilities

- rax2
- rabin2
- rasm2
- radiff2
- rafind2
- rahash2
- radare2
- r2pm
- rarun2/ragg2/ragg2-cc
utilities: rafind2

**rafind2** — Advanced commandline hexadecimal editor

```
$ rafind2 -X -s passwd dump.bin

Search for the string passwd
```
utilities

- rax2
- rabin2
- rasm2
- radiff2
- rafind2
- rahash2
- radare2
- r2pm
- rarun2/ragg2/ragg2-cc
Utilities: rahash2

rahash2 — block based hashing utility

$ rahash2 -a all binary.exe

Display hashes of the whole file with all algos

$ rahash2 -B -b 512 -a md5

Compute md5 per block of 512

$ rahash2 -B -b 512 -a entropy

Compute md5 per block of 512

$ echo -n "admin" | rahash2 -a md5 -s "

Compute md5 of the string admin
utilities

• rax2
• rabin2
• rasm2
• radiff2
• rafind2
• rahash2
• radare2
• r2pm
• rarun2/ragg2/ragg2-cc
radare2 — command line
1 command $\leftrightarrow$ 1 reverse-engineering’notion

Keep in mind that:

1. Every character has a meaning i.e. $(w = \text{write}, \ p = \text{print})$
2. Every command is a succession of character i.e. $\text{pdf} = \ p \leftrightarrow \text{print} \ d \leftrightarrow \text{disassemble} \ f \leftrightarrow \text{function}$
3. Every command is documented with $\text{cmd?}$, i.e. $\text{pdf?}, \ ?, \ ???, \ ???, \ ?$?, \ ?@@
the # command — hashing command

1. Open a file with radare2 radare2 file.exe
2. Get Usage on the command #? Usage: #algo <size>@ addr
3. List of all existing algorithms ##
4. SHA1 #sha1
5. Hashing from the begin #sha1 @ 0
6. with a hash block size corresponding to the size of the file #sha1 $s @ 0\times0

This command is same as rahash2 -a sha1 file.exe
flags

- Flags are used to specify a name for an offset: f?.
- Add a function af+ hand craft a function (requires afb+)
- f. name @ offset set local function label named ‘blah’

- R2 is an block-based hexadecimal editor. Change the blocksize with the ‘b’ command.
the i command — information command

1. Get Usage on the command i?
2. Same as rabin2
3. izj for displaying in json
4. internal commands: ~ ls, {}, ..
Quick Demo
Quick Demo
radare2 — cli main commands

1. r2 -A or r2 then aaa : Analysis
2. s : Seek
3. pdf : Print disassemble function
4. af? : Analyse function
5. ax? : Analyse XREF
6. /? : Search
7. ps? : Print strings
8. C? : Comments
9. w? : Write
radare2 — visual mode
2. p/P : rotate print modes
3. move using arrows/hjkl
4. o : seek to
5. e : r2configurator
6. v : Function list
7. _ : HUD
8. V : ASCII Graph
9. 0-9 : Jump to function
10. u : Go back
radare2 — webui
radare2 webui

r2 -A -c=H filename
radare2 — debugger
1. `radare2 -d`
2. Quickly switch to Visual debugger mode: `Vpp`
3. OllyDBG/IDApro shortcuts friendly
utilities

- rax2
- rabin2
- rasm2
- radiff2
- rafind2
- rahash2
- radare2
- r2pm
- rarun2/ragg2/ragg2-cc
R2PM — radare2 package manager

1. r2pm -s (list all plugins)
2. r2pm -i retdec
debugging

- Native local debug (r2 -d)
- r2 agent (rap:// protocol)
- GDB remote protocol support
- WinDBG remote protocol support
1. Will be shown in Julien and Crowell’s parts
now your turn!

- **Crackmes:** IOLI-Crackme, flare-on 2015 challenges
- **Exploitation:** pwnablekr "bof", simple ret2libc demo, ropasaurus
- **Malware(1/3):** Practical malware analysis samples
- **Malware(2/3):** Any RAT samples see decoder on: https://github.com/kevthehermit/RATDecoders/
- **Malware(3/3):** AVCaesar.lu, MalekalDB
- **Firmware/BIOS/UEFI:** TODO
- **Website:** [http://rada.re/](http://rada.re/)
- **Blog:** [http://radare.today](http://radare.today)
- **Book:** [http://radare.gitbooks.io/radare2book/content](http://radare.gitbooks.io/radare2book/content)
- **Cheatsheet:** [https://github.com/pwntester/cheatsheets/blob/master/radare2.md](https://github.com/pwntester/cheatsheets/blob/master/radare2.md)
Available for a lot of programming languages

Radare2 Bindings —

R2Pipe —

Demo time!
using r2 for exploit
popular tools

- gdb + peda - search memory, dereference stack/registers, debug.
- ida - find xrefs/calls, debug
- ropgadget - search for gadgets
- r2 can do all of this...
getting binary info

- "checksec" - get info: pie, stack canaries, nx
- find strings - find references to calls, etc.
- find writable/executable sections
getting binary info

```plaintext
[0x004048c5]> i~pic
pic     false
[0x004048c5]> i~canary
canary  true
[0x004048c5]> i~nx
nx      true
[0x004048c5]> i~gnu.org
vaddr=0x00417278 paddr=0x00017278 ordinal=369 sz=39 len=38 section=.rodata type=ascii
software/coreutils/
vaddr=0x00418587 paddr=0x00018587 ordinal=422 sz=22 len=21 section=.rodata type=ascii
vaddr=0x004185b8 paddr=0x000185b8 ordinal=424 sz=203 len=202 section=.rodata type=ascii
GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you can redistribute it.
There is NO WARRANTY, to the extent permitted by law.

[0x004048c5]> iS | grep perm=....x
idx=10 vaddr=0x00402168 paddr=0x00002168 sz=26 vsz=26 perm=---r-x name=.init
idx=11 vaddr=0x00402190 paddr=0x00002190 sz=1808 vsz=1808 perm=---r-x name=.plt
idx=12 vaddr=0x004028a0 paddr=0x000028a0 sz=64730 vsz=64730 perm=---r-x name=.text
idx=13 vaddr=0x0041257c paddr=0x0001257c sz=9 vsz=9 perm=---r-x name=.fini
idx=27 vaddr=0x00400000 paddr=0x00000000 sz=113364 vsz=2097152 perm=m-r-x name=phdr0
[0x004048c5]>  
```
• "telescoping" registers
• "telescoping" stack references
• we lose our analysis capabilities on gdb
"telescoping" register

- we can do the same thing with r2
- display references to code/ascii/etc. from registers/stack
- quite useful for dynamic analysis.
- keep flags, symbols, etc.
- drr (registers)
  pxr N @ esp/rsp (stack)
knowing context is useful

- does your register point to a string you control?
- what’s in the stack?
- keep flags, symbols, etc.
- use from within visual mode ‘e dbg.slow = true’
• DeBruijn patterns.
• made famous by metasploit pattern_create.rb
• cyclic patterns, find offset in string.
• Where’s our faked struct/string/etc. being referenced?
• Where did we crash?
• ragg2 -P -r or woD to write
• ragg2 -q or woO to find your offset.
• native, or remote (windows, gdb, …)
• d?
• db addr/flag
• dc[u] debug, continue [until]
• visual mode ”?” c for cursor, b for breakpoints
• starts in the loader, ”dcu entry0” before doing any analysis.
debug 'profiles'

- r2 -de dbg.profile=file.rr2 exec.elf
- set custom arguments, redirect stdin/out to files/sockets
- useful for reproducing environments
context + patterns

- bof from pwnable.kr\(^1\)
- super simple challenge, overflow a buffer
- offset at a certain place must be.
- let’s use rarun2 + references + patterns!

\(^1\) Pwnable kr (2015).
context + patterns

● write your own expl ;)}
• ragg2 isn’t just for generating patterns
• front-end for generating shellcodes
• still up to you to ensure null-free, etc.
• relocatable
• testable (compile directly into elf)
• call arbitrary syscalls easily!
• x86, amd64, arm, windows, mac, linux, ios
• ragg2 file.r -s to show the emitted asm.

```c
execve@syscall(59) # name@syscall(#)

main@global(32) {
    # name (stdcall)
    .var0 = "/bin/sh" # .var(offset)
    execve(.var0, 0, 0); # call!
}
```
code reuse

- return to libc
- rop
- r2 can make this easy
code reuse

- magic shell-spawning gadget
- thanks dragon sector for making this well-known
- exists in amd64 glibc, libruby, and more...
- let’s find it with r2
code reuse

- demo
- r2 -A /path/to/libc
- axt sym.execve
- through xrefs, find it.
- simple demo program on vm does 1 call of your base10 input address
• can’t always use this magic gadget
• rsi must point to something argv-like
• sometimes need to find some odd bespoke gadget!
• r2 can dump gadgets
• regular expression search
• dump to json, write your own tool via r2pipe.
stack layout

- when you "ret"
- ebp is increased by 4, jump to new ebp - 4
- add esp,4
- jmp dword ptr [esp-4]
searching for gadgets

- sequence of instructions followed by "end/stop" gadget
- (arbitrary instructions) - ret/call/jmp/etc...
- finding the right ones is hard, r2 has regexp support
- we can set variable filters.
• super basic rop expl.
• combine finding sections, patterns, rop search.
• r2 makes this easy
searching for gadgets
debugging
gdb protocol

Just run gdbserver somewhere and connect r2 to it:

- r2 -D gdb -d /bin/ls gdb://99.44.23.50:4589
gdb protocol + wine

Winedbg allows to run windows command using the gdbserver too:

- `winedbg --gdb --no-start malware.exe`
- `r2 -a x86 -b 32 -D gdb -d malware.exe gdb://localhost:44840`
r2 allows to connect WinDBG/KD²

For example, to debug windows kernel via the serial port:

- bcdedit /debug on
- bcdedit /dbgsettings serial debugport:1 baudrate:115200

then connect r2:

- r2 -a x86 -b 32 -D wind windbg:///tmp/windbg.pipe

For now, connecting to the QEMU and VirtualBox are tested

²WinDbg in radare2 (2014).
debugging omap bootrom

Just run it in the modified qemu\(^3\)

- \./configure \-target-list=arm-softmmu ; make ; sudo make install
- qemu-system-arm \-M milestone \-m 256 \-L . \-bios bootrom.bin \-mtdblock mbmloader-1.raw \-d in_asm,cpu,exec \-nographic \-s \-S
- r2 \-D gdb \-b arm gdb://localhost:9999

Same approach could be used for any customized hardware

\(^3\)Anton Kochkov (2013). *QEMU patched for loading OMAP bootroms.*
https://github.com/XVilka/qemu.
Winedbg allows to run windows command using the gdbserver too:

- `winedbg -gdb -no-start malware.exe`
- `r2 -a x86 -b 32 -D gdb -d malware.exe gdb://localhost:44840`
firmware analysis
uefi analysis

- Dump the image using flashrom or hardware
- Unpack the image using UEFITool\(^4\)
- Open the selected PE or TE file using r2

old legacy bios analysis

- Load the whole image or unpack it using bios_extract
- Open it using the correct segment and offset
- r2 load the whole BIOS image automatically
- r2 asrock_p4i65g.bin
- >. asrock_p4i65g.r2

---

5 Bios_extract (2015).
the t command — types management

1. Get Usage on the command t?\(^6\)
2. to to load the types from the C header file
3. tl link type to the memory, tf shows it like the pf
4. add j to get the output in the json format

\(^6\)Radare2 types command (2014).
searching guids

1. We need r2pipe (python) for that #?\(^7\)
2. . command to load the pipe script
3. >. search_guids.py
4. this script using the EFI guids list from the snarez’s repo\(^8\)

\(^7\)R2pipe API (2014).
embedded controller - 8051

Let's start from the static analysis

- r2 -a 8051 ite_it8502.rom
- >. ite_it8502.r2
embedded controller - 8051 - esil vm

- r2 -a 8051 ite_it8502.rom
- . ite_it8502.r2
- run `e io.cache=true` to use the cache for write operations
- run `aei` command to init ESIL VM
- run `aeim` command to init ESIL VM stack
- run `aeip` command to start from the current offset
- run `aecu [addr]` to emulate until the [addr] is reached

---

\(^9\) ESIL emulation in radare2 (2014).
Lets start again from the same place

- r2 -a 8051 ite_it8502.rom
- . ite_it8502.r2
- run ‘pae 36’ to show the esil expression of the ‘set_SMBus_frequency’
- run ‘aetr `pae 36`’ to convert the previous esil output to REIL\(^\text{10}\)
- store this to some file and use the ‘openreil’ utility to SMT it

\(^{10}\text{Dmytro Oleksiuk (2015).} \text{https://github.com/Cr4sh/openreil.}\)
not a lot of them I

references


https://github.com/Cr4sh/openreil.


*R2pipe API* (2014).


