radare2 as a tool for ctf

Basic reference

Anton Kochkov (@akochkov)

June 4, 2017

KeenLab of Tencent, TCTF 2017

introduction

history

- Incepted in 2006 by Sergi Alvarez aka Pancake (@trufae) as Radare1
- Initially designed as a forensic tool
- Rewritten from scratch in 2009 for modular architecture
- Scriptable and extendable framework, not just a tool
- Gained popularity only a few years ago
- Participation in GSoC[1] and institutioning own RSoC[2]

[1] Radare2 Team (2017a). *GSoC 2017*.
[2] Radare2 Team (2017g). *RSoC 2017*.

short overview

- Core part written in the pure C without dependencies
- Highly portable to different platforms
- UNIX (KISS) -like tools
- Around 1000 commands
- More than 400 configuration options
- Supports both static and dynamic analysis
- Own package manager and external plugins infrastructure
- Windows support from the box (and GSoC task)

installation

- First rule - don't use packaged version
- sys/install.sh on *nix, including OS X
- prebuilt EXE on Windows (best in ConEmu)
- Current release version is **1.5.0**
- Debian (and Ubuntu?) still ship **0.9.6**
- Use version from git for bug-reporting

toolset review

- **Radare2** - the main tool, incorporate everything
- **Rabin2** - parsing binaries
- **Radiff2** - diffing binaries
- **Rafind2** - searching
- **Ragg2** - writing shellcodes
- **Rahash2** - calculating hashes
- **Rarun2** - setting up running environment
- **Rasm2** - CLI assembler/disassembler

short cheatsheet

commands concept

- Most of the commands are abbreviations
- Similar as in GDB (**info registers** -> **i r**) but w/o space[3]
- See registers - dr (Debug), aer (ESIL)
- dr == debug register
- aer == analysis ESIL register
- Each command has internal help - just add **'?'**
- Most of the commands has JSON output - just add **'j'**
- Case sensitive (see commands' count)

[3] Radare2 Team (2017e). *Migration from IDA Pro, GDB or WinDbg*.

basic disassembly[4]

- **rasm2 -a arm -b 64 -d 058d00f8**
- **rasm2 -a arm "mov r0, 5"**
- pd [N] - linear disassembly of [N] insturctions
- pi [N] - the same, just instructions

Those 3 below require functions defined first (analysis)

- pdf [function name] - linear disassembly of function
- pdr - recursive disassembly across function graph
- pdc - basic **C-like** pseudocode

[4] Radare Team (2017c). *R2 Book - Disassembly*.

basic debug[5]

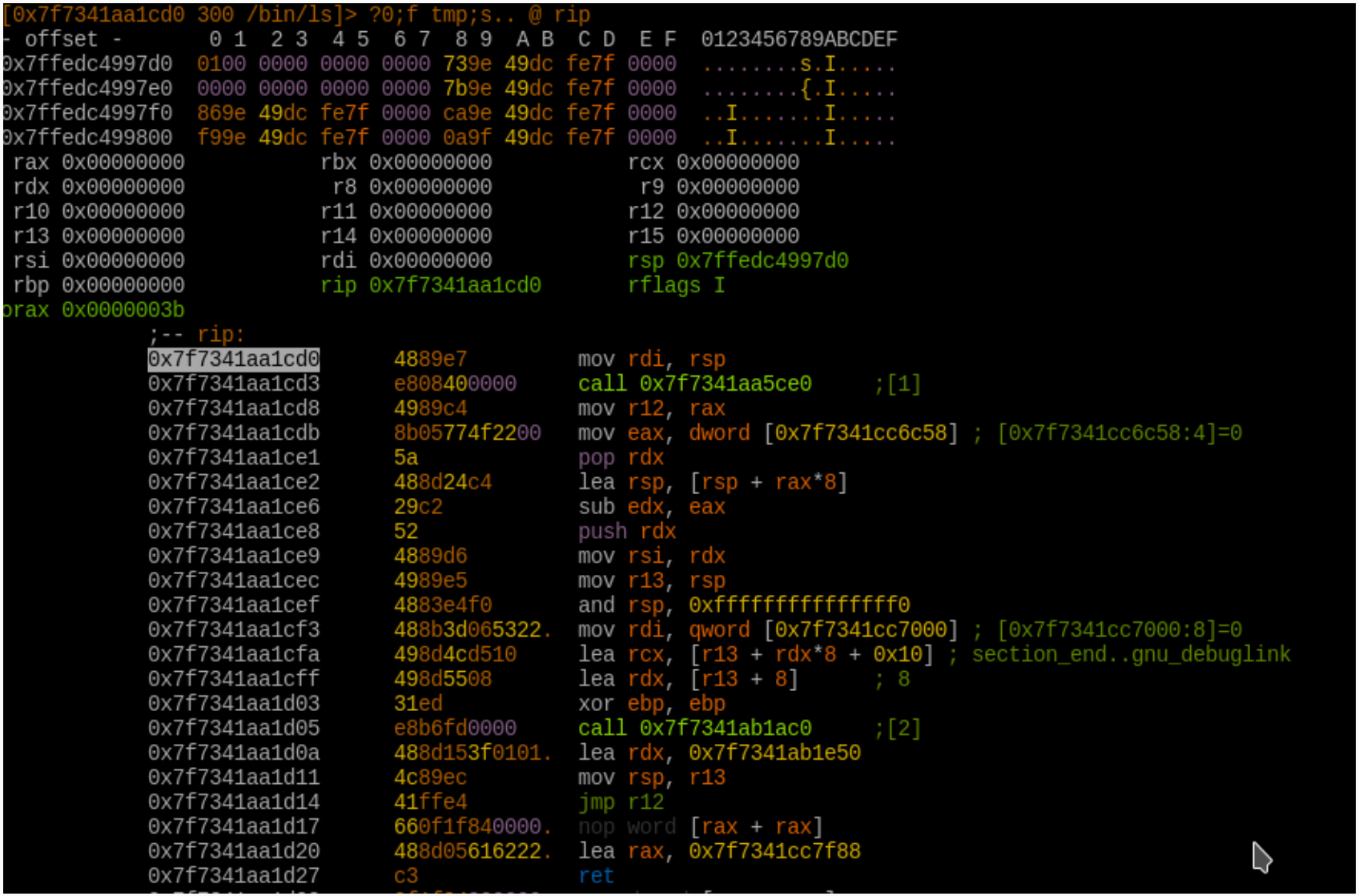
- Add **"-d"** option to radare2
- Or reopen it using ood command
- **r2 -d /bin/ls**

Debug commands are under **"d"** namespace - see d?

- db - Setting breakpoint
- dc - Continuing execution
- ds - Single step
- ds [address] - Step until
- Functional keys in visual mode, see V?

[5] Radare Team (2017b). *R2 Book - Debugger*.

visual debug



basic patching[6]

- Add **"-w"** option to radare2 to open in write mode
- Or reopen it using oo+ command

Writing commands are under **"w"** namespace - see w?

- w - Write ASCII string
- wx - Write hexadecimal sequence
- wa - Write assembly
- wtf - Write to file

[6] Radare Team (2017a). *R2 Book - Basic Commands - Write*.

visual modes

- Can use UTF-8 and True color
- e scr.utf8=true ; e scr.truecolor=true
- All visual modes are under **"V"** namespace
- Just press **V** and loop forward/back using **p**/**P**
- VV - ASCII graphs
- V_ - HUD mode
- VF - navigation through flags/symbols (supports p/P too)
- Ve - navigation (and preview) of internal config variables
- Vt - navigation through types
- Has different hotkeys - see V?

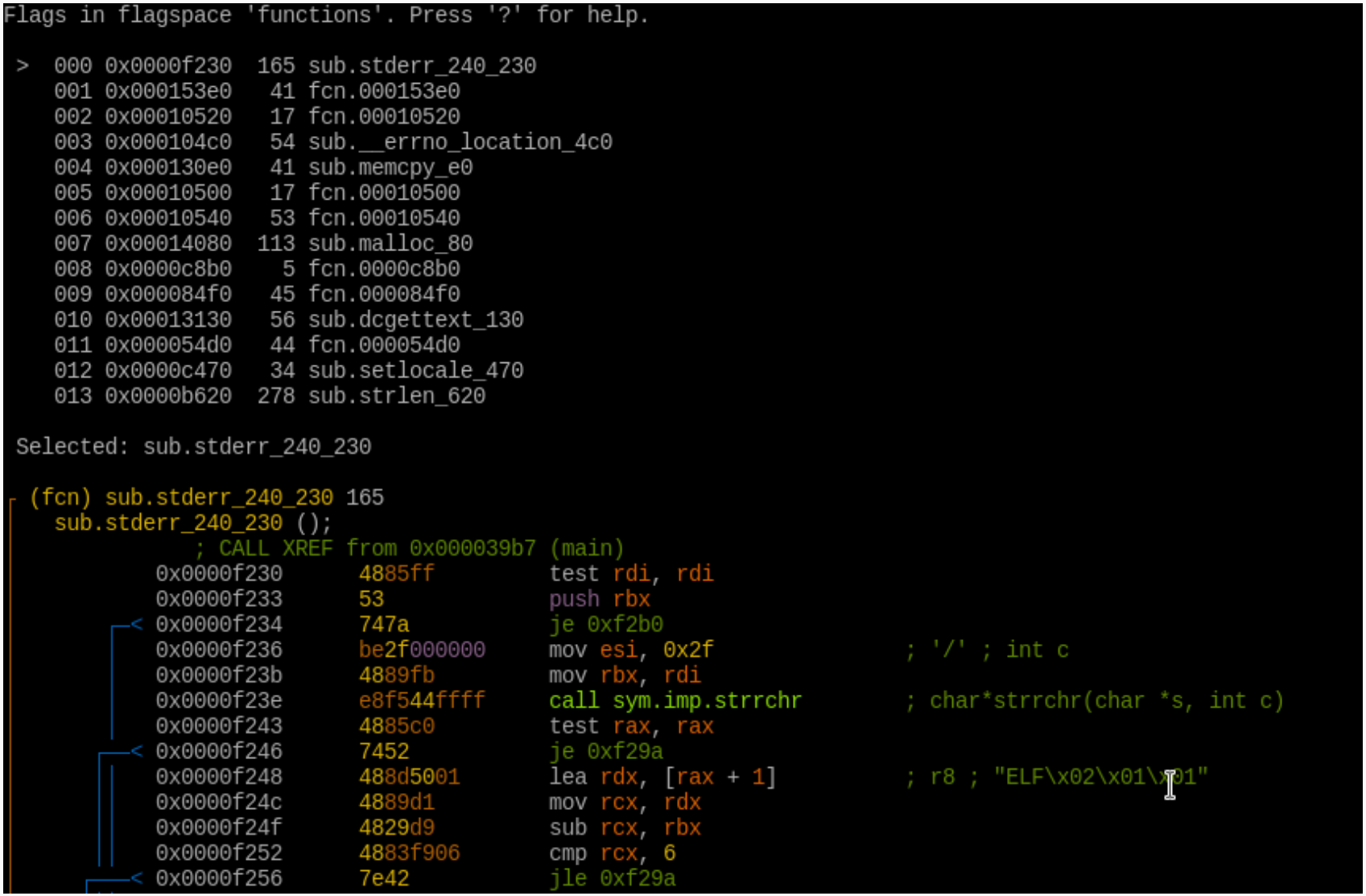
ve - visual config

```
>  asm.esil = true
   asm.family = false
   asm.fcncalls = true
   asm.fcnlines = true
   asm.features =
   asm.filter = true
   asm.flags = true
   asm.flagsinbytes = false
   asm.flgoff = false
   asm.functions = true
   asm.hints = false
   asm.indent = false
   asm.indentspace = 2
   asm.invhex = false
   asm.jmphints = true

Selected: asm.esil (Show ESIL instead of mnemonic)

           ;-- section..text:
┌ (fcn) main 265
   main ();
           ; var int local_38h @ rsp+0x38
              ; DATA XREF from 0x0000535d (entry0)
           0x00003990      4157            r15,8,rsp,-=,rsp,=[8]       ; section 13 va=0x00003990 pa=0x00003990 s
           0x00003992      4156            r14,8,rsp,-=,rsp,=[8]
           0x00003994      4155            r13,8,rsp,-=,rsp,=[8]
           0x00003996      4154            r12,8,rsp,-=,rsp,=[8]
           0x00003998      4189fc          rdi,r12d,=,0xffffffff,r12,&=
           0x0000399b      55              rbp,8,rsp,-=,rsp,=[8]
           0x0000399c      53              rbx,8,rsp,-=,rsp,=[8]
           0x0000399d      4889f3          rsi,rbx,=
           0x000039a0      4883ec48        72,rsp,-=,$o,of,=,$s,sf,=,$z,zf,=,$p,pf,=,$b8,cf,= ; 'H'
           0x000039a4      488b3e          rsi,[8],rdi,=
           0x000039a7      64488b042528.   0x28,[8],rax,=             ; [0x28:8]=0x200f0 ; '('
           0x000039b0      4889442438      rax,0x38,rsp,+,=[8]
           0x000039b5      31c0            rax,eax,^=,$z,zf,=,$p,pf,=,$s,sf,=,$0,cf,=,$0,of,=,0xffffffff,rax,&=
           0x000039b7      e874b80000      rip,8,rsp,-=,rsp,=[],62000,rip,= ; sub.stderr_240_230
           0x000039bc      488d35526401.   0x16452,rip,+,rsi,=         ; 0x19e15 ; const char * locale
           0x000039c3      bf06000000      6,rdi,=                     ; int category
           0x000039c8      e8cbfeffff      rip,8,rsp,-=,rsp,=[],14488,rip,= ; sym.imp.setlocale ; char *setlocale
           0x000039cd      488d354c3401.   0x1344c,rip,+,rsi,=         ; str._usr_share_locale ; 0x16e20 ; "/usr/
           0x000039d4      488d3d2b3401.   0x1342b,rip,+,rdi,=         ; str.coreutils ; 0x16e06 ; "coreutils"
```

vf - visual flagspaces



analysis

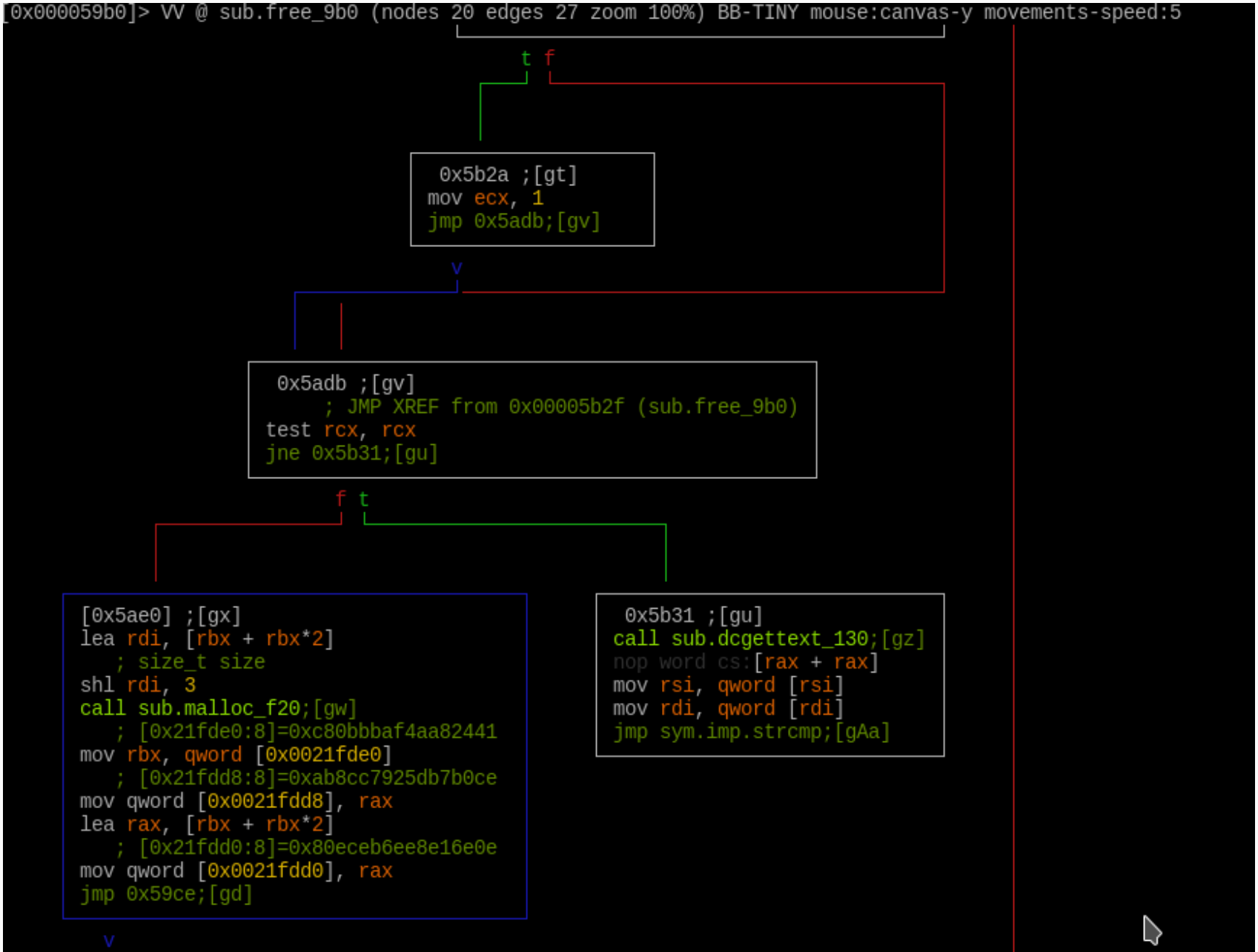
analysis options

- Located in **e anal.*** config var namespace
- For example enabling jump tables analysis e anal.jmptbl=true
- We don't enable all analysis by default
- **r2 -A /bin/ls** - open with autoanalysis
- **r2 -e bin.demangle=true /bin/ls** - open with demangling
- e asm.demangle=true - show demangled symbols in disassembly
- e pdb. - options related to loading/downloading PDBs
- e asm. - options to control information shown in disasm

analysis commands

- There are a few autoanalysis commands: aa, aaa and aaaa
- Those offer different analysis depth
- All analysis commands are under **"a"** namespace - see a?
- **r2 -A**, **r2 -AA**,.. - same with options
- af - analyze the function (see also af?)
- av - show virtual tables (C++)
- aa? - particular commands for recursive analysis
- aac - analyze function calls
- aad - analyze data references
- ax? - particular commands for analyzing references (e.g. axt)

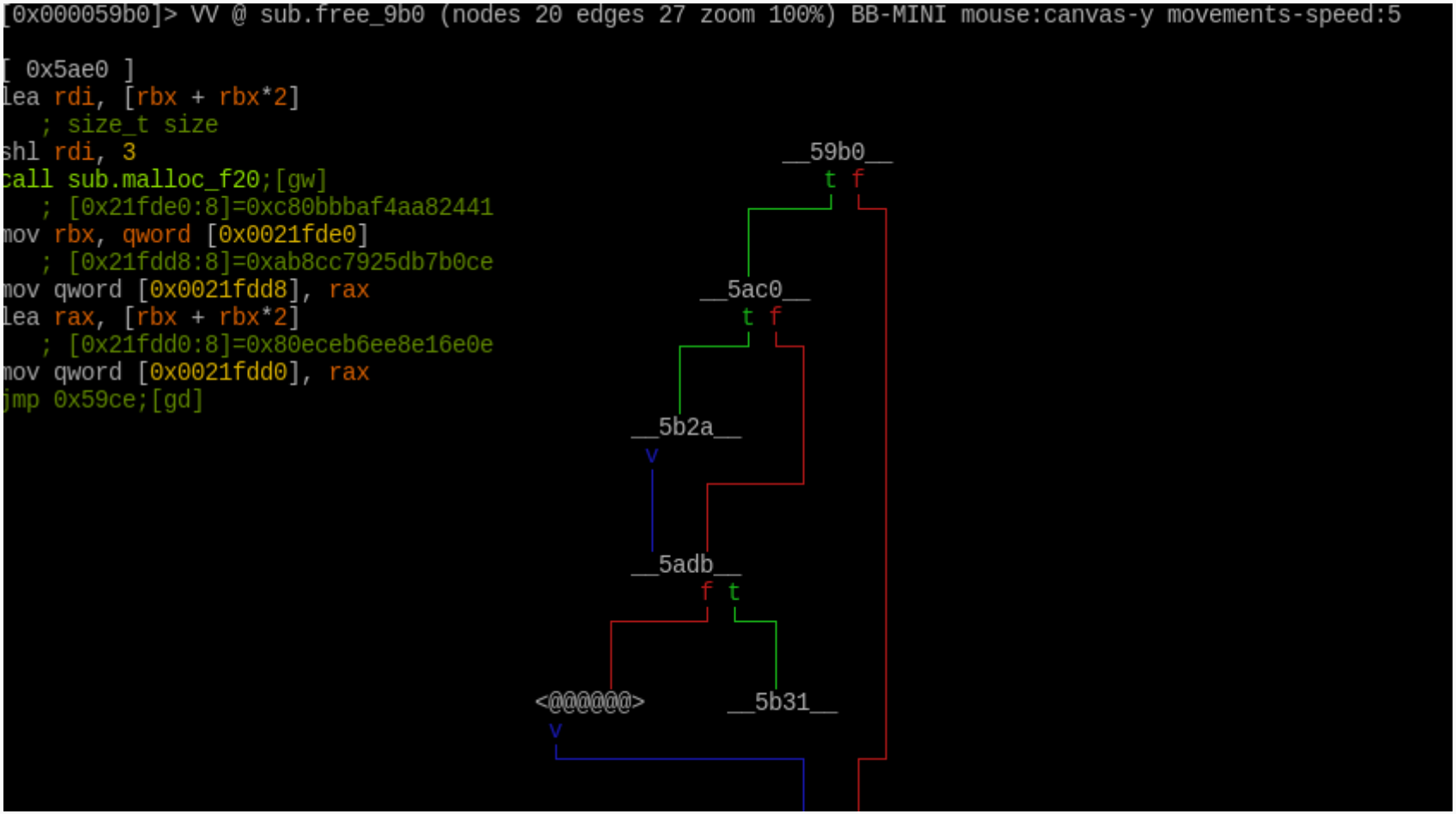
ascii graph


[0x000059b0]> VV @ sub.free_9b0 (nodes 20 edges 27 zoom 100%) BB-TINY mouse:canvas-y movements-speed:5
t f
0x5b2a ;[gt]
mov ecx, 1
jmp 0x5adb;[gv]
v
0x5adb ;[gv]
; JMP XREF from 0x00005b2f (sub.free_9b0)
test rcx, rcx
jne 0x5b31;[gu]
f t
[0x5ae0] ;[gx]
lea rdi, [rbx + rbx*2]
; size_t size
shl rdi, 3
call sub.malloc_f20;[gw]
; [0x21fde0:8]=0xc80bbbaf4aa82441
mov rbx, qword [0x0021fde0]
; [0x21fdd8:8]=0xab8cc7925db7b0ce
mov qword [0x0021fdd8], rax
lea rax, [rbx + rbx*2]
; [0x21fdd0:8]=0x80eceb6ee8e16e0e
mov qword [0x0021fdd0], rax
jmp 0x59ce;[gd]
0x5b31 ;[gu]
call sub.dcgettext_130;[gz]
nop word cs:[rax + rax]
mov rsi, qword [rsi]
mov rdi, qword [rdi]
jmp sym.imp.strcmp;[gAa]
v

ascii/unicode graphs

- After analysis is done graph information is available
- agC - generate Graphviz code
- agf - show ASCII graph of the function
- VV - visual ASCII graph
- It's possible to navigate through nodes using **Tab**/**Shift**+**Tab**
- There is support for **"zoom"** - +/-
- Minimap support - press **p**/**P** to scroll via all modes
- Also graphs information available via JSON (for WebUI)

minimap


[0x000059b0]> VV @ sub.free_9b0 (nodes 20 edges 27 zoom 100%) BB-MINI mouse:canvas-y movements-speed:5
[0x5ae0]
lea rdi, [rbx + rbx*2]
; size_t size
shl rdi, 3
call sub.malloc_f20;[gw]
; [0x21fde0:8]=0xc80bbbaf4aa82441
mov rbx, qword [0x0021fde0]
; [0x21fdd8:8]=0xab8cc7925db7b0ce
mov qword [0x0021fdd8], rax
lea rax, [rbx + rbx*2]
; [0x21fdd0:8]=0x80eceb6ee8e16e0e
mov qword [0x0021fdd0], rax
jmp 0x59ce;[gd]
__59b0__
t f
__5ac0__
t f
__5b2a__
__5adb__
f t
<@@@@@@>
__5b31__

types

- There is a support for C types
- All types commands are located in **“t”** namespace - see t?
- to file.h - parse C header file and load types from it
- td struct qwe {int a; char* b;}; - parse type desc
- tp <typename> = <address> - cast data at some addr to type
- tl <typename> = <address> - permanently link type to addr
- tk - raw requests to SDB database (types storage)
- Vt - visual types navigation

arguments

- There is a support for function arguments
- All arguments commands are located in **“afv”** namespace - see afv?
- afv - show locals/arguments
- afvn - rename argument/local
- afvr? - manipulation of register based arguments
- afvs? - manipulation of SP based arguments
- afvb? - manipulation of BP based arguments
- afta - analyze all arguments + types across file

esil[8]

- R2 has its own Intermediate Language and its VM
- It's RPN form (Reverse Polish Notation)[7]
- Allows to emulate parts of the programs
- Supported architectures are x86, ARM, MIPS, etc...
- Being used in the deep analysis for resolving indirect jumps
- ao esil - shows the ESIL representation of current instruction
- ae eax,5,+= - evaluate ESIL expression
- e asm.esil=true - show ESIL instead of disassembly

[7] Radare2 Team (2015b). *ESIL Instruction Set*.
[8] Radare2 Team (2015a). *ESIL description*.

memory inspection

For this R2 need to be started in debug mode (**“-d”**)

- dm? commands allows you to inspect the memory
- dmm - shows loaded libraries
- dm - inspecting the memory map
- dmh? - commands to inspect the heap
- dmhg - show the ASCII graph of the heap
- dmhb - display parsed bins for selected heap arena
- dbta - show the ASCII graph of call stack

stack visualization

```
[0x55f398302840]> dmh?
|Usage:  dmh # Memory map heap
| dmh                                    List chunks in heap segment
| dmh [malloc_state]                     List heap chunks of a particular arena
| dmha                                   List all malloc_state instances in application
| dmhb                                   Display all parsed Double linked list of main_arena's bins instance
| dmhb [bin_num|bin_num:malloc_state]    Display parsed double linked list of bins instance from a particular arena
| dmhbg [bin_num]                        Display double linked list graph of main_arena's bin [Under developemnt]
| dmhc @[chunk_addr]                     Display malloc_chunk struct for a given malloc chunk
| dmhf                                   Display all parsed fastbins of main_arena's fastbinY instance
| dmhf [fastbin_num|fastbin_num:malloc_state]  Display parsed single linked list in fastbinY instance from a particular arena
| dmhg                                   Display heap graph of heap segment
| dmhg [malloc_state]                    Display heap graph of a particular arena
| dmhi @[malloc_state]                   Display heap_info structure/structures for a given arena
| dmhm                                   List all elements of struct malloc_state of main thread (main_arena)
| dmhm [malloc_state]                    List all malloc_state instance of a particular arena
| dmh?                                   Show map heap help
[0x55f398302840]> dmhb
Bins {
 Bin 001:
  double linked list unsorted bin {
    0x7f9114d5cb38->fd = 0x7f9114d5cb38
    0x7f9114d5cb38->bk = 0x7f9114d5cb38
  }
  Empty bin  0x0
 Bin 002:
  double linked list small bin {
    0x7f9114d5cb48->fd = 0x7f9114d5cb48
    0x7f9114d5cb48->bk = 0x7f9114d5cb48
  }
  Empty bin  0x0
 Bin 003:
  double linked list small bin {
    0x7f9114d5cb58->fd = 0x7f9114d5cb58
    0x7f9114d5cb58->bk = 0x7f9114d5cb58
  }
  Empty bin  0x0
 Bin 004:
  double linked list small bin {
    0x7f9114d5cb68->fd = 0x7f9114d5cb68
    0x7f9114d5cb68->bk = 0x7f9114d5cb68
  }
  Empty bin  0x0
```

stack visualization

```
[0x7f911518ecd0]> db sym.imp.free
[0x7f911518ecd0]> db sym.imp.malloc
[0x7f911518ecd0]> dc
Selecting and continuing: 12334
hit breakpoint at: 55f398302840
[0x55f398302840]> dbta
0x00007ffc6e722000  STACK END  ^^^
0x00007ffc6e741898  STACK POINTER: rsp
                   .-----------------------.
0x00007ffc6e741898  | rsp     [frame 0]     | ; size -702404264
                   |           ...         |
0x000055f39851ef40  | rbp 0x000055f398302840 | ; return address
                   )-----------------------(
0x00007ffc6e741898  | rsp     [frame 1]     | ; size 0
                   |           ...         |
0x00007ffc6e741898  | rbp 0x000055f398311f29 | ; return address
                   )-----------------------(
0x00007ffc6e7418a8  | rsp     [frame 2]     | ; size 16
                   |           ...         |
0x00007ffc6e741898  | rbp 0x000055f3983120f4 | ; return address
                   )-----------------------(
0x00007ffc6e7418c8  | rsp     [frame 3]     | ; size 32
                   |           ...         |
0x00007ffc6e7418a8  | rbp 0x000055f39830f4ed | ; return address
                   )-----------------------(
0x00007ffc6e7418e8  | rsp     [frame 4]     | ; size 32
                   |           ...         |
0x00007ffc6e7418c8  | rbp 0x000055f398302cee | ; return address
                   )-----------------------(
0x00007ffc6e741968  | rsp     [frame 5]     | ; size 128
                   |           ...         |
```

debugging

native debugging

- For this R2 need to be started in debug mode (**"-d"** option)
- Or reopen using ood
- dr= to show the registers
- dbt - display backtrace
- dt? - tracing features
- dp? - listing/attaching/etc to processes
- dk? - working with signals

gdb remote connection

- Just connect using **gdb://** protocol
- **gdbserver /bin/ls**
- **r2 -D gdb gdb://localhost:1234**
- Support the same commands as other debug engines
- Can be used to connect GDBserver of QEMU, VMWare, etc
- This year our GSoC student also working on improving[9]
- And also working on small gdbserver implementation

[9] Radare2 Team (2017c). *GSoC 2017 Tasks - GDB server and protocol*.

windbg remote connection

- Just connect using **windbg://** protocol
- Run VBox or QEMU with Windows debug over pipes
- **r2 -D wind windbg:///tmp/windbg.pipe**
- Support the same commands as other debug engines
- This year another GSoC student also working on improving this[10]
- For now it supports only connection via serial (pipe) - no network
- Our own WinDbg protocol parser[11] - can be reused in other projects

[10]Radare2 Team (2017d). *GSoC 2017 Tasks - Windows platform support*.
[11]Radare2 Team (2015d). *WinDbg protocol parser*.

integration with frida[13]

- Just install Frida and r2frida[12] (**r2pm -i r2frida**)
- **r2 frida://<phone_id>/Appname**
- r2frida implements IO layer smoothly redirecting commands

[12] *r2frida* (2016).
[13] *Frida* (2015).

esil[14] debugging

- aei - initialize ESIL VM
- aeim - initialise memory for selected parameters
- aeip - point ESIL VM to current IP
- aer eax=0x5 - set register value
- aes - step using ESIL emulation
- aecu - continue until using ESIL emulation
- Also stepping using ESIL is available in visual debug mode

[14]Radare2 Team (2015a). *ESIL description*.

timeless debug

R2 supports basic timeless debug. Moreover, this year GSoC[15] student working on improving it.

- dts+ - add the trace session
- dts - show the active trace sessions
- dms? - work with memory snapshots
- dsb - single step back

[15]Radare2 Team (2017b). *GSoC 2017 accepted projects*.

signatures

native signatures

Radare2 has different searching capabilities[16] including signatures:

- z - show signatures status and z* to list them
- zb - define signature in place
- zo - manage signature files
- zos - save the signature into file
- z/ - search signatures
- zs - manage singature spaces

[16]Radare Team (2017e). *R2 Book - Searching*.

flirt and yara

- R2 has support for loading FLIRT signatures (IDA < 6.8)
- zfs - load FLIRT signature from the file and scan
- zfd - dump the contents of the FLIRT file
- zfz - convert FLIRT file to native signature format

R2 can support Yara via plugin: **r2pm -i yara-lib; r2pm -i yara**

- yara add - load Yara rules from the file
- yara scan - to scan the file for matches
- yara list - show loaded Yara signatures
- yara clear - clear all loaded rules

scripting

internal scripting[17]

- R2 has variables - see ?$? output
- R2 supports iterators - see ?@? output
- pd 1; ao 1 - sequence the commands
- ao | grep esil - pipe into external commands
- #! tcpdump - run system command
- px 10 @ ‘ao ptr[1]‘ - like shell backtick
- Redirecting output using > and »

[17]Radare Team (2017d). *R2 Book - Scripting*.

loops, macroses and aliases

- afi @@ fcn.* name - grep through output of afi of matching
- ao @@=$$ $$2 - looping over cur offset and cur offset +2
- pi 1 @@i - looping over instructions in the funtion
- afi @@@ functions name[1] - looping over functions
- (qwe, pd4, ao) - add macro **"qwe"**, call with .(qwe)
- (foo x y, pd $0; s +$1) - macro with arguments
- $<aliasname>=<command> - define an alias
- $dis='pi 1;ao', then call $dis

r2pipe

- R2 has a mechanism for interacting with scripts via pipe[18]
- **pip install r2pipe**, **npm install r2pipe**
- In python just do **import r2pipe** and use provided commands
- Python r2pipe scripts can be run from internal shell
- . myscript.py - just use source command .

[18]Radare2 Team (2015c). *R2pipe repository*.

r2pipe

```
import r2pipe

r2 = r2pipe.open("/bin/ls")
r2.cmd('aa')
func_lst = r2.cmdj("aflj")
print(func_lst)
r2.quit()
```

angr

- R2 supports getting information from angr
- **r2pm -i r2angr**
- **r2 angr:///bin/ls** - open the file via angr
- afl - list all functions recognized by angr
- Can be extended - see **radare2-extras/r2angr/** directory[19]

[19]Radare2 Team (2017f). *r2angr*.

contributing

contributing

- Main repo https://github.com/radare/radare2
- Extras - https://github.com/radare/radare2-extras
- Qt GUI - https://github.com/hteso/iaito
- WebUI - https://github.com/radare/radare2-webui
- Radeco - https://github.com/radare/radeco-lib
- You can pick up issues marked as "easy"
- Travis CI, AppVeyor, Jenkins, Coverity, etc

references

references

- *Frida* (2015).
- *r2frida* (2016).
- Team, Radare (2017a). *R2 Book - Basic Commands - Write*.
- — (2017b). *R2 Book - Debugger*.
- — (2017c). *R2 Book - Disassembly*.
- — (2017d). *R2 Book - Scripting*.
- — (2017e). *R2 Book - Searching*.
- Team, Radare2 (2015a). *ESIL description*.
- — (2015b). *ESIL Instruction Set*.
- — (2015c). *R2pipe repository*.
- — (2015d). *WinDbg protocol parser*.

a lot of them II

- Team, Radare2 (2017a). *GSoC 2017*.
- — (2017b). *GSoC 2017 accepted projects*.
- — (2017c). *GSoC 2017 Tasks - GDB server and protocol*.
- — (2017d). *GSoC 2017 Tasks - Windows platform support*.
- — (2017e). *Migration from IDA Pro, GDB or WinDbg*.
- — (2017f). *r2angr*.
- — (2017g). *RSoC 2017*.