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

maxime morin

- 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

anton kochkov

- Living in Moscow, Russia
- Reverse Engineering, Languages and Travel
- Reverse engineer, firmware security analyst at SecurityCode Ltd.
- Member of r2 crew

julien voisin

- Living in Paris
- I like to reverse/pwn things
- Mostly bugfixer and warning silencer

jeffrey crowell

- Boston, MA, USA
- Shellphish CTF

generality on radare2 framework

- 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

utilities: rabin2

rabin2 — Binary program info extractor

```
$ 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
```

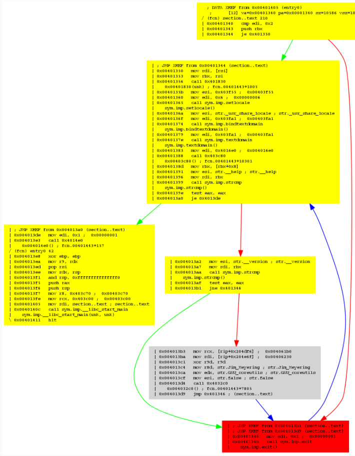
Code diffing using graphdiff algorithm

```
$ 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



```
   ; DATA XREF from 0x00401405 (entry0)
    ;      [12] va=0x00401340 pa=0x00001340 sz=10586 vsz=10586 rwx=-r-x .text
/ (fcn) section..text 210
| 0x00401340   cmp edi, 0x2
| 0x00401343   push rbx
| 0x00401344   je 0x401350
```

```
| ; JMP XREF from 0x00401344 (section..text)
| 0x00401350   mov rdi, [rsi]
| 0x00401353   mov rbx, rsi
| 0x00401356   call 0x401830
|    0x00401830(unk) ; fcn.00401443+1005
| 0x0040135b   mov esi, 0x403f55 ;  0x00403f55
| 0x00401360   mov edi, 0x6 ;  0x00000006
| 0x00401365   call sym.imp.setlocale
|    sym.imp.setlocale()
| 0x0040136a   mov esi, str._usr_share_locale ; str._usr_share_locale
| 0x0040136f   mov edi, 0x403fa1 ;  0x00403fa1
| 0x00401374   call sym.imp.bindtextdomain
|    sym.imp.bindtextdomain()
| 0x00401379   mov edi, 0x403fa1 ;  0x00403fa1
| 0x0040137e   call sym.imp.textdomain
|    sym.imp.textdomain()
| 0x00401383   mov edi, 0x4016e0 ;  0x004016e0
| 0x00401388   call 0x403c80
|    0x00403c80() ; fcn.00401443+10301
| 0x0040138d   mov rbx, [rbx+0x8]
| 0x00401391   mov esi, str.__help ; str.__help
| 0x00401396   mov rdi, rbx
| 0x00401399   call sym.imp.strcmp
|    sym.imp.strcmp()
| 0x0040139e   test eax, eax
| 0x004013a0   je 0x4013de
```

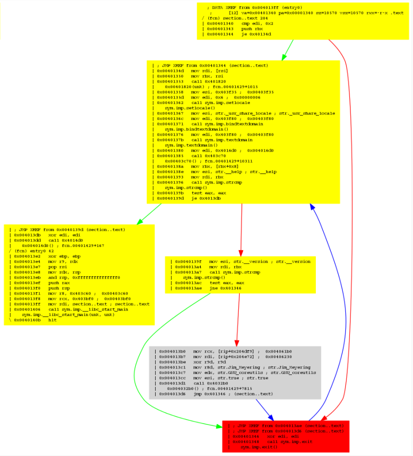
```
| ; JMP XREF from 0x004013a0 (section..text)
| 0x004013de   mov edi, 0x1 ;  0x00000001
| 0x004013e3   call 0x4014e0
|    0x004014e0() ; fcn.00401443+157
/ (fcn) entry0 42
| 0x004013e8   xor ebp, ebp
| 0x004013ea   mov r9, rdx
| 0x004013ed   pop rsi
| 0x004013ee   mov rdx, rsp
| 0x004013f1   and rsp, 0xfffffffffffffff0
| 0x004013f5   push rax
| 0x004013f6   push rsp
| 0x004013f7   mov r8, 0x403c70 ;  0x00403c70
| 0x004013fe   mov rcx, 0x403c00 ;  0x00403c00
| 0x00401405   mov rdi, section..text ; section..text
| 0x0040140c   call sym.imp.__libc_start_main
|    sym.imp.__libc_start_main(unk, unk)
\ 0x00401411   hlt
```

```
| 0x004013a2   mov esi, str.__version ; str.__version
| 0x004013a7   mov rdi, rbx
| 0x004013aa   call sym.imp.strcmp
|    sym.imp.strcmp()
| 0x004013af   test eax, eax
| 0x004013b1   jne 0x401346
```

```
| 0x004013b3   mov rcx, [rip+0x204df6] ;  0x004061b0
| 0x004013ba   mov rdi, [rip+0x204e6f] ;  0x00406230
| 0x004013c1   xor r9d, r9d
| 0x004013c4   mov r8d, str.Jim_Meyering ; str.Jim_Meyering
| 0x004013ca   mov edx, str.GNU_coreutils ; str.GNU_coreutils
| 0x004013cf   mov esi, str.false ; str.false
| 0x004013d4   call 0x4032c0
|    0x004032c0() ; fcn.00401443+7805
| 0x004013d9   jmp 0x401346 ; (section..text)
```

```
| ; JMP XREF from 0x004013b1 (section..text)
| ; JMP XREF from 0x004013d9 (section..text)
| 0x00401346   mov edi, 0x1 ;  0x00000001
| 0x0040134b   call sym.imp.exit
|    sym.imp.exit()
```

/bin/false



```
   ; DATA XREF from 0x004013ff (entry0)
    ;      [12] va=0x00401340 pa=0x00001340 sz=10570 vsz=10570 rwx=-r-x .text
/ (fcn) section..text 204
| 0x00401340   cmp edi, 0x2
| 0x00401343   push rbx
| 0x00401344   je 0x40134d
```

```
| ; JMP XREF from 0x00401344 (section..text)
| 0x0040134d   mov rdi, [rsi]
| 0x00401350   mov rbx, rsi
| 0x00401353   call 0x401820
|    0x00401820(unk) ; fcn.00401429+1015
| 0x00401358   mov esi, 0x403f35 ;  0x00403f35
| 0x0040135d   mov edi, 0x6 ;  0x00000006
| 0x00401362   call sym.imp.setlocale
|    sym.imp.setlocale()
| 0x00401367   mov esi, str._usr_share_locale ; str._usr_share_locale
| 0x0040136c   mov edi, 0x403f80 ;  0x00403f80
| 0x00401371   call sym.imp.bindtextdomain
|    sym.imp.bindtextdomain()
| 0x00401376   mov edi, 0x403f80 ;  0x00403f80
| 0x0040137b   call sym.imp.textdomain
|    sym.imp.textdomain()
| 0x00401380   mov edi, 0x4016d0 ;  0x004016d0
| 0x00401385   call 0x403c70
|    0x00403c70() ; fcn.00401429+10311
| 0x0040138a   mov rbx, [rbx+0x8]
| 0x0040138e   mov esi, str.__help ; str.__help
| 0x00401393   mov rdi, rbx
| 0x00401396   call sym.imp.strcmp
|    sym.imp.strcmp()
| 0x0040139b   test eax, eax
| 0x0040139d   je 0x4013db
```

```
| ; JMP XREF from 0x0040139d (section..text)
| 0x004013db   xor edi, edi
| 0x004013dd   call 0x4014d0
|    0x004014d0() ; fcn.00401429+167
/ (fcn) entry0 42
| 0x004013e2   xor ebp, ebp
| 0x004013e4   mov r9, rdx
| 0x004013e7   pop rsi
| 0x004013e8   mov rdx, rsp
| 0x004013eb   and rsp, 0xfffffffffffffff0
| 0x004013ef   push rax
| 0x004013f0   push rsp
| 0x004013f1   mov r8, 0x403c60 ;  0x00403c60
| 0x004013f8   mov rcx, 0x403bf0 ;  0x00403bf0
| 0x004013ff   mov rdi, section..text ; section..text
| 0x00401406   call sym.imp.__libc_start_main
|    sym.imp.__libc_start_main(unk, unk)
\ 0x0040140b   hlt
```

```
| 0x0040139f   mov esi, str.__version ; str.__version
| 0x004013a4   mov rdi, rbx
| 0x004013a7   call sym.imp.strcmp
|    sym.imp.strcmp()
| 0x004013ac   test eax, eax
| 0x004013ae   jne 0x401346
```

```
| 0x004013b0   mov rcx, [rip+0x204df9] ;  0x004061b0
| 0x004013b7   mov rdi, [rip+0x204e72] ;  0x00406230
| 0x004013be   xor r9d, r9d
| 0x004013c1   mov r8d, str.Jim_Meyering ; str.Jim_Meyering
| 0x004013c7   mov edx, str.GNU_coreutils ; str.GNU_coreutils
| 0x004013cc   mov esi, str.true ; str.true
| 0x004013d1   call 0x4032b0
|    0x004032b0() ; fcn.00401429+7815
| 0x004013d6   jmp 0x401346 ; (section..text)
```

```
| ; JMP XREF from 0x004013ae (section..text)
| ; JMP XREF from 0x004013d6 (section..text)
| 0x00401346   xor edi, edi
| 0x00401348   call sym.imp.exit
|    sym.imp.exit()
```

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 <—>1 reverse-engineering'notion

Keep in mind that:

1. Every character has a meaning i.e (w = write, p = print)
2. Every command is a succession of character i.e pdf = p <->print d <->disassemble f <->function
3. Every command is documented with **cmd?**, i.e 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 @ 0x0

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, {}, ..

radare2 — ‘major’ command example: pf

Quick Demo

radare2 - types command example

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

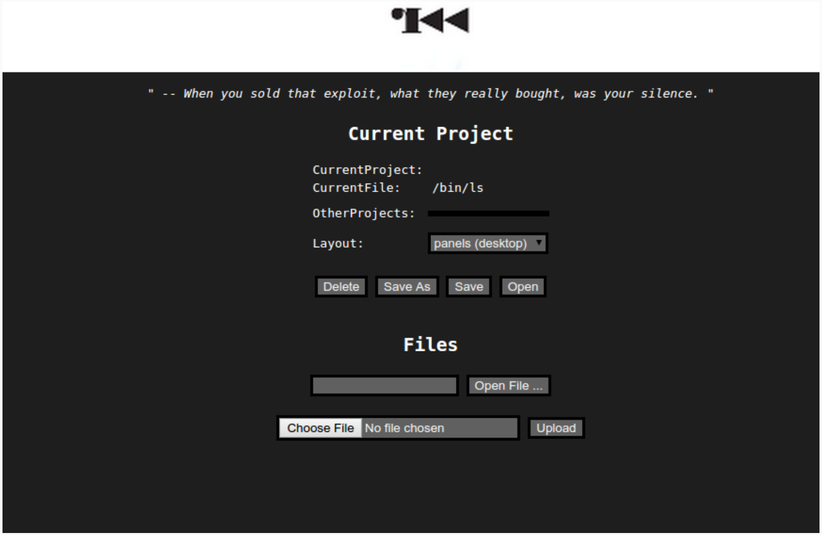
radare2 — visual mode main commands

1. V? : Visual help
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



" -- When you sold that exploit, what they really bought, was your silence. "

Current Project

```
CurrentProject:
CurrentFile:     /bin/ls
OtherProjects:
Layout:          panels (desktop)
```

Files

radare2 — debugger

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

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

rarun2 && ragg2 && ragg2-cc

1. Will be shown in Julien and Crowell’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

documentation

- Website: http://rada.re/
- Blog: http://radare.today
- Book: http://radare.gitbooks.io/radare2book/content/

scripting capabilities

Available for a lot of programming languages

Radare2 Bindings —

R2Pipe —

Demo time !

using r2 for exploit

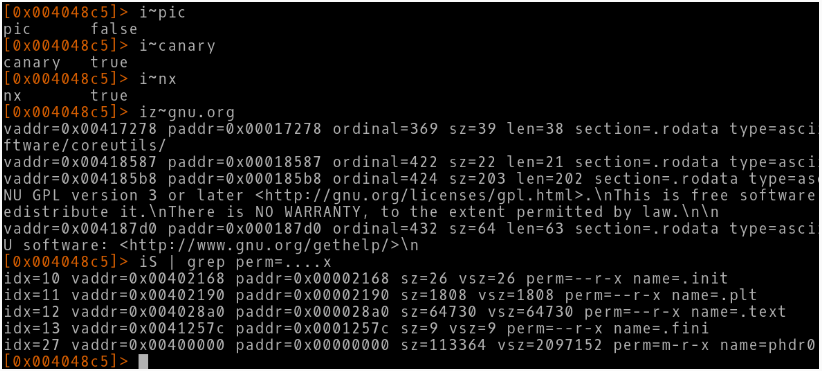
popular tools

- 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

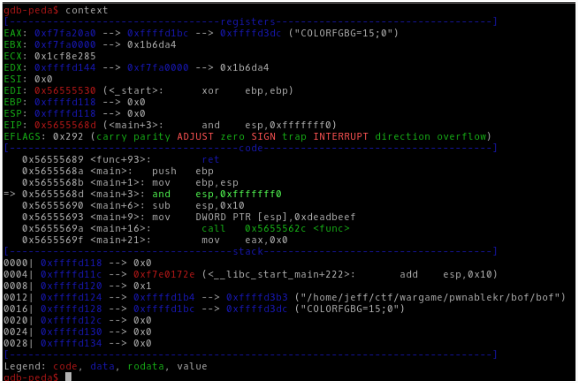
getting binary info



```
[0x004048c5]> i~pic
pic      false
[0x004048c5]> i~canary
canary   true
[0x004048c5]> i~nx
nx       true
[0x004048c5]> iz~gnu.org
vaddr=0x00417278 paddr=0x00017278 ordinal=369 sz=39 len=38 section=.rodata type=asci
ftware/coreutils/
vaddr=0x00418587 paddr=0x00018587 ordinal=422 sz=22 len=21 section=.rodata type=asci
vaddr=0x004185b8 paddr=0x000185b8 ordinal=424 sz=203 len=202 section=.rodata type=as
NU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.\nThis is free software
edistribute it.\nThere is NO WARRANTY, to the extent permitted by law.\n\n
vaddr=0x004187d0 paddr=0x000187d0 ordinal=432 sz=64 len=63 section=.rodata type=asci
U software: <http://www.gnu.org/gethelp/>\n
[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" register

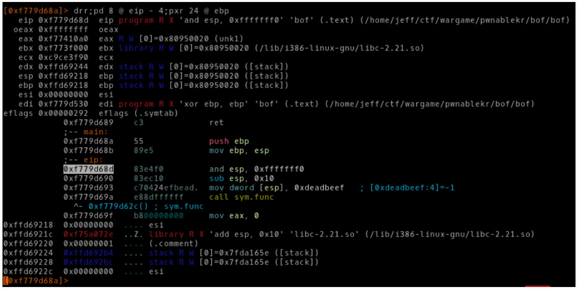
- "telescoping" registers
- "telescoping" stack references
- we lose our analysis capabilities on gdb



```
gdb-peda$ context
[----------------------------------registers-----------------------------------]
EAX: 0xf7fa20a0 --> 0xffffd1bc --> 0xffffd3dc ("COLORFGBG=15;0")
EBX: 0xf7fa0000 --> 0x1b6da4
ECX: 0x1cf8e285
EDX: 0xffffd144 --> 0xf7fa0000 --> 0x1b6da4
ESI: 0x0
EDI: 0x56555530 (<_start>:      xor    ebp,ebp)
EBP: 0xffffd118 --> 0x0
ESP: 0xffffd118 --> 0x0
EIP: 0x5655568d (<main+3>:      and    esp,0xfffffff0)
EFLAGS: 0x292 (carry parity ADJUST zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x56555689 <func+93>:        ret
   0x5655568a <main>:   push   ebp
   0x5655568b <main+1>: mov    ebp,esp
=> 0x5655568d <main+3>: and    esp,0xfffffff0
   0x56555690 <main+6>: sub    esp,0x10
   0x56555693 <main+9>: mov    DWORD PTR [esp],0xdeadbeef
   0x5655569a <main+16>:        call   0x5655562c <func>
   0x5655569f <main+21>:        mov    eax,0x0
[------------------------------------stack-------------------------------------]
0000| 0xffffd118 --> 0x0
0004| 0xffffd11c --> 0xf7e0172e (<__libc_start_main+222>:       add    esp,0x10)
0008| 0xffffd120 --> 0x1
0012| 0xffffd124 --> 0xffffd1b4 --> 0xffffd3b3 ("/home/jeff/ctf/wargame/pwnablekr/bof/bof")
0016| 0xffffd128 --> 0xffffd1bc --> 0xffffd3dc ("COLORFGBG=15;0")
0020| 0xffffd12c --> 0x0
0024| 0xffffd130 --> 0x0
0028| 0xffffd134 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
gdb-peda$ 
```

"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)



```
[0xf779d68a]> drr;pd 8 @ eip - 4;pxr 24 @ ebp
   eip 0xf779d68d  eip program R X 'and esp, 0xfffffff0' 'bof' (.text) (/home/jeff/ctf/wargame/pwnablekr/bof/bof)
  oeax 0xffffffff  oeax
   eax 0xf77410a0  eax R W [0]=0x80950020 (unk1)
   ebx 0xf773f000  ebx library R W [0]=0x80950020 (/lib/i386-linux-gnu/libc-2.21.so)
   ecx 0xc9ce3f90  ecx
   edx 0xffd69244  edx stack R W [0]=0x80950020 ([stack])
   esp 0xffd69218  ebp stack R W [0]=0x80950020 ([stack])
   ebp 0xffd69218  ebp stack R W [0]=0x80950020 ([stack])
   esi 0x00000000  esi
   edi 0xf779d530  edi program R X 'xor ebp, ebp' 'bof' (.text) (/home/jeff/ctf/wargame/pwnablekr/bof/bof)
eflags 0x00000292  eflags (.symtab)
            0xf779d689    c3             ret
            ;-- main:
            0xf779d68a    55             push ebp
            0xf779d68b    89e5           mov ebp, esp
            ;-- eip:
            0xf779d68d    83e4f0         and esp, 0xfffffff0
            0xf779d690    83ec10         sub esp, 0x10
            0xf779d693    c70424efbead.  mov dword [esp], 0xdeadbeef   ; [0xdeadbeef:4]=-1
            0xf779d69a    e88dffffff     call sym.func
              ^- 0xf779d62c() ; sym.func
            0xf779d69f    b800000000     mov eax, 0
0xffd69218  0x00000000  .... esi
0xffd6921c  0xf75a072e  ..Z. library R X 'add esp, 0x10' 'libc-2.21.so' (/lib/i386-linux-gnu/libc-2.21.so)
0xffd69220  0x00000001  .... (.comment)
0xffd69224  0xffd692b4  .... stack R W [0]=0x7fda165e ([stack])
0xffd69228  0xffd692bc  .... stack R W [0]=0x7fda165e ([stack])
0xffd6922c  0x00000000  .... esi
[0xf779d68a]>
```

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‘

pattern generate

- 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.

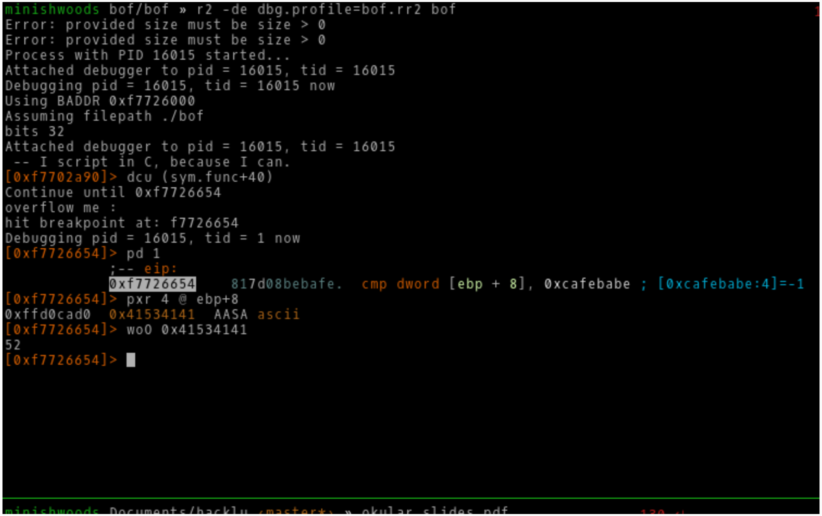
debug 'profiles'

- 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
- super simple challenge, overflow a buffer
- offset at a certain place must be.
- let's use rarun2 + references + patterns!

context + patterns



```
minishwoods bof/bof » r2 -de dbg.profile=bof.rr2 bof
Error: provided size must be size > 0
Error: provided size must be size > 0
Process with PID 16015 started...
Attached debugger to pid = 16015, tid = 16015
Debugging pid = 16015, tid = 16015 now
Using BADDR 0xf7726000
Assuming filepath ./bof
bits 32
Attached debugger to pid = 16015, tid = 16015
 -- I script in C, because I can.
[0xf7702a90]> dcu (sym.func+40)
Continue until 0xf7726654
overflow me :
hit breakpoint at: f7726654
Debugging pid = 16015, tid = 1 now
[0xf7726654]> pd 1
            ;-- eip:
            0xf7726654    817d08bebafe.  cmp dword [ebp + 8], 0xcafebabe ; [0xcafebabe:4]=-1
[0xf7726654]> pxr 4 @ ebp+8
0xffd0cad0  0x41534141  AASA ascii
[0xf7726654]> wo0 0x41534141
52
[0xf7726654]> █
```

- write your own expl ;)

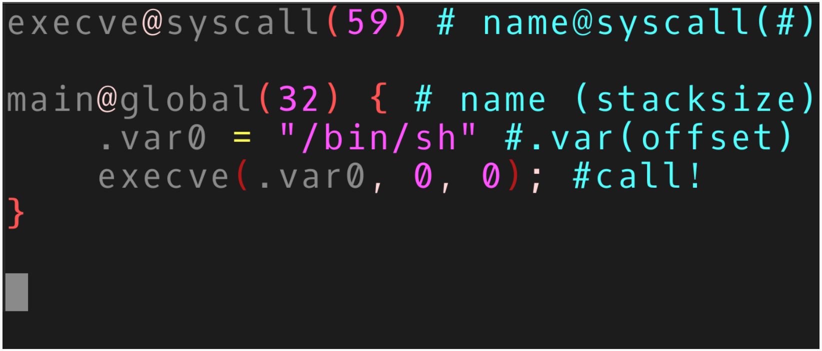
shellcoding

- ragg2 isn't just for generating patterns
- front-end for generating shellcodes
- still up to you to ensure null-free, etc.

shellcoding

- relocatable
- testable (compile directly into elf)
- call arbitrary syscalls easily!
- x86, amd64, arm, windows, mac, linux, ios

shellcoding



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

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

- ragg2 file.r -s to show the emmitted asm.

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

rop

- 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]

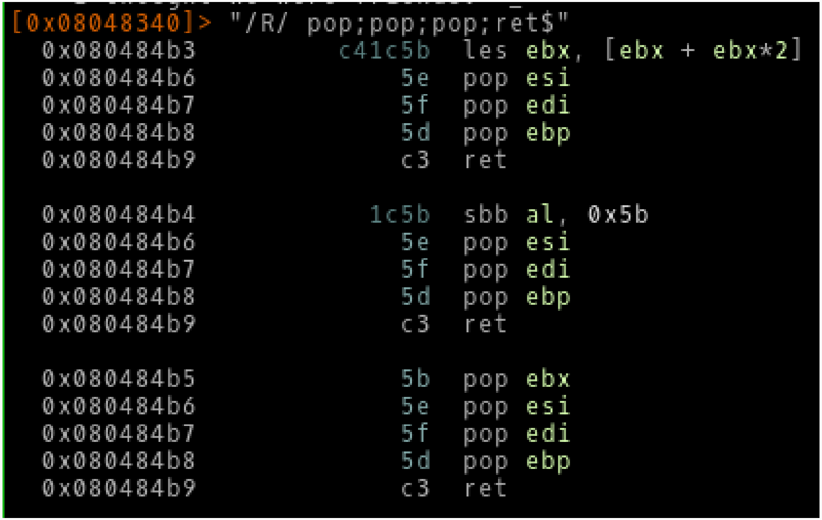
searching for gadgets

- 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.

demo time

- super basic rop expl.
- combine finding sections, patterns, rop search.
- r2 makes this easy

searching for gadgets



```
[0x08048340]> "/R/ pop;pop;pop;ret$"
  0x080484b3           c41c5b  les ebx, [ebx + ebx*2]
  0x080484b6               5e  pop esi
  0x080484b7               5f  pop edi
  0x080484b8               5d  pop ebp
  0x080484b9               c3  ret

  0x080484b4             1c5b  sbb al, 0x5b
  0x080484b6               5e  pop esi
  0x080484b7               5f  pop edi
  0x080484b8               5d  pop ebp
  0x080484b9               c3  ret

  0x080484b5               5b  pop ebx
  0x080484b6               5e  pop esi
  0x080484b7               5f  pop edi
  0x080484b8               5d  pop ebp
  0x080484b9               c3  ret
```

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
```

windbg

r2 allows to connect WinDBG/KD[1]

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

[1] *WinDbg in radare2* (2014).

debugging omap bootrom

Just run it in the modified qemu https://github.com/XVilka/qemu

./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

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
```

firmware analysis

uefi analysis

- Dump the image using flashrom or hardware
- Unpack the image using UEFITool[2]
- Open the selected PE or TE file using r2

[2]Nicolaj Shlej (2013). https://github.com/LongSoft/UEFITool.

old legacy bios analysis

- Load the whole image or unpack it using bios_extract[3]
- Open it using the correct segment and offset
- r2 load the whole BIOS image automatically
- r2 asrock_p4i65g.bin
- ¿. asrock_p4i65g.r2

[3] *Bios_extract* (2015).

the t command — types management

1. Get Usage on the command t?[4]
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

[4] *Radare2 types command* (2014).

embedded controller - 8051

Lets start from the static analysis

```
r2 -a 8051 ite_it8502.rom
```

```
>. ite_it8502.r2
```

embedded controller - 8051 - esil vm[5]

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

[5] *ESIL emulation in radare2* (2014).

embedded controller - 8051 - esil2reil

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

store this to some file and use the ‘openreil’ utility to SMT it

documentation

- Website: http://rada.re/
- Blog: http://radare.today
- Book: http://maijin.gitbooks.io/radare2book/content/

references

a lot of them I

references

Bios_extract (2015).

ESIL emulation in radare2 (2014).

Radare2 types command (2014).

Shlej, Nicolaj (2013). https://github.com/LongSoft/UEFITool.

WinDbg in radare2 (2014).