AN INTRODUCTION TO RADARE2

{*alu*|*l*0*gic*}

April 7, 2011

Outline

Overview

Outline

Outline

Outline

Outline

What is radare2?

- Debugger
- Disassembler
- Everything else you'd expect from a reverse engineering toolchain
  - (Including way too many features to be usable for new users)
- Work in progress
- Open Source

Features

- Extract information from binaries
- Hash binaries
- Analyze opcodes
- Relocatable code compiler
- Shellcode helper
- Binary diffing
- Commandline (dis)assembler
- Base conversion

The stuff at the end of every talk

r2

- Main binary
- Interface for all subsystems
- Provides
  - Visual debugger interface
  - Visual disassembler
  - Shell

Extract information from binaries I

rabin2

- Imports
- Strings
- And many more

Extract information from binaries II

rabin2

```
$ rabin2 -z a.out
[strings]
address=0x08048924 offset=0x00000924 ordinal=000 size=39
     section=.rodata string=I'm not accepting any
    arguments, sorry.
address=0x0804894c offset=0x0000094c ordinal=001 size=6
    section=.rodata string=FIXME!
address=0x08048954 offset=0x00000954 ordinal=002 size=21
     section=.rodata string=Looks like it's ok :)
address=0x0804896a offset=0x0000096a ordinal=003 size=17
     section=.rodata string=Try readelf -h %s

4 strings
```

Analyze opcodes

ranal2

- Supports different architectures
- Usually invoked from within r2

Relocatable compiler I

rarc2

- Relocatable compiler
- Uses C-like syntax

Relocatable compiler II

rarc2

```
$ echo 'main@global(,64){printf("hello world\n");}' \
  | rarc2 -s > hello.S
$ gcc hello.S
$ ./a.out
hello world
```

Binary diffing I

radiff2

```
if(a == b)
  printf("Nope.\n");
else
  printf("Everything's ok :)\n");

if(a != b)
  printf("Nope.\n");
else
  printf("Everything's ok :)\n");
```

Binary diffing II

radiff2

```
$ radiff2 1 2
0x0000040d 75 => 74 0x0000040d
0x000010ab 31 => 32 0x000010ab
```

Shellcode helper I

rasc2

- Shellcode helper
- Has a list of 50 hardcoded shellcodes
- Different output formats

Shellcode helper II

rasc2

```
$ rasc2 -i x86.bsd.suidsh -c
unsigned char shellcode[] = {
  0x31, 0xc0, 0x50, 0x50, 0xb0, 0x17, 0xcd, 0x80,
  0x31, 0xc0, 0x50, 0x68, 0x2f, 0x2f, 0x73, 0x68,
  0x68, 0x2f, 0x62, 0x69, 0x6e, 0x89, 0xe3, 0x50,
  0x54, 0x53, 0x50, 0xb0, 0x3b, 0xcd, 0x80,
};
```

Commandline (dis)assembler I

rasm2

- Supports different architectures
- Supports intel and AT&T syntax

Commandline (dis)assembler II

rasm2

```
$ rasm2 -a x86.nasm -
mov eax, [esp+0x1c]
8b44241c
```

Commandline (dis)assembler III

rasm2

```
$ rasc2 -i x86.bsd.suidsh -x | rasm2 -d -
xor eax, eax
push eax
push eax
mov al, 0x17
int 0x80
xor eax, eax
push eax
push dword 0x68732f2f
push dword 0x6e69622f
mov ebx, esp
push eax
push esp
push ebx
push eax
mov al, 0x3b
int 0x80
```

False disassembly I

```
xor eax, eax
jnz no_magic
jz no_magic+1
no_magic:
mov eax, 0xc3c948
```

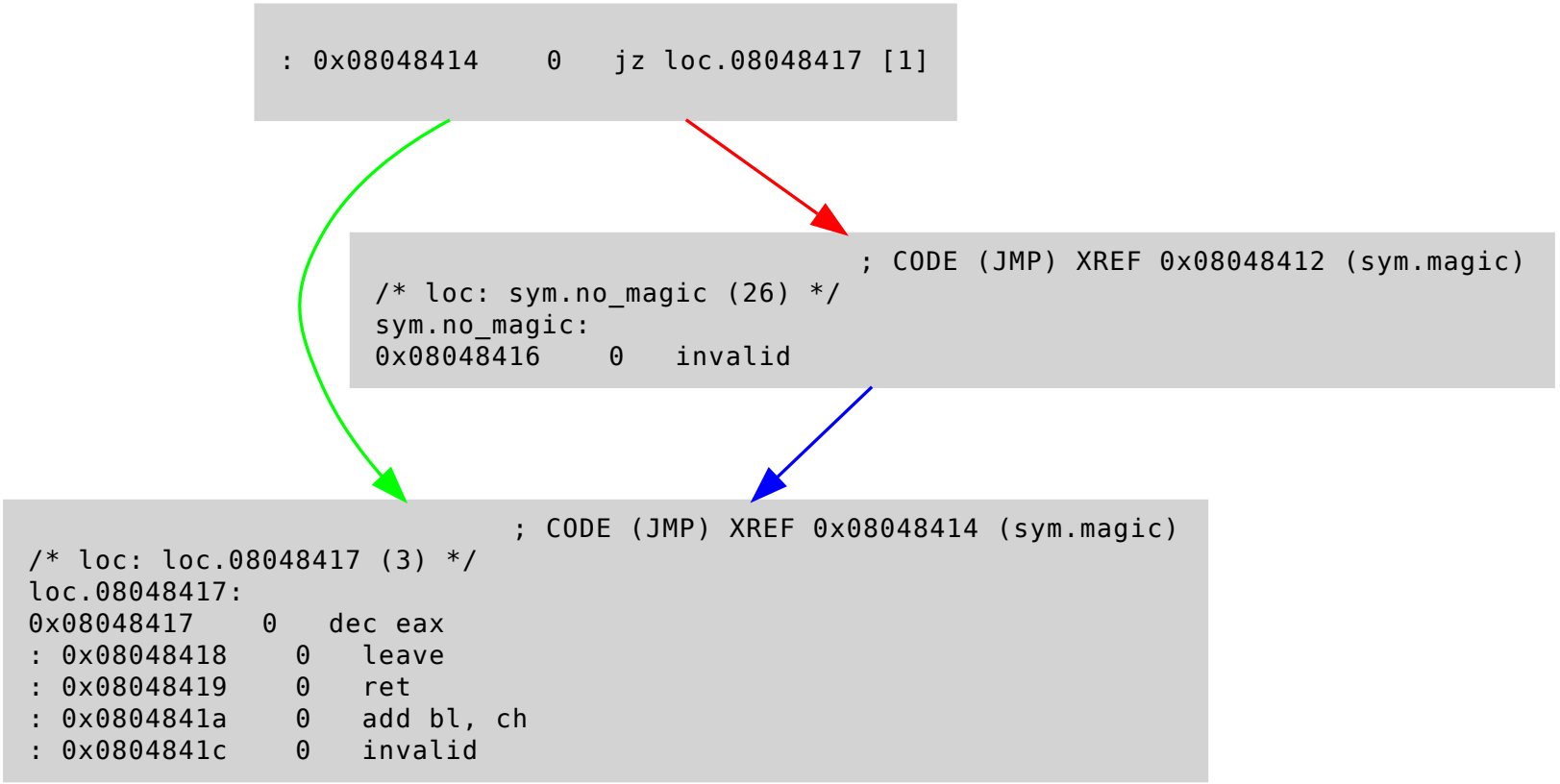
False disassembly II

```
      0x08048410    28                  31c0   xor eax, eax
  .=< 0x08048412    28                  7502   jnz sym.no_magic [2]
.==< 0x08048414    28                  7401   jz 0x8048417 [3]
`--> 0x08048416    28    *[  sym.no_magic]   mov eax, 0xc3c948
```

Two possible solutions:

1. Change Byte at 0x08048416 to 0x90 (nop)
2. Use radare2's codegraph ;)

False disassembly III



Dynamic call (or jump) targets I

```
mov eax, [esp+0x1c]
xor eax, 0x539
sub eax, 0xc8
ror eax, 0x3
call eax
```

Solutions:

- Understand how the targed is being computed
- Don't care and set eax by hand[1]

[1] No reallife use, I'd say...

Dynamic call (or jump) targets II

The one time solution

1. Find correct target
2. Set BP @call eax (db <offset>)
3. Run (dc)
4. Set eax (dr eax=<offset>)
5. Continue (dc)

Breakpoint detection I

Software Breakpoints

- Replaces instruction with int3 (0xCC)

```
if ((*(volatile unsigned *)((unsigned)foo) & 0xFF) == 0xCC)
  /* Some anti debugging foo */
```

Breakpoint detection II

Hardware Breakpoints I

- Use the debug registers
  - Max 4 hardware breakpoints
  - Direct access needs ring0 privileges

Breakpoint detection II

Hardware Breakpoints II

```
mov eax, dr0
cmp eax, 0
jnz bad_guy
```

- Will cause a SIGSEGV for ring3 users
- ptrace() and fork() to the rescue!

Breakpoint detection II

Hardware Breakpoints III

```
#define DR_OFFSET(dr) ((int)((((struct user *)0)->
    u_debugreg) + (dr)))
childpid = fork();
if(childpid == 0) {
  ppid = getppid();
  ptrace(PTRACE_ATTACH, ppid, 0, 0);
  wait(&status);
  for(i = 0; i <= 3; i++) {
    dr = ptrace(PTRACE_PEEKUSER, ppid, DR_OFFSET(i), 0);
    if(dr != 0) {
      ptrace(PTRACE_KILL, ppid, 0);
      kill(ppid, SIGKILL);
      return 1;
    }
  }
  ptrace(PTRACE_DETACH, ppid, 0);
  return 12; } else {
```

Detect debuggers...

... which use ptrace

```
if (ptrace(PTRACE_TRACEME, 0, 1, 0) < 0) {
  printf("I don't like being traced!\n");
  return 0;
}
```

- Not as scary as the last example, huh?

Corrupted headers I

- binutils don't like it
- r2 has a lot of language bindings ;)[2]

[2] i.e. http://radare.org/y/?p=examples&f=vala

Corrupted headers

```
$ echo wx FF @ 0x21 | r2 -nw a.out
$ objdump -d a.out
objdump: a.out: File truncated
```

Targets

- What does ./bins/debugme/debugme?
- Write a keygen for ./bins/keygenme/keygenme
- Fix ./bins/fixme/fixme

I'll trade Sourcecode for solutions ;)

Rules I

debugme

- Use r2 ;)

Rules II

keygenme

- No patching![3]

[3] As in: Don't change the checks for good/false msgs ;)

Rules III

fixme

- Only allowed patch offsets are[4]
  - 0x20
  - 0x21

[4] On x86_64 it's 0x28/0x29

Questions?

Are there any?

Now is the time to ask your questions.
Don't have any? Good, then go and crunch some asm!

More info

Aka where to get it? And where is the friggin' doc?

Point your browsers or telnets to:

- http://radare.org
- http://is.gd/jNEphA (ML)
- #radare on freenode