Reverse Engineering Embedded Software

an introduction

Using Radare2

Linux.conf.au Auckland 2015

Andrew McDonnell
@pastcompute
retro@andrewmcdonnell.net

Reverse Engineering Embedded Software

an introduction

Using Radare2

Some housekeeping
Please

```
echo 'e cfg.fortunes=false' > ~/.radare2rc
```

https://linux.conf.au/wiki/Tutorials/Reverse_engineering_embedded_software_using_Radare2
https://github.com/pastcompute/lca2015-radare2-tutorial
https://github.com/pastcompute/radare2

Tutorial Outline

- A introduction to Radare2
- Intermission / questions
- MIPS architecture & disassembly
- Extracting embedded device binary images
- Use of BYO binaries encouraged!

https://linux.conf.au/wiki/Tutorials/Reverse_engineering_embedded_software_using_Radare2
https://github.com/pastcompute/lca2015-radare2-tutorial
https://github.com/pastcompute/radare2

Outcomes

- Gain some familiarity with radare2
- Learn a few facts about MIPS architecture
- Discover new tools

https://linux.conf.au/wiki/Tutorials/Reverse_engineering_embedded_software_using_Radare2
https://github.com/pastcompute/lca2015-radare2-tutorial
https://github.com/pastcompute/radare2

Conventions, Examples and Solutions

- Examples / solutions git repository
- Assumed: read wiki and built OK

http://github.com/pastcompute/lca2015-radare2-tutorial

```
# Files in the git repository are shown in bold courier brown

$ examples/gen_radiff2_random_example.sh

# Layout:

examples/       various scripts & source code
data/           pre-existing open source binaries
solutions/      pre-generated binaries and solution output
temp/           generated binaries / data you create while following
```

Reverse Engineering Frameworks

- Features of a RE framework may include:
  - Static & dynamic disassembly
  - Detect high level language features
  - Discover & visualise execution flows
  - Extract data structures
  - Modification & instrumentation, fuzzing

https://linux.conf.au/wiki/Tutorials/Reverse_engineering_embedded_software_using_Radare2
https://github.com/pastcompute/lca2015-radare2-tutorial
https://github.com/pastcompute/radare2

Why?

- Many reasons!
- Interoperability
- Lost source code
- Learning
- Malware Analysis
- (etc)

Radare2

- A reverse engineering & analysis framework
- Extensive CPU & platform coverage
- Scriptable, with extensive API
- Editing of binaries
- Supports debugger integration (gdb)
- Built with a comprehensive library backend
- Web GUI interface (in beta!)

Tutorial : radare2 tools

- Utility toolsuite – unixy philosophy:
  - rax2
  - rabin2
  - rasm2
  - rafind2
  - radiff2

rax2

- Rax2 converts between formats

```
$ rax2 65537
0x10001
$ rax2 0xa1b2c3d4
2712847316
$ rax2 -b 01111010 ; echo
z
$ rax2 -S HelloWorld
48656c6c6f576f726c64
$ rax2 -s 476f6f646279650a
Goodbye
```

rabin2

- Rabin2 dumps information from binaries

```
$ sudo rabin2 -g /vmlinux

(… info … )

$ sudo rabin2 -zz /vmlinux
vaddr=0x00000040 paddr=0x00000040 ordinal=000 sz=120 len=119
section=unknown type=a string=Direct floppy boot is not supported.

(… strings, in detail … )
```

rasm2

- Rasm2 lets you (dis)assemble from shell

```
$ rasm2 -a x86 "mov eax, 0xdeadbeef"
b8efbeadde

$ rasm2 -a x86 "mov eax, 65537"
b801000100

$ rasm2 -a x86 -b32 -D b8efbeadde
0x00000000   5               b8efbeadde  mov eax, 0xdeadbeef

# Beware of github bug 1100 (may be in-progress in trunk):
$ rasm2 -a x86 "mov eax, helloworld"
89f8
# This could be insidious if you use 0xf in a shell script and forget the 0x
```

radiff2

- Radiff2 is a binary code diffing tool
- Example:

```
$ examples/radiff2_random_example.sh

# Creates files, then re-runs: radiff2 temp/file1 temp/file2

Changing 4 bytes
0x00000012 d682aba8 => efbeadde 0x00000012
Changing 4 more bytes
0x00000012 d682aba8 => efbeadde 0x00000012
0x00000212 3679b932 => efbe2dde 0x00000212
Changing 4 more bytes
0x00000012 d682aba8 => efbeadde 0x00000012
0x00000212 3679b932 => efbe2dde 0x00000212
0x00000337 0aa0acbf => efb3adde 0x00000337
```

radiff2

- Is similar to 'cmp', but has extra features
- Example: estimating % similarity of code

```
$ radiff2 -C examples/similar1 examples/similar2

(Files created from examples/similar1.c)

(...)

fcn.00400506  0x400506 |   MATCH  (1.000000) | 0x400506  fcn.00400506
sym.imp.rand  0x400510 |   MATCH  (1.000000) | 0x400510  sym.imp.rand
fcn.00400516  0x400516 |   MATCH  (1.000000) | 0x400516  fcn.00400516
        main  0x400520 | UNMATCH  (0.944444) | 0x400520  main
fcn.004005b0  0x4005b0 |   MATCH  (1.000000) | 0x4005b0  fcn.004005b0
fcn.004005e0  0x4005e0 |   MATCH  (1.000000) | 0x4005e0  fcn.004005e0

(...)
```

rafind2

- Rafind2 is a binary file search & edit tool

```
$ rafind2 -X -s OpenWrt data/openwrt-ar71xx-generic-a02-rb-w300n-
squashfs-sysupgrade.bin
0x20
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000020, 4d49 5053 204f 7065 6e57 7274 204c 696e  MIPS OpenWrt Lin
0x00000030, 7578 2d33 2e31 302e 3439 0000 0000 0000  ux-3.10.49......
0x00000040, 6d00 0080 00e4 3e32 0000 0000 0000 006f  m.....>2.......o
0x00000050, fdff ffa3 b77f 4c34 f7a2 db89 7a6e db3c  ......L4....zn.<
0x00000060, ec7d 4ea1 7178 8f3e 662e 5921 b152       .}N.qx.>f.Y!.R
```

Tutorial: radare2 – ELF example

- Lets look at /sbin/init on your laptop...

```
$ radare2 /sbin/init
[0x000096e9]>

[0x000096e9]> a?
Usage: a[?adfFghoprsx]
 a8 [hexpairs]    ; analyze bytes
 aa               ; analyze all (fcns + bbs)
 ad               ; analyze data trampoline (wip)

(etc)
```

Prompt – Colourised shows current address

All commands have built in help

radare2 – disassembly

- Disassembly (shows x86_64 example)

```
$ radare2 /sbin/init
[0x000096e9]> pd?
[0x000096e9]> pd 32
            ;-- entry0:
            0x000096e9    31ed           xor ebp, ebp
            0x000096eb    4989d1         mov r9, rdx
            0x000096ee    5e             pop rsi
            0x000096ef    4889e2         mov rdx, rsp
            0x000096f2    4883e4f0       and rsp, 0xfffffffffffffff0
            0x000096f6    50             push rax
            0x000096f7    54             push rsp
            0x000096f8    4c8d0571700.   lea r8, [rip+0x27071] ; 0x00010770
            0x000096ff    488d0dfa6f0.   lea rcx, [rip+0x26ffa] ; 0x00010700
            0x00009706    488d3d83f0f.   lea rdi, [rip-0xf7d] ; 0x00018790
            0x0000970d    e80ee7ffff     call sym.imp.__libc_start_main
               0x00007e20(unk, unk) ; sym.imp.__libc_start_main
            0x00009712    f4             hlt
            0x00009713    662e0f1f840.   o16 nop [cs:rax+rax]
            0x0000971d    0f1f00         nop [rax]
```

radare2 – disassembly

- Navigation & inline math; note rax2 similarity

```
$ radare2 /sbin/init
[0x000096e9]> s?
[0x000096e9]> s +100 ; pd 10
(disassembly, etc)

[0x0000974d]> s +0x100 ; pd 0x16
(disassembly, etc)

[0x0000984d]> s sym._init ; pD 32   # ← same without moving: pD 32 @ sym._init
(disassembly, etc)
[0x000076b8]> af ; pdf
(disassembly, etc, with function block shown)
[0x000076b8]> s entry0 +(0x100 + 0xff * 2)
[0x00009be7]>
```

radare2 - assembly

- Edit data, create or modify code

```
$ radare2 malloc://32
[0x00000000]> e asm.arch=6502 ; wx a9018d0002a9058d0102a908aa8e0202 ; pd 10

(disassembly, etc)

[0x00000000]> wt file.6502.bin 16

(save to disk)

$ radare2 -a x86 -b 16 -w dosfile.com
[0x0000:0000]> "wa mov dx,0x20; mov ah,9; int 0x21; mov ah, 0x4c; int 0x21;"
[0x0000:0000]> s 0x20
[0x0000:0020]> w HelloWorld\x0d\x0a$
[0x0000:0020]> q
```

Note quotes around entire command line

Note by default, we need to move insertion point

radare2 – scripting

- Script Solutions for previous example

```
$ radare2 -ax86 -b16 -w temp/dos1.com < examples/dos_asm_example_1.script
$ od -Ax -tx1z temp/dos1.com

$ radare2 -i examples/dos_asm_example_2.script --
$ od -Ax -tx1z temp/dos2.com
```

radare2 – shell

- Shell interaction

```
$ radare2 /sbin/init
[0x000096e9]> pd | wc -l
(print number of lines in outout!)

[0x0000974d]> pd `a_program_calculates_address.sh`
(disassembly, etc)

[0x0000974d]> pd > somefile.txt
(redirection)

[0x0000974d]> !ls -l
(shell command output)

[0x0000974d]> f | less
(browse all flags)

[0x0000974d]> pD `echo 42`
(inline substitution, disassemble 42 bytes)
```

radare2 – shell

- Example: examine partition boot sector code

```
$ sudo radare2 --
[0x00000000]> on /dev/sda
[0x00000000]> px 512
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000000  eb63 9010 8ed0 bc00 b0b8 0000 8ed8 8ec0  .c..............
(… etc ...)
[0x00000000]> pd
        ,=< 0x00000000    eb63         jmp 0x65
        |   0x00000002    90           nop
(… etc ...)
[0x00000000]> s 0x65 ; pD 32
           0x00000065    fa           cli
           0x00000066    90           nop
           0x00000067    90           nop
           0x00000068    f6c280       test dl, -0x80
       ,=< 0x0000006b    7405         je 0x72
       |   0x0000006d    f6c270       test dl, 0x70                    ; 'p'
      ,==< 0x00000070    7402         je 0x74
(… etc ...)
```

radare2 – configuration

- Information, Variables

```
$ radare2 /sbin/init
[0x000096e9]> e??
(… dump of all configuration variables …)

[0x0000974d]> e asm.lineswidth=7 ; pd
(… disassembly, etc. …)

[0x0000974d]> i
file    /sbin/init
type    DYN (Shared object file)
pic true
has_va  true
size    0x40e78
mode    r--
(… etc …)

[0x0000974d]> is
(… list of known symbols in current file …)
```

radare2 - configuration

- ~/.radarerc runs commands on start
- Homework: configure your disassembly layout
- Homework: edit ~/.radarerc to save history

radare2 – configuration

- ~/.radarerc runs commands on start
- Homework: configure your disassembly layout
- Homework: edit ~/.radarerc to save history

radare2 – callgraph

- Find all functions, generate callgraph

(File created from examples/similar1.c)

```
$ radare2 temp/similarfile1
[0x00400586]> aa
[0x00400586]> afl
```

(list all functions detected)
(named if ELF with symbols info available)

```
[0x00400586]> ag > temp/x.dot
[0x00400586]> !xdot temp/x.dot
[0x00400586]> ag main >  temp/xmain.dot
[0x00400586]> !xdot temp/xmain.dot
```

- ***Using a small binary for purpose of demo!***
- ***May be slow for large files***

radare2 – callgraph

- We can also automate the above

```
$ radare2 -nq -c 'aa ; ag main > temp/main.dot' similarfile1
$ xdot temp/main.dot
$ dot -Tpng -otemp/main.png temp/main.dot && eog temp/main.png

# Or altogether
```

examples/gen_main_callgraph.sh

 # Also see: ***solutions/main.dot, solutions/main.png***

radare2 – string data

- Strings

```
$ radare2 /sbin/init
[0x000096e9]> / rlimit
Searching 6 bytes from 0x00000238 to 0x0023e650: 72 6c 69 6d 69 74
# 6 [0x238-0x23e650]
hits: 7
0x00003247 hit0_0 "rlimit"
0x00031a66 hit0_1 "rlimit"
(...)

[0x000096e9]> ps @0x31a66
rlimit
[0x000096e9]> ps @hit0_1
rlimit
[0x000096e9]> s @0x31a66
[0x00031a66]> ps
rlimit
[0x00031a66]> pxl 1
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00003247  726c 696d 6974 0067 6574 7067 6964 0065  rlimit.getpgid.e
```

radare2 – string data

- String Identifiers (flags)

```
[0x00031a66]> aa; fs strings ; f

(… discovered string references …)

[0x00031a66]> f|grep rlimit
0x00031f70 23 str.state_rlimit_serialise
0x00031f50 27 str.state_rlimit_serialise_all
0x00031f30 25 str.state_rlimit_deserialise
0x00031f10 29 str.state_rlimit_deserialise_all
0x00031a7f 8 str.rlimits
0x00031a66 7 str.rlimit

[0x00031a66]> pd 1
           ;-- str.rlimit:
           0x00031a66     .string "rlimit" ; len=7
```

radare2 – specifying files

- Running for tests, experiments

```
$ radare2 malloc://32
[0x00000000]>

$ radare2 -
[0x00000000]>

( same as malloc://512 )

$ radare2 -w some_file.bin
[0x00000000]>

$ radare2 -a mips -m 0x80060000 bootloader.bin
[0x80060000]>
```

Some other features include:

- Text-mode UI **V**
- Recently implemented, WWW UI **W**
- Clipboard (yank buffer) **y?**
- Macros **(?**
- Flags **f?**
- Variables **$?**

Homework Exercise

- Modify a small Arduino binary sans source
- Use: examples/arduino/arduino.cpp.hex as input
- Challenge: play tune always, not occasionally
- Hints (more in **solutions/ARDUINO.MD** file):
  - objcopy can convert hex to binary
  - we have another known Arduino program to look at
  - Useful AVR opcodes: ldi call or brne
  - everything is 8-bit, so integers load 2 or 4 registers in a row

Intermission – Questions?

Embedded Architectures

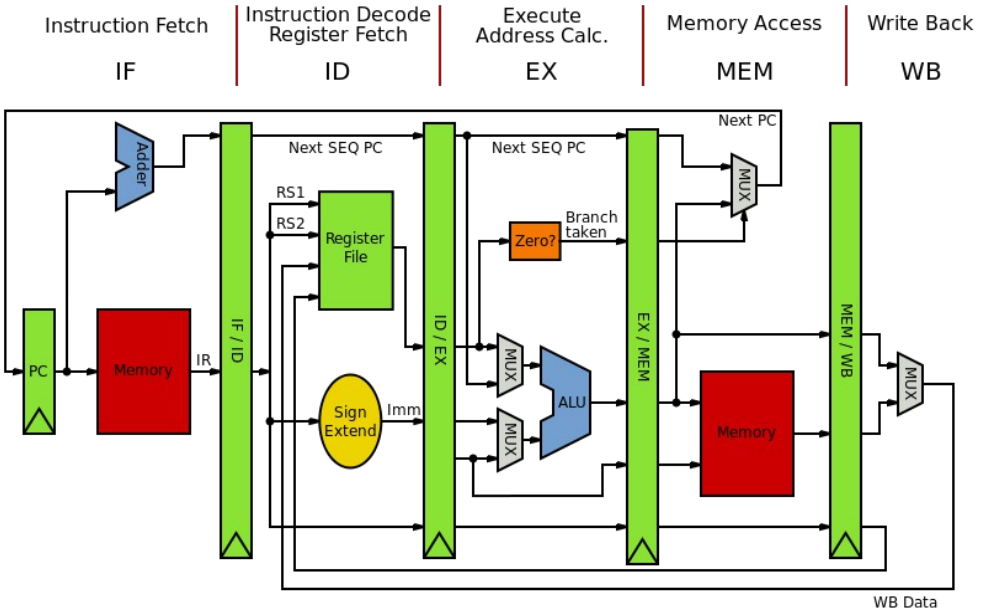
- Traditional microcontrollers
  - very low ram, clock speed, 8,16,32 bits
  - 8051, ATMEGA, TI LPC, ARM, PIC, Arduino platform, etc
- Embedded-linux capable systems
  - ARM : iPhone, Raspberry PI, etc
  - Intel : Atom, Galileo, etc
  - MIPS : Carambola2, WRTnode, routers / webcams

- Trivia: name three past MIPS machines

32-bit MIPS processors in brief

- RISC; Pipeline architecture
- Pipeline: concurrent stage execution

(requires all instructions to be 32-bit)



Attribution: user:InductiveLoad / Wikimedia Commons / Public Domain
https://commons.wikimedia.org/wiki/File:MIPS_Architecture_%28Pipelined%29.svg

MIPS disassembly

- Register include: a0-a3, t0-t9
- Stores / add: DEST ← SOURCE(S)
-

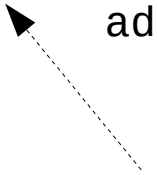
```
$ radare2 temp/mipshello
[0x00400790]> fs symbols ; f
[0x00400790]> s sym.main ; af ; pdf
(… …)
[0x00400730]>
```

MIPS Quirks

- Pipeline : delayed branches
- Instruction following a jump always executes!

  (One of the biggest 'gotchas' reading MIPS assembly)

```
[0x00400730]> s fcn.004006A0 ; pdf
/ (fcn) fcn.004006A0 16
|           ; CALL XREF from 0x0040073c (sym.main)
|           0x00400710    3c0f0041     lui t7, 0x41
|           0x00400714    8df90a58     lw t9, 0xa58(t7)
|           0x00400718    03200008     jr t9
\           0x0040071c    25f80a58     addiu t8, t7, 0xa58
```



This instruction is executed as well!

Other MIPS factors

- Instruction set extensions (ASE)
  - MIPS16e ASE (in use by OpenWRT) is not yet supported
- Cache configuration
- Alignment constraints
- Interrupt handling

MIPS Memory Map

(kernel mode only, translated through MMU, typ. modules)

0xC0000000

(uncached mapped to low 512MB e.g. MMIO, direct flash read)

0xA0000000

(cached mapped to low 512MB physical, typ. kernel)

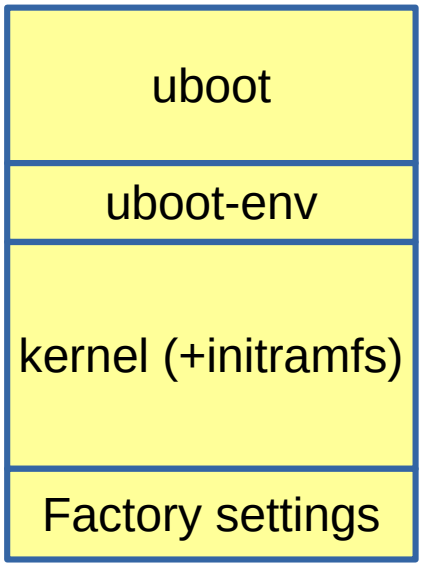
0x80000000

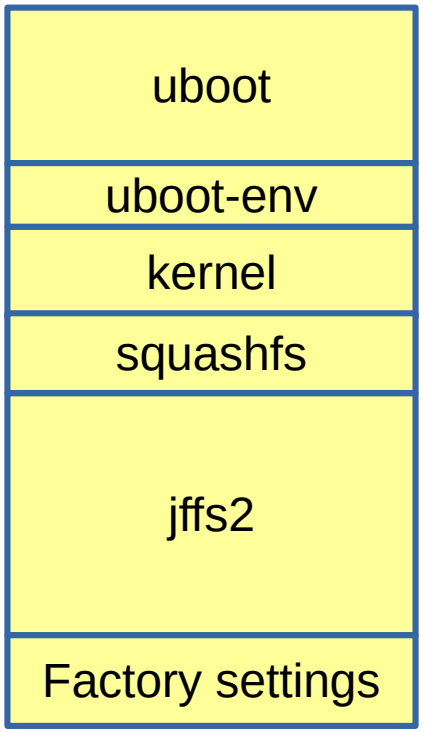
(user mode, translated through MMU)

0x00000000

Embedded MIPS, flash layout

- Typically a SOC
- Various core types – 4k, 24kc, 34kc
- Typical: uboot, kernel, jffs2 &/or squashfs





Finding something to RE? binwalk

- Use openwrt unsquashfs if binwalk outdated

```
$ binwalk -e data/openwrt-ar71xx-generic-a02-rb-w300n-squashfs-sysupgrade.bin

DECIMAL    HEX        DESCRIPTION
--------------------------------------------------------------------------------
0          0x0        uImage header, header size: 64 bytes, header CRC: 0x52196D55, created: Thu Oct  2
16:28:59 2014, image size: 1106985 bytes, Data Address: 0x80060000, Entry Point: 0x80060000, data CRC:
0x5869C399, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name: "MIPS
OpenWrt Linux-3.10.49"
64         0x40       LZMA compressed data, properties: 0x6D, dictionary size: 8388608 bytes, uncompressed
size: 3292900 bytes
1107049    0x10E469   Squashfs filesystem, little endian, version 4.0, compression:  size: 2148770 bytes,  950
inodes, blocksize: 262144 bytes, created: Thu Oct  2 16:28:59 2014
```

Firmware : bootloaders

- U-boot – open source
  - but usually modified to support the hardware
- Other bootloaders
  - how to load Linux?
  - reverse engineering challenge!

Firmware Extraction Approaches

- Find the serial port and attach a USB adapter
- If gadget has network, find means to download
  - may need creativity: `dd | openssl` may work if all else fails
- Or upload: cross compile netcat
- Worst case: copying/pasting hex in serial!

Today was only 1.5 hours :-)

- Too brief to teach in-depth reverse engineering
  - Understanding common patterns can take experience
- Radare2 has very many commands & options
  - Reverse engineering can be very problem-specific
  - Some options are very esoteric
- Other tools often required in concert
- Real world use case hard to demonstrate here:
  - closed source blobs: copyright / license issues?

Where to from here?

- Radare2 has many features:
  - debugger integration
  - infosec features
  - language bindings: python, perl, vala, lua, etc.
- Study / try real world blog articles. Examples:
  - Patch firmware in a cheap IP camera. Discount department stores sell rebadged cams running MIPS for $50 AUD
  - The following done with IDA; see if you can do it using radare2: http://www.devttys0.com/2013/10/reverse-engineering-a-d-link-backdoor/
  - Analyse a virus sample. But take precautions! (use a VM)

Where to from here?

- Radare2 is under active development
- Help out!
  - hack the code & submit patches on github!
  - online help & regression tests always need improving
  - find and report analyser, web UI & callgraph bugs

http://radare.org

https://github.com/radare/radare2

Further Reading

- “See Mips Run” (search oreilly.com)
- The radare2 book online
- Reverse engineering for beginners: http://beginners.re/

Thanks for Coming :-)

Andrew McDonnell
@pastcompute
retro@andrewmcdonnell.net
http://au.linkedin.com/in/amcdonnell/

Acknowledgements to James Brown for valuable feedback & assistance
@radare @jvoisin @deeso @xvilka on radare2 Github for their patience :-)
And not least my family for putting up with my late nights on this project

This material is licensed under the Creative Commons Attribution-ShareAlike 3.0 Australia License.
To view a copy of this license, visit https://creativecommons.org/licenses/by-sa/3.0/au/