Introductory malware RE with radare

sha0 // r2con2024
@

Introduction

What Am I Doing Here?

- What is r2?
- How to use the shell
- Analyzing
- Debugging
- Patching
- Scripting

Why Radare2?

- It’s free and opensource
- Runs everywhere (Windows, Mac, Linux, QNX, iOS, ..)
- Easy to script and extend with plugins
- Embeddable
- Grows fast
- Supports tons of file-formats
- Handles gazillions of architectures
- Easy to hack
- Commandline cowboy-friendly
- Great community and even better leader
- Collaborative

Why I use Radare2?

- Unique features
- Reproduce a situation
- Multiple samples
- Patches
- Advanced surgery
- Quick automation (bash + r2 -c, @@=`iz~1024[1]`, r2pipe)
- Defeating anti-attachs
- Recognizing functions even with wiped PE structures
- Document the work with text instead of screenshots
- r2pm ecosystem
- Export bytes to c, python, etc.

What's Radare2?

- Reverse Engineering
  - Analyze Code/Data/..
  - Understanding Programs
- Low Level Debugging
  - Similar to olly
  - Multi-platform, and support for remote
- Forensics
  - File Systems
  - Memory Dumps
- Assembler/Disassembler
  - Several architectures
  - Multiplatform

Tools

Radare2 is composed by some core libraries and a set of tools that use those libraries and plugins.

radare2

rabin2

rasm2

r2pm

radiff2

rafind2

rarun2

rax2

r2agent

ragg2

rahash2

rasign2

History

Radare was born in 2006 as a forensic tool for performing manual and interactive carving to recover some files from disk or ram.

It grew quickly adding support for disassembler, debugger, code analyzer, scripting, …

And then I decided to completely rewrite it to fix the maintenance and monolithic design problems.

But First.. A Poll!

(who are you?)

Which is your main OS?

Do you know assembly?

Do you program? c? python?

How’s your UNIX foo?

Did you used r2 before?

Installation

(always use git)

PROTIP: Installing radare2 is recommended method to use it.

Installing from Git

```
$ git clone https://github.com/radareorg/radare2.git

$ cd radare2

$ sys/install.sh

   or

$ sys/user.sh
```

Package Management

```
$ r2pm update

$ r2pm -s yara

$ r2pm -i yara

$ r2pm -i r2ghidra

$ r2pm -i r2ghidra-sleigh
```

Package Management

Some of the most interesting packages:

- Yara (2 / 3)
- RetDec decompiler (@nighterman)
- Unicorn - code emulator
- Native Python bindings
- Duktape (Embedded javascript)
- Radeco decompiler (@sushant94)
- Baleful (SkUaTeR)
- r2pipe apis for NodeJS, Python and Ruby
- Vala/Vapi/Valabind/Swig/Bokken/...

Loading a binary

```
r2 -n sample.bin
r2 sample.bin
```

Parse structure

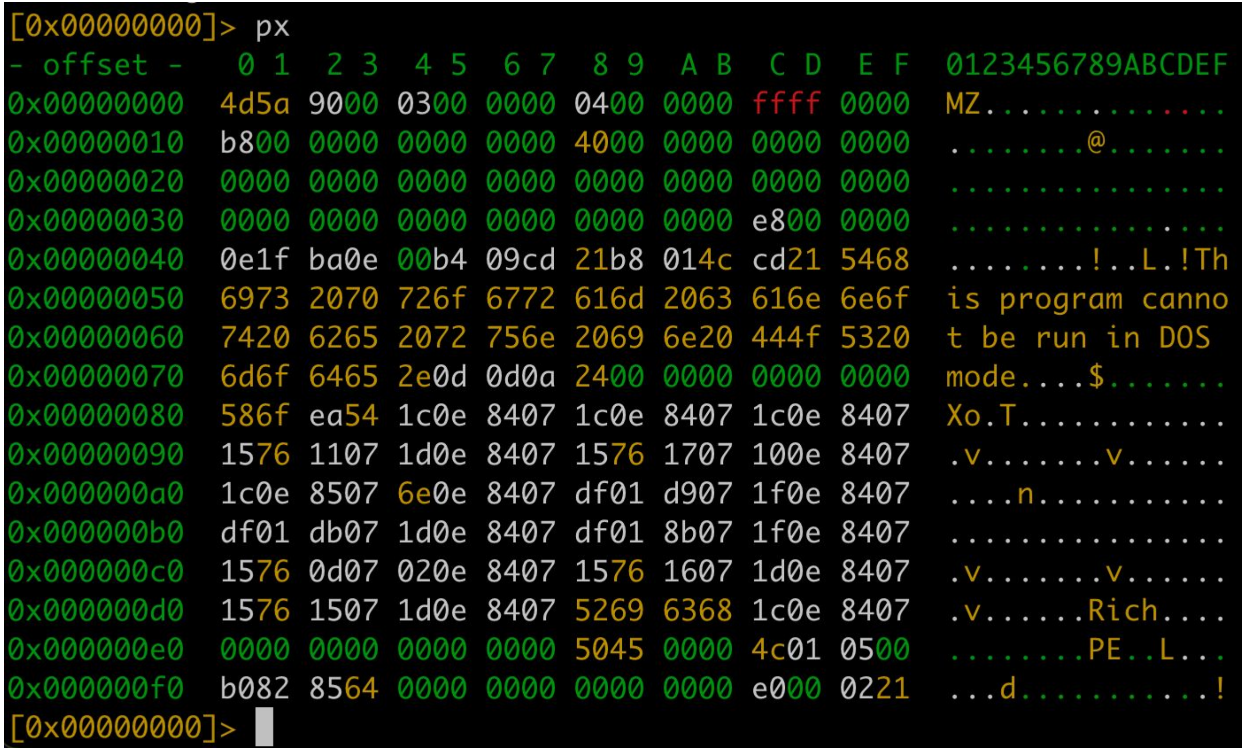
VS

Don’t parse.

-n Load the binary as is from disk.

Otherwise would parse the headers as the OS loader does. So will have virtual addresses, sections, etc.

-n mode



```
[0x00000000]> px
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000000  4d5a 9000 0300 0000 0400 0000 ffff 0000  MZ..............
0x00000010  b800 0000 0000 0000 4000 0000 0000 0000  ........@.......
0x00000020  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00000030  0000 0000 0000 0000 0000 0000 e800 0000  ................
0x00000040  0e1f ba0e 00b4 09cd 21b8 014c cd21 5468  ........!..L.!Th
0x00000050  6973 2070 726f 6772 616d 2063 616e 6e6f  is program canno
0x00000060  7420 6265 2072 756e 2069 6e20 444f 5320  t be run in DOS
0x00000070  6d6f 6465 2e0d 0d0a 2400 0000 0000 0000  mode....$.......
0x00000080  586f ea54 1c0e 8407 1c0e 8407 1c0e 8407  Xo.T............
0x00000090  1576 1107 1d0e 8407 1576 1707 100e 8407  .v.......v......
0x000000a0  1c0e 8507 6e0e 8407 df01 d907 1f0e 8407  ....n...........
0x000000b0  df01 db07 1d0e 8407 df01 8b07 1f0e 8407  ................
0x000000c0  1576 0d07 020e 8407 1576 1607 1d0e 8407  .v.......v......
0x000000d0  1576 1507 1d0e 8407 5269 6368 1c0e 8407  .v......Rich....
0x000000e0  0000 0000 0000 0000 5045 0000 4c01 0500  ........PE..L...
0x000000f0  b082 8564 0000 0000 0000 0000 e000 0221  ...d...........!
[0x00000000]>
```

Patch binary

`Let’s write two nop (0x90) at offset 0xe7:`

- `r2 -n -w sample.bin`
- `s 0xe7`
- `wx 9090`

- `s 0`
- `px`

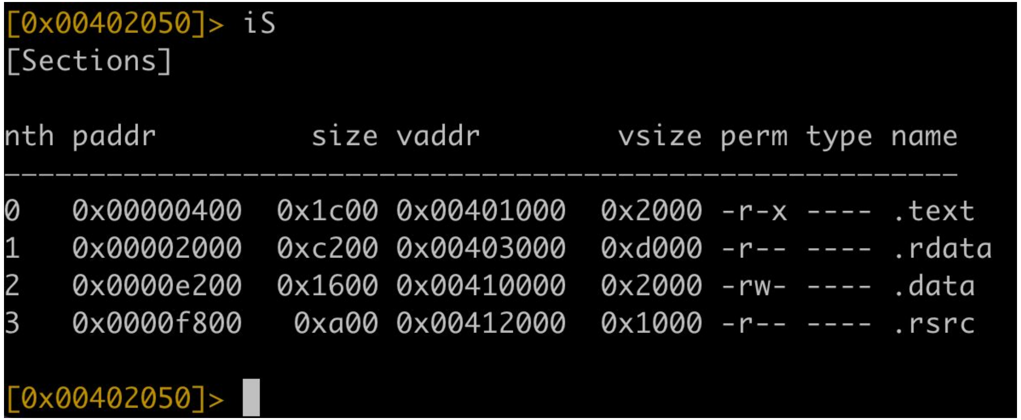
Basic bash automations

What if we want to patch 200 samples?

```
for i in *.bin
do
    r2 -n -w -q -c ‘s 0xe7; wx 9090’ $i
done
```

Loading Executable

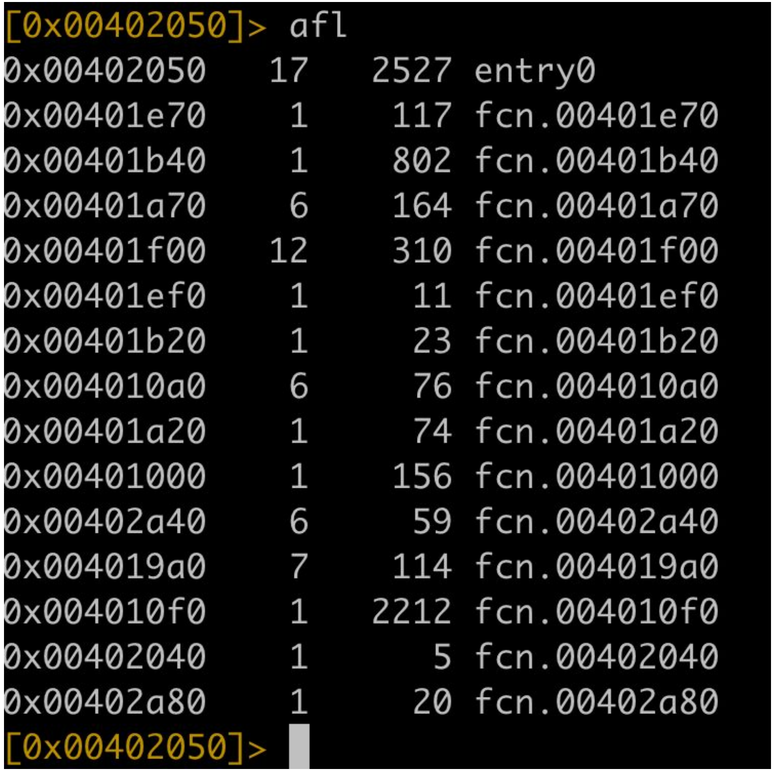
- r2 sample.bin
- iS   # sections
- iE   # exports
- ii   # imports
- is   # symbols
- iz   # strings
- it   # hash
- il   # linked libs
- i?
- iz?



```
[0x00402050]> iS
[Sections]

nth paddr          size vaddr          vsize perm type name
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――
0   0x00000400   0x1c00 0x00401000   0x2000 -r-x ---- .text
1   0x00002000   0xc200 0x00403000   0xd000 -r-- ---- .rdata
2   0x0000e200   0x1600 0x00410000   0x2000 -rw- ---- .data
3   0x0000f800    0xa00 0x00412000   0x1000 -r-- ---- .rsrc

[0x00402050]>
```

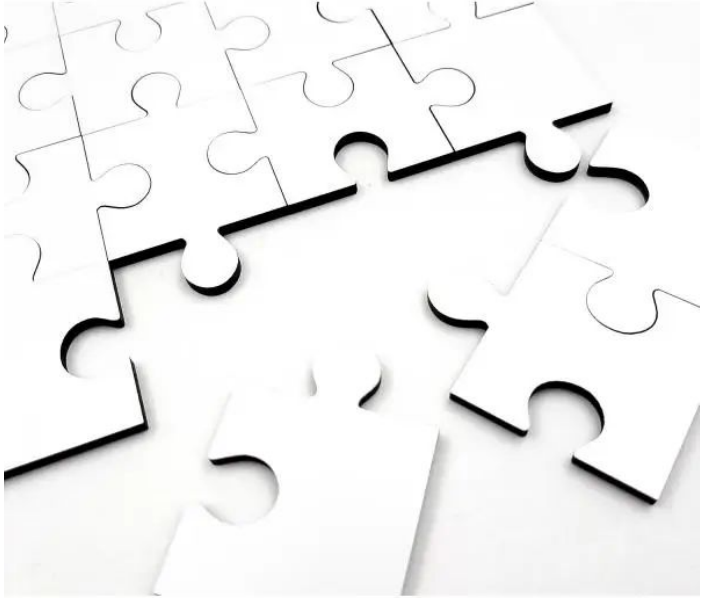
Analyzing Executable

- aaaa
- afl
- afl ~main
- afl ~entry
- s entry0
- pdf
- pdg



```
[0x00402050]> afl
0x00402050   17   2527 entry0
0x00401e70    1    117 fcn.00401e70
0x00401b40    1    802 fcn.00401b40
0x00401a70    6    164 fcn.00401a70
0x00401f00   12    310 fcn.00401f00
0x00401ef0    1     11 fcn.00401ef0
0x00401b20    1     23 fcn.00401b20
0x004010a0    6     76 fcn.004010a0
0x00401a20    1     74 fcn.00401a20
0x00401000    1    156 fcn.00401000
0x00402a40    6     59 fcn.00402a40
0x004019a0    7    114 fcn.004019a0
0x004010f0    1   2212 fcn.004010f0
0x00402040    1      5 fcn.00402040
0x00402a80    1     20 fcn.00402a80
[0x00402050]>
```

Static analysis



How to start?

- `iz     # strings`
- `ii     # imports`

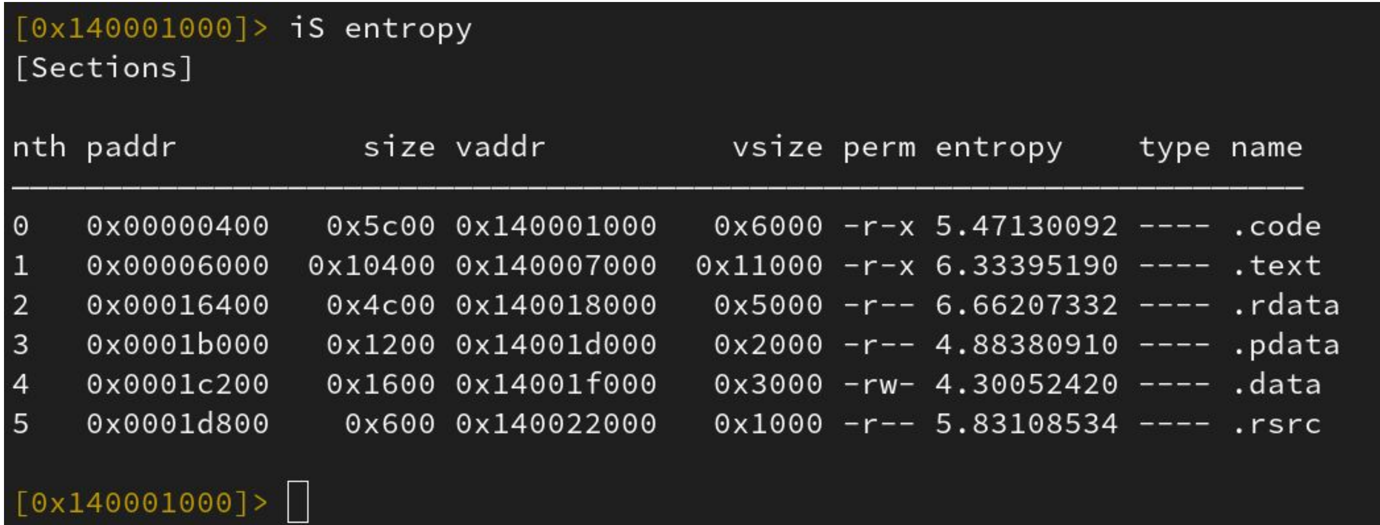
- `entry / main     # starting from the beginning`

Locating communications

- `ii ~connect           # ws2_32`
- `ii ~recv              # ws2_32`
- `ii ~inet_addr         # ws2_32`
- `ii ~gethostbyname     # ws2_32`
- `ii ~InternetConnect   # wininet`
- `ii ~InternetReadFile# wininet`
- `ii ~HttpSendRequest   # wininet`
- `Ii ~WinHttpConnect    # winhttp`

Is it packed?

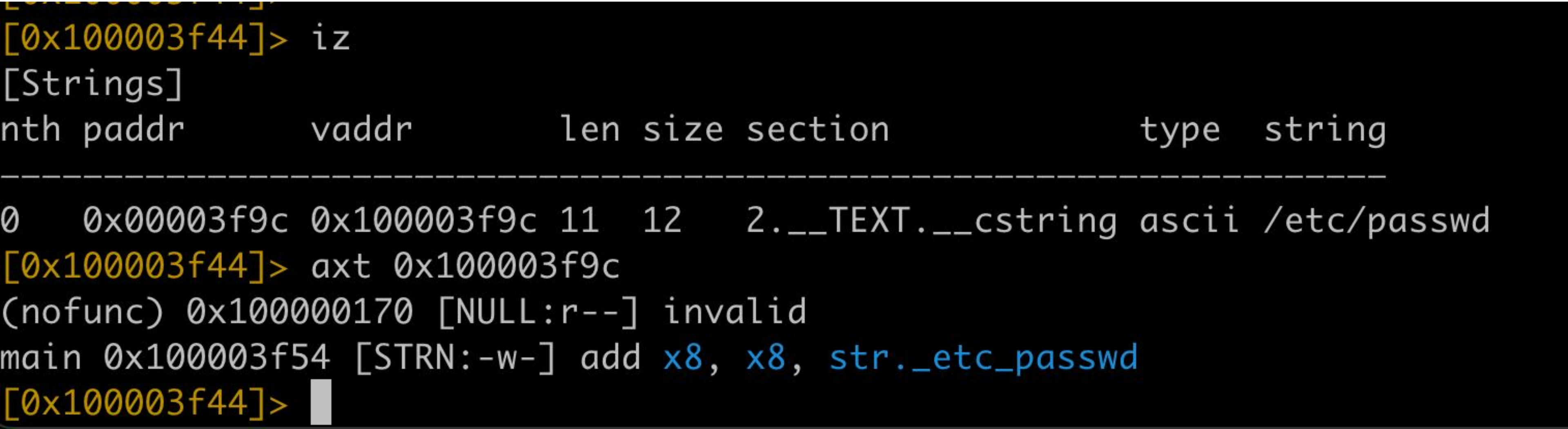
- iS entropy
- axt @@ str*
- axt @@ sub*
- ii
- afl |wc -l



```
[0x140001000]> iS entropy
[Sections]

nth paddr            size vaddr            vsize perm entropy    type name
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――
0   0x00000400     0x5c00 0x140001000     0x6000 -r-x 5.47130092 ---- .code
1   0x00006000    0x10400 0x140007000    0x11000 -r-x 6.33395190 ---- .text
2   0x00016400     0x4c00 0x140018000     0x5000 -r-- 6.66207332 ---- .rdata
3   0x0001b000     0x1200 0x14001d000     0x2000 -r-- 4.88380910 ---- .pdata
4   0x0001c200     0x1600 0x14001f000     0x3000 -rw- 4.30052420 ---- .data
5   0x0001d800      0x600 0x140022000     0x1000 -r-- 5.83108534 ---- .rsrc

[0x140001000]> 
```

XRefs everything

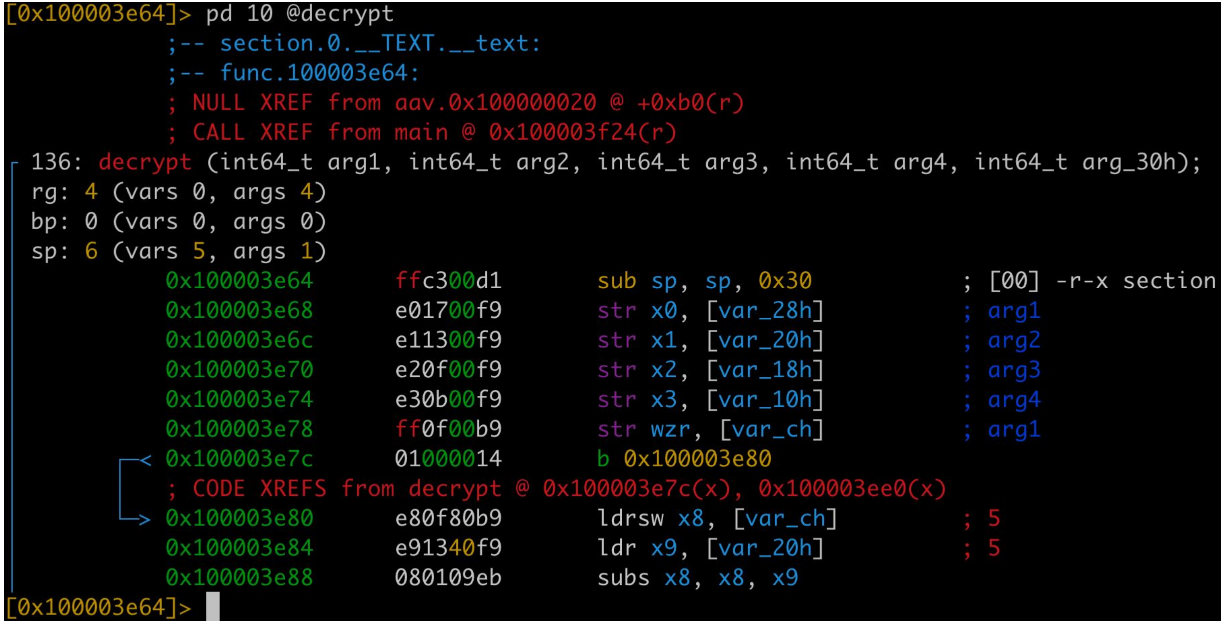
- `axt 0x100003f9c`
- `axt @@ str*`
- ``axt @@=`afll~crypt[0]` ``



```
[0x100003f44]> iz
[Strings]
nth paddr      vaddr       len size section              type  string
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――

0   0x00003f9c 0x100003f9c 11  12   2.__TEXT.__cstring ascii /etc/passwd
[0x100003f44]> axt 0x100003f9c
(nofunc) 0x100000170 [NULL:r--] invalid
main 0x100003f54 [STRN:-w-] add x8, x8, str._etc_passwd
[0x100003f44]> 
```

Renaming everything

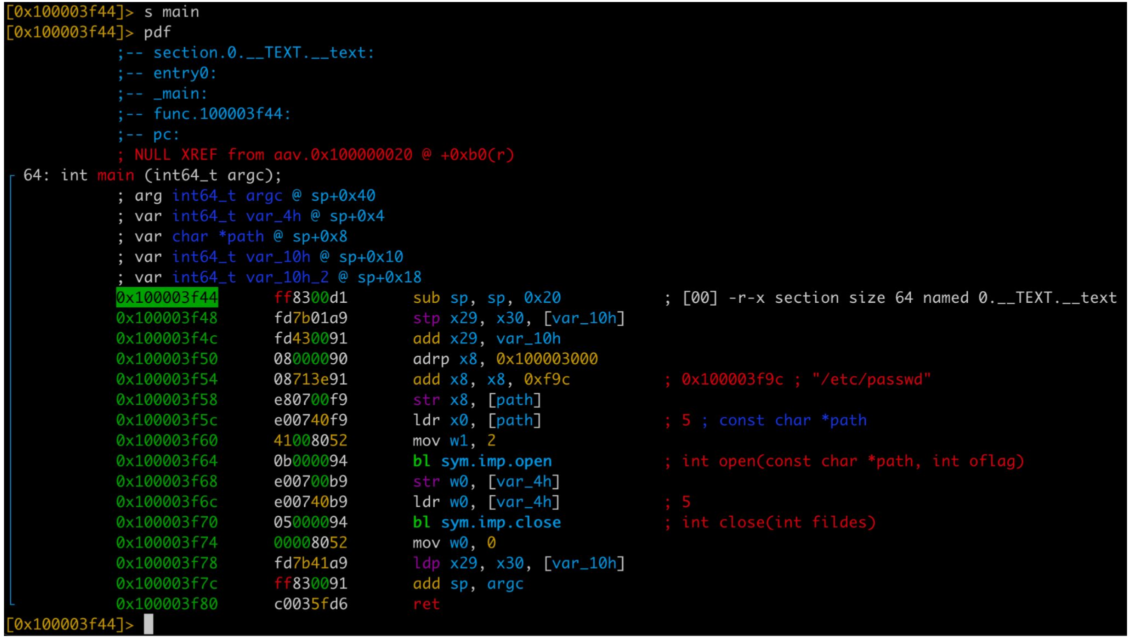
- aaaa
- afn decrypt 0x100003e64



```
[0x100003e64]> pd 10 @decrypt
            ;-- section.0.__TEXT.__text:
            ;-- func.100003e64:
            ; NULL XREF from aav.0x100000020 @ +0xb0(r)
            ; CALL XREF from main @ 0x100003f24(r)
┌ 136: decrypt (int64_t arg1, int64_t arg2, int64_t arg3, int64_t arg4, int64_t arg_30h);
│ rg: 4 (vars 0, args 4)
│ bp: 0 (vars 0, args 0)
│ sp: 6 (vars 5, args 1)
│           0x100003e64      ffc300d1       sub sp, sp, 0x30           ; [00] -r-x section
│           0x100003e68      e01700f9       str x0, [var_28h]          ; arg1
│           0x100003e6c      e11300f9       str x1, [var_20h]          ; arg2
│           0x100003e70      e20f00f9       str x2, [var_18h]          ; arg3
│           0x100003e74      e30b00f9       str x3, [var_10h]          ; arg4
│           0x100003e78      ff0f00b9       str wzr, [var_ch]          ; arg1
│       ┌─< 0x100003e7c      01000014       b 0x100003e80
│       │   ; CODE XREFS from decrypt @ 0x100003e7c(x), 0x100003ee0(x)
│       └─> 0x100003e80      e80f80b9       ldrsw x8, [var_ch]         ; 5
│           0x100003e84      e91340f9       ldr x9, [var_20h]          ; 5
│           0x100003e88      080109eb       subs x8, x8, x9
[0x100003e64]>
```

Viewing functions

- `pdf`
- `pdc`
- `pdg`
- `pd 10 @somefunc`
- `axg`



```
[0x100003f44]> s main
[0x100003f44]> pdf
            ;-- section.0.__TEXT.__text:
            ;-- entry0:
            ;-- _main:
            ;-- func.100003f44:
            ;-- pc:
            ; NULL XREF from aav.0x100000020 @ +0xb0(r)
┌ 64: int main (int64_t argc);
│           ; arg int64_t argc @ sp+0x40
│           ; var int64_t var_4h @ sp+0x4
│           ; var char *path @ sp+0x8
│           ; var int64_t var_10h @ sp+0x10
│           ; var int64_t var_10h_2 @ sp+0x18
│           0x100003f44      ff8300d1       sub sp, sp, 0x20            ; [00] -r-x section size 64 named 0.__TEXT.__text
│           0x100003f48      fd7b01a9       stp x29, x30, [var_10h]
│           0x100003f4c      fd430091       add x29, var_10h
│           0x100003f50      08000090       adrp x8, 0x100003000
│           0x100003f54      08713e91       add x8, x8, 0xf9c           ; 0x100003f9c ; "/etc/passwd"
│           0x100003f58      e80700f9       str x8, [path]
│           0x100003f5c      e00740f9       ldr x0, [path]              ; 5 ; const char *path
│           0x100003f60      41008052       mov w1, 2
│           0x100003f64      0b000094       bl sym.imp.open             ; int open(const char *path, int oflag)
│           0x100003f68      e00700b9       str w0, [var_4h]
│           0x100003f6c      e00740b9       ldr w0, [var_4h]            ; 5
│           0x100003f70      05000094       bl sym.imp.close            ; int close(int fildes)
│           0x100003f74      00008052       mov w0, 0
│           0x100003f78      fd7b41a9       ldp x29, x30, [var_10h]
│           0x100003f7c      ff830091       add sp, argc
└           0x100003f80      c0035fd6       ret
[0x100003f44]>
```

Visual mode

- `V   # to enter`
- `q   # to quit`
- `pagUp / pagDown`
- `p   # change mode`
- `G   # seek to end`
- `g   # seek to offset`

Modes: hex, asm, asm + regs + mem, entropy, strings

And more …

Specifying architecture

```
-a <arch>    # specify architecture (RAsm Plugin name)

-b <bits>    # specify 8, 16, 32, 64 register size in bits
```

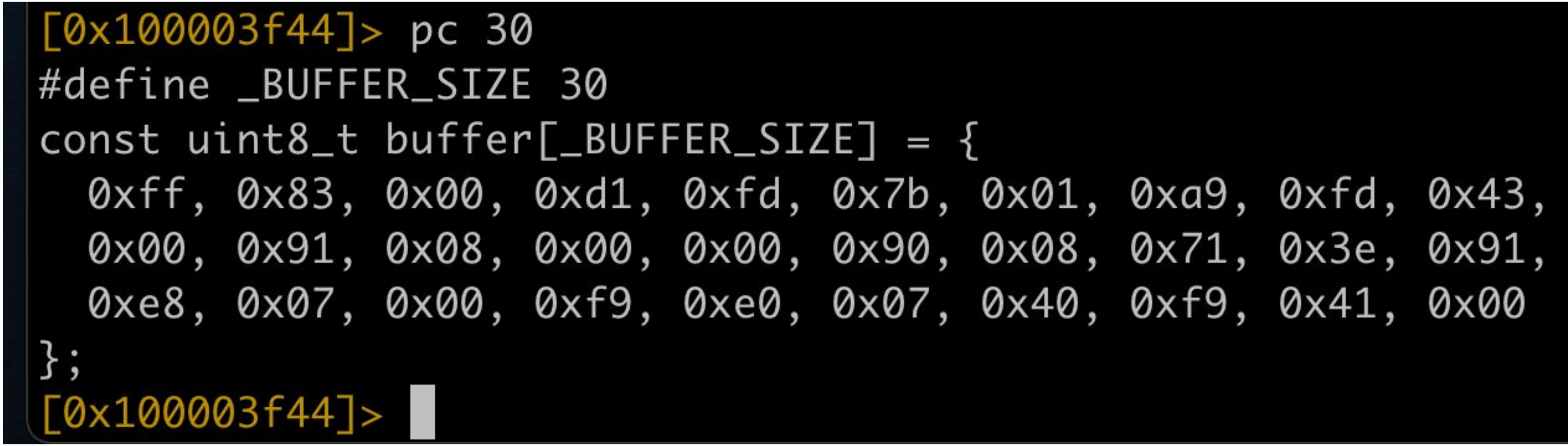
- `e arch.bits=32`
- `e asm.bits=32`
- `e anal.arch=arm`
- `e arch.endian=little`
- `e ~arch`

Printing data

- px # print hexa bytes
- pxw # print list of 32bits
- pxq # print list of 64bits
- ps # print string
- psw # print wide string
- p?
- ps?
- px?

Export bytes to c array, python array, etc.

- `pc          # c array`
- `pcp         # python array`
- `pc?         # view other formats`



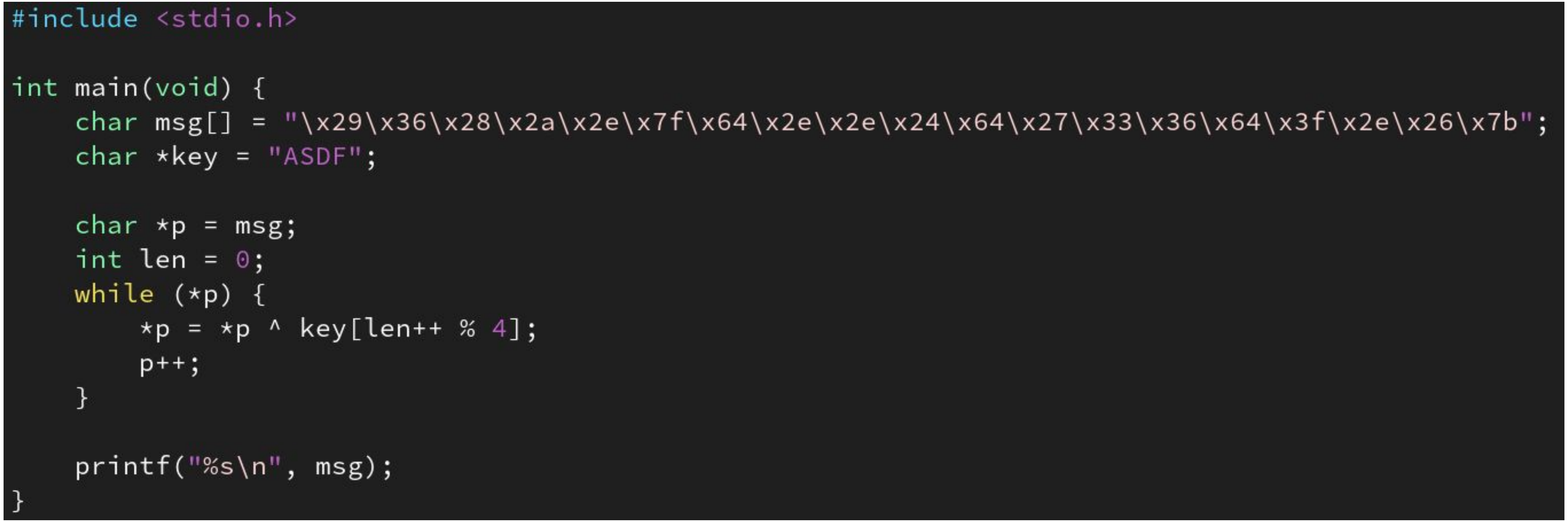
```
[0x100003f44]> pc 30
#define _BUFFER_SIZE 30
const uint8_t buffer[_BUFFER_SIZE] = {
  0xff, 0x83, 0x00, 0xd1, 0xfd, 0x7b, 0x01, 0xa9, 0xfd, 0x43,
  0x00, 0x91, 0x08, 0x00, 0x00, 0x90, 0x08, 0x71, 0x3e, 0x91,
  0xe8, 0x07, 0x00, 0xf9, 0xe0, 0x07, 0x40, 0xf9, 0x41, 0x00
};
[0x100003f44]>
```

Static decryption

- Locate decryption function
- Locate ciphertext
- Locate key
- export bytes to python

- Perform static decryption.

Custom crypto



```c
#include <stdio.h>

int main(void) {
    char msg[] = "\x29\x36\x28\x2a\x2e\x7f\x64\x2e\x2e\x24\x64\x27\x33\x36\x64\x3f\x2e\x26\x7b";
    char *key = "ASDF";

    char *p = msg;
    int len = 0;
    while (*p) {
        *p = *p ^ key[len++ % 4];
        p++;
    }

    printf("%s\n", msg);
}
```

Standard crypto

Symetric

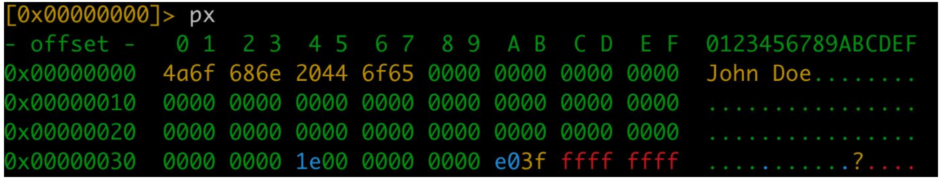
- AES
- DES
- 3DES
- Serpent
- Blowfish
- RC4

Standard crypto

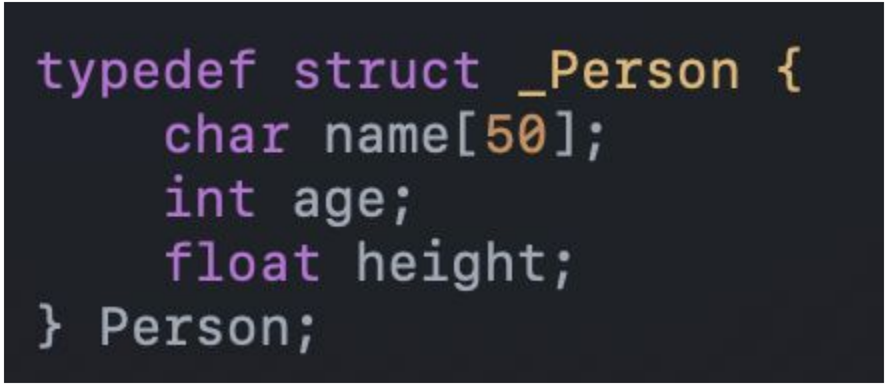
Asymetric

- RSA
- DSA
- ECC
- ElGamal
- Diffie-Hellman

Structure RE



```
[0x00000000]> px
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000000  4a6f 686e 2044 6f65 0000 0000 0000 0000  John Doe........
0x00000010  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00000020  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00000030  0000 0000 1e00 0000 0000 e03f ffff ffff  ...........?....
```



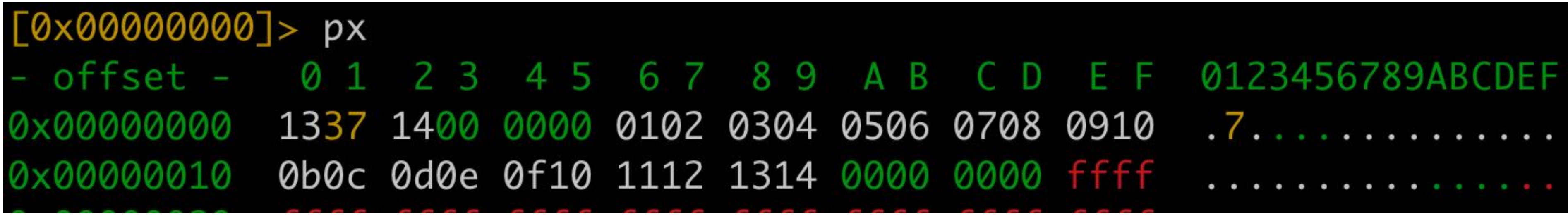
```
typedef struct _Person {
    char name[50];
    int age;
    float height;
} Person;
```

Basic structure reversing

- 0xff # 1 byte 2 nibble
- 0xffff # 16bits or word
- 0x11223344# 32bits or dword
- 0x1122334411223344 # 64bits or qword

Basic structure reversing

- Magic number
- little endian dword Size
- Message
- eof



```
[0x00000000]> px
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000000  1337 1400 0000 0102 0304 0506 0708 0910  .7..............
0x00000010  0b0c 0d0e 0f10 1112 1314 0000 0000 ffff  ................
```

It depends on the point of view

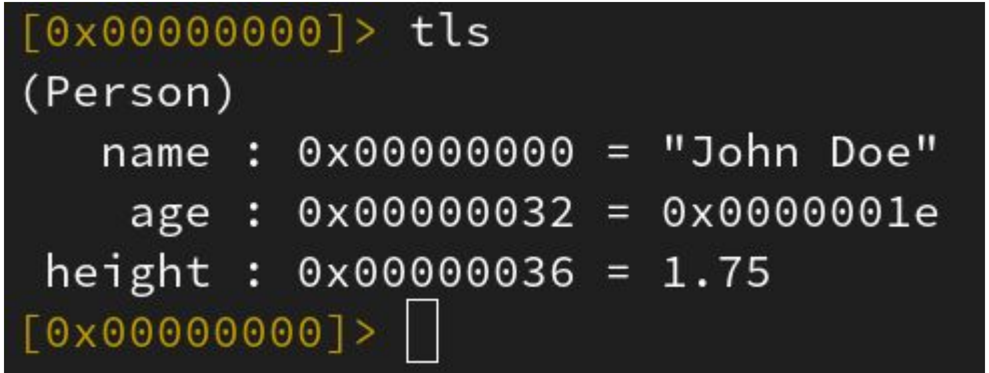
There is only code and data, and code is data.

- `AAAA       # ascii`
- `0x41414141 # hex dword`
- `1094795585 # decimal`
- `inc eax    #  x86 disasm`
- `0b1000001010000010100000101000001 # what it’s really`

Loading structures

Create or use a existing .h

- !cat person.h
- to person.h # load structure
- ts # view loaded structures
- tls # display the structure
- tp



```
[0x00000000]> tls
(Person)
   name : 0x00000000 = "John Doe"
    age : 0x00000032 = 0x0000001e
 height : 0x00000036 = 1.75
[0x00000000]> 
```

Structures

pf - define function signatures

Can load include files with the t command.

010 templates can be loaded using 010 python script.

Load the bin with r2 -nn to load the struct/headers definitions of the target bin file.

Use pxa to visualize them in colorized hexdump.

Shellcode emulation

With r2 esil

Init esil

- aeim
- aei
- aeir rax=0

Stepping:

- aes

Visual emulation:

- V + p + p

Yara patterns

One of those in .radare2rc:

- `(yara x; !yara -msgwr ~/yaras/all.yar `o.`)`
- `(yara x; !yara -$0w ~/yaras/all.yar `o.`)`

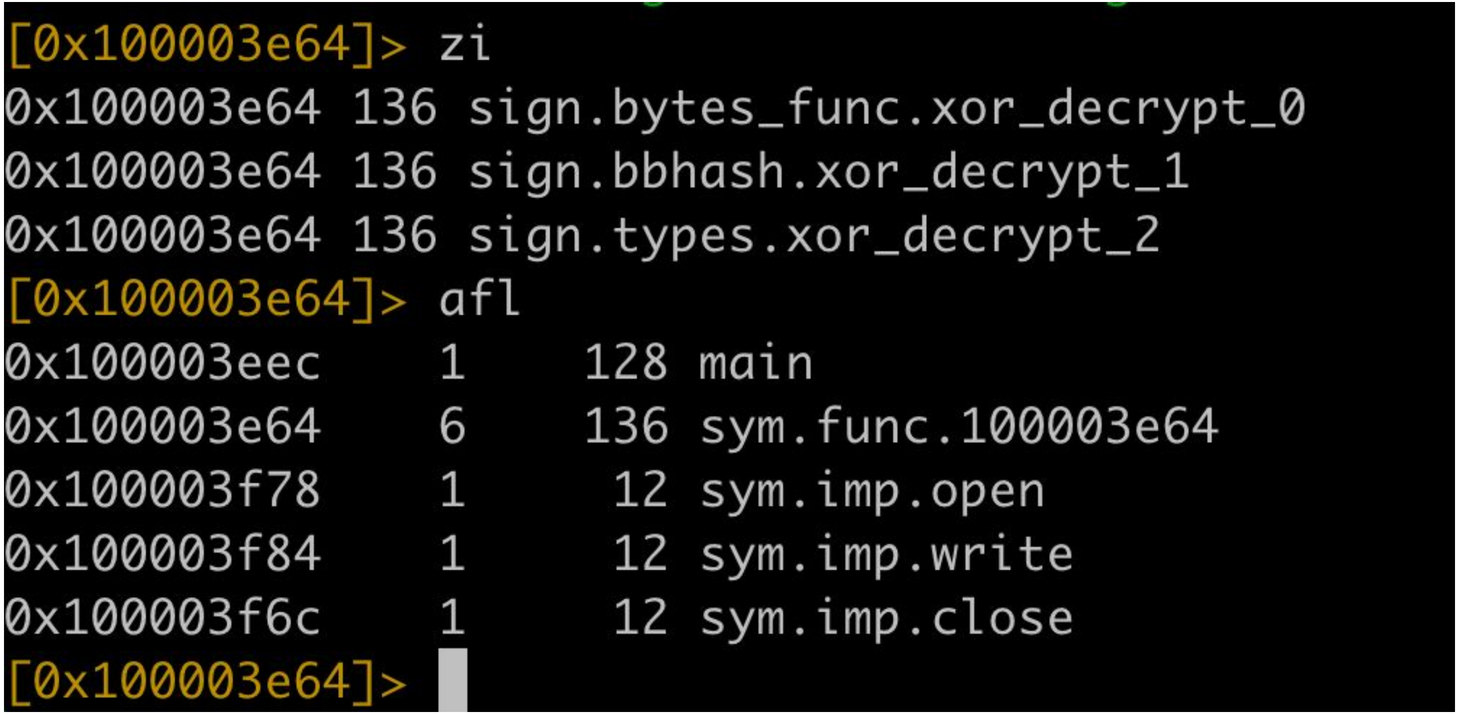
From r2:

- `.(yara m)`

Zignatures

```
zq               # View zignatures
zaf addr name    # create zignature
zos filename     # save zignatures
zo filename      # load zignatures
z.               # scan from current offset
zi               # matches information
```

Zignatures



```
[0x100003e64]> zi
0x100003e64 136 sign.bytes_func.xor_decrypt_0
0x100003e64 136 sign.bbhash.xor_decrypt_1
0x100003e64 136 sign.types.xor_decrypt_2
[0x100003e64]> afl
0x100003eec    1    128 main
0x100003e64    6    136 sym.func.100003e64
0x100003f78    1     12 sym.imp.open
0x100003f84    1     12 sym.imp.write
0x100003f6c    1     12 sym.imp.close
[0x100003e64]> 
```

Save project

- `Ps projectname          #  save project`
- `P projectname           #  load project`

Projects folder:

- `~/.local/share/radare2/projects/`

rasm2

Disassembling and assembling code can be done with pa/pad or using the rasm2 commandline tool.

```
$ rasm2 -a x86 -b 32 nop

$ rasm2 -a x86 -b 64 nop
```

Binary Info

(parsing fileformats)

RBin detects file type and parses the internal structures to provide symbolic and other information.

RBin Information

```
$ rabin2 -s

> is

> fs symbols;f
```

Symbols

Imports

Sections

Relocs

Strings

Libraries

Classes

Demangling

SourceLines

Entrypoints

Exports

ExtraInfo

Scripting

(automation)

The art of automating actions in r2 using your favourite programming language (or not).

Using R2Pipe For Automation

R2 provides a very basic interface to use it based on the cmd() api call which accepts a string with the command and returns the output string.

```
$ pip install r2pipe

r2 = r2pipe.open(‘sample.bin’)

data = r2.cmd(‘pxj’)
```

Analyzing Code

(and graphing)

Analyzing is the “art” of understanding the purpose of a sequence of instructions.

Analyzing From The Metal

R2 provides tools for analyzing code at different levels.

ae - emulates the instruction (microinstructions)

ao - provides information about the current opcode

afb - analyze the basic blocks

af - analyzes the function (or a2f)

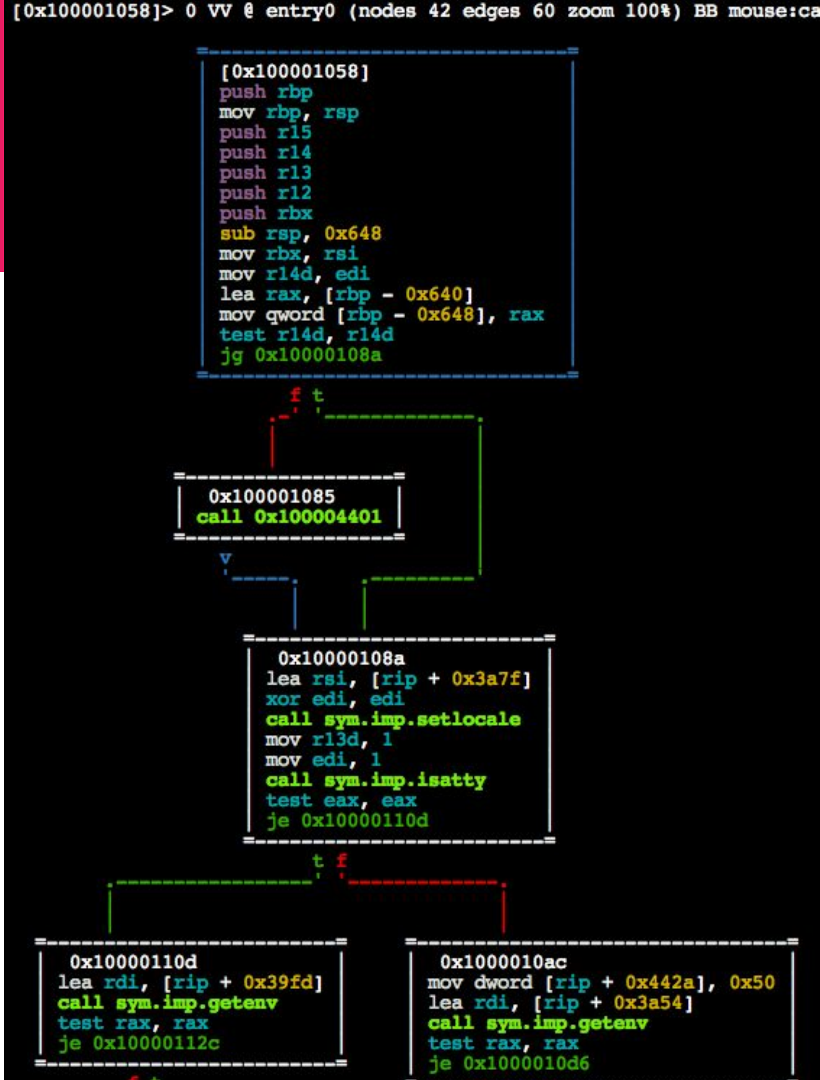
ax - code/data references/calls

Graphing Code

Functions can be rendered as an ascii-art graph using the ‘ag’.

Enter visual mode using the V key

Then press V again to get the graph view.



```
[0x100001058]> 0 VV @ entry0 (nodes 42 edges 60 zoom 100%) BB mouse:ca

[0x100001058]
push rbp
mov rbp, rsp
push r15
push r14
push r13
push r12
push rbx
sub rsp, 0x648
mov rbx, rsi
mov r14d, edi
lea rax, [rbp - 0x640]
mov qword [rbp - 0x648], rax
test r14d, r14d
jg 0x10000108a

f t

 0x100001085
call 0x100004401

v

 0x10000108a
lea rsi, [rip + 0x3a7f]
xor edi, edi
call sym.imp.setlocale
mov r13d, 1
mov edi, 1
call sym.imp.isatty
test eax, eax
je 0x10000110d

t f

 0x10000110d
lea rdi, [rip + 0x39fd]
call sym.imp.getenv
test rax, rax
je 0x10000112c

 0x1000010ac
mov dword [rip + 0x442a], 0x50
lea rdi, [rip + 0x3a54]
call sym.imp.getenv
test rax, rax
je 0x1000010d6
```

BinDiffing

(and graphing)

Finding differences between two binaries looking for bugfixes.

Debugging

(and emulation)

R2 supports native debugger for Linux, BSD, XNU and Windows.

But there’s more!

First Steps

R2 is a low level debugger (not a source debugger).

It provides much more low level information than source debuggers use to provide. Doesn’t competes with GDB/LLDB.

Basic Actions for a debugger are:

ds	step	db	breakpoint	dr	show regs
dso	step over	dcu	continue-until	dx	code-inject
dc	continue	dm	memory-maps	dd	file-desc

Remote Debugging

R2 supports WINDBG, GDB and native remote protocols. But, as long as r2 runs everywhere it is recommended to use it in place.

ESIL

ESIL stands for Evaluable Strings Intermediate Language.

A forth-like language (stack based language) using comma as a tokenizer and used for emulating and analyzing code.

Widely used for decrypting malware routines and analyzing shellcodes and other payloads.

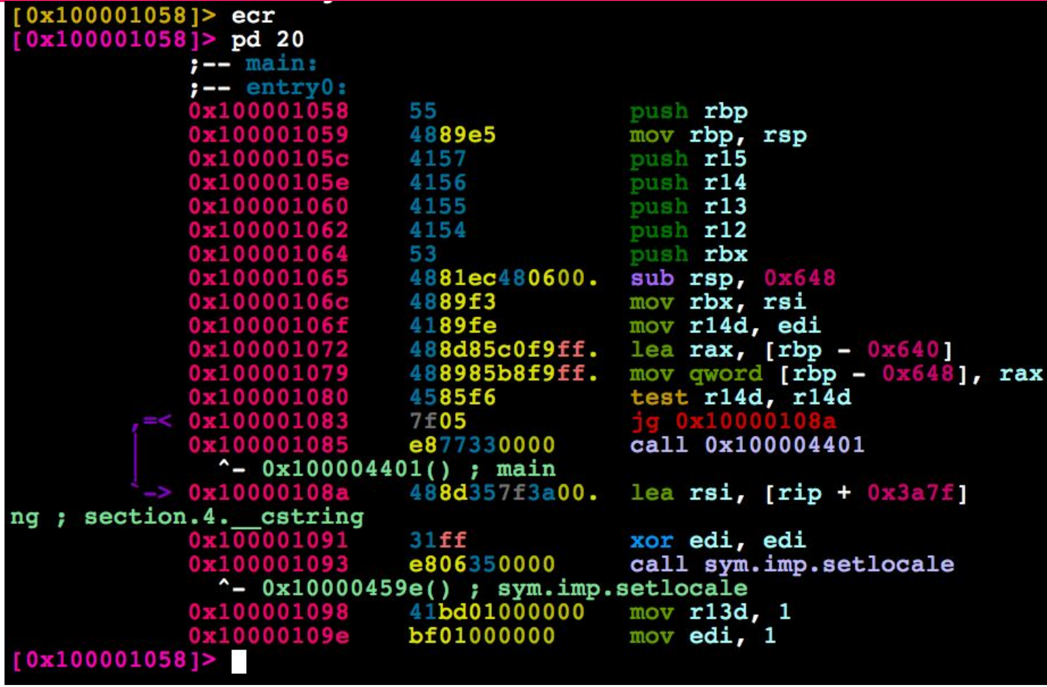
```
mov eax, 33        =>        33,eax,=
```

User Interface

- WebUI
- Bokken
- Iaito
- Visual Mode
- Visual Panels
- Commandline
- R2Pipe
- Colors!

Colors!

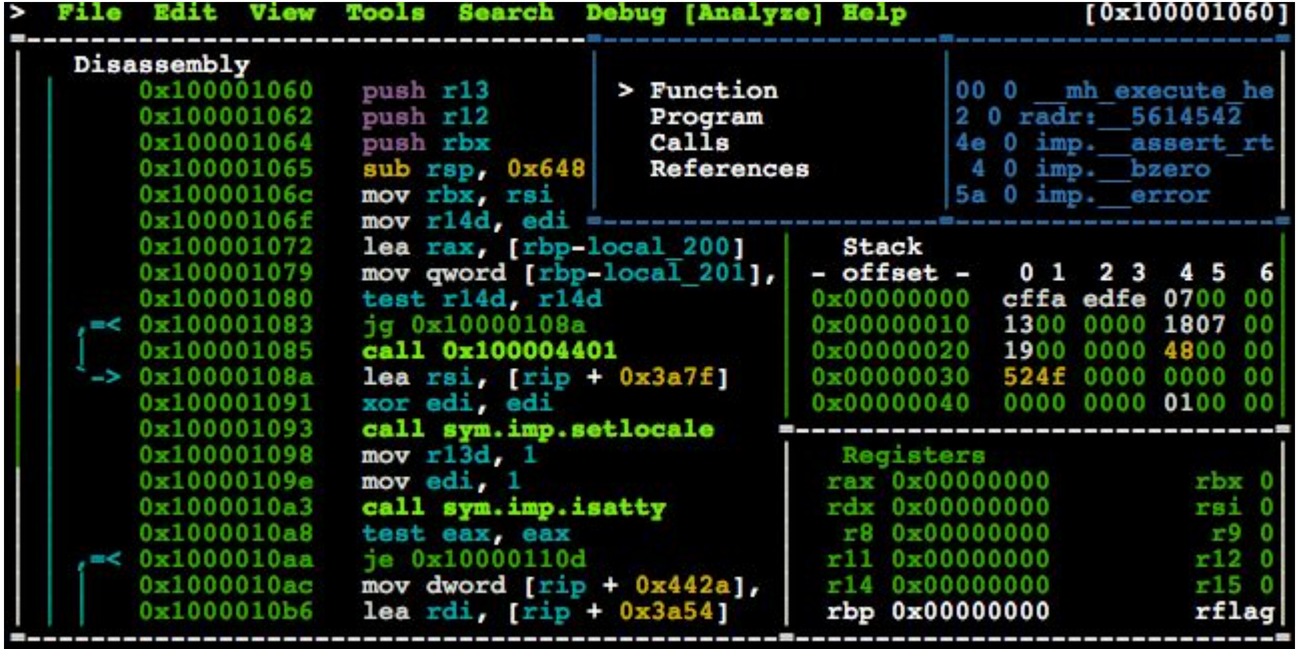
```
> e scr.color=true

> e scr.rgb=true

> e scr.truecolor=true

> e scr.utf8=true


> ecr     # Random colors

> eco X   # Select color palette
```



```
[0x100001058]> ecr
[0x100001058]> pd 20
            ;-- main:
            ;-- entry0:
            0x100001058      55              push rbp
            0x100001059      4889e5          mov rbp, rsp
            0x10000105c      4157            push r15
            0x10000105e      4156            push r14
            0x100001060      4155            push r13
            0x100001062      4154            push r12
            0x100001064      53              push rbx
            0x100001065      4881ec480600.   sub rsp, 0x648
            0x10000106c      4889f3          mov rbx, rsi
            0x10000106f      4189fe          mov r14d, edi
            0x100001072      488d85c0f9ff.   lea rax, [rbp - 0x640]
            0x100001079      488985b8f9ff.   mov qword [rbp - 0x648], rax
            0x100001080      4585f6          test r14d, r14d
        ,=< 0x100001083      7f05            jg 0x10000108a
        |   0x100001085      e877330000      call 0x100004401
        |     ^- 0x100004401() ; main
        `-> 0x10000108a      488d357f3a00.   lea rsi, [rip + 0x3a7f]
ng ; section.4.__cstring
            0x100001091      31ff            xor edi, edi
            0x100001093      e806350000      call sym.imp.setlocale
              ^- 0x10000459e() ; sym.imp.setlocale
            0x100001098      41bd01000000    mov r13d, 1
            0x10000109e      bf01000000      mov edi, 1
[0x100001058]> █
```

Visual Panels

Press ‘!’ in the Visual mode



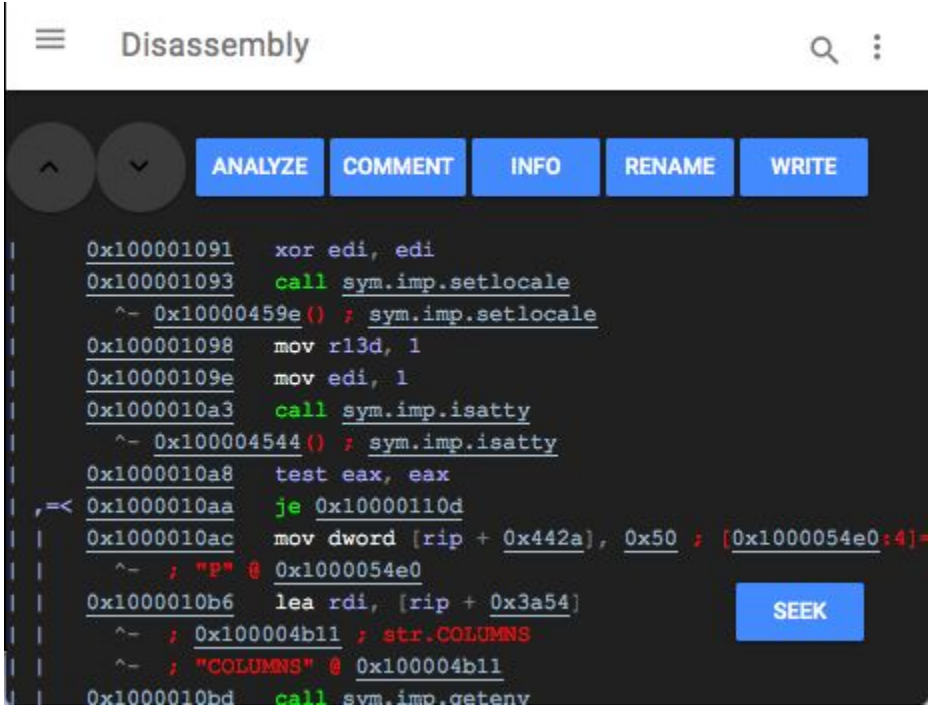
```
> File  Edit  View  Tools  Search  Debug [Analyze] Help              [0x100001060]
-----------------------------------------------------------------------------------
|   Disassembly                                   |                    |
|       0x100001060   push r13                    | > Function         |00 0 __mh_execute_he
|       0x100001062   push r12                    |   Program          |2 0 radr:__5614542
|       0x100001064   push rbx                    |   Calls            |4e 0 imp.__assert_rt
|       0x100001065   sub rsp, 0x648              |   References       | 4 0 imp.__bzero
|       0x10000106c   mov rbx, rsi                |                    |5a 0 imp.__error
|       0x10000106f   mov r14d, edi      -------------------------------------------------
|       0x100001072   lea rax, [rbp-local_200]    |   Stack
|       0x100001079   mov qword [rbp-local_201],  | - offset -    0 1  2 3  4 5  6
|       0x100001080   test r14d, r14d             | 0x00000000  cffa edfe 0700 00
|   ,=< 0x100001083   jg 0x10000108a              | 0x00000010  1300 0000 1807 00
|   |   0x100001085   call 0x100004401            | 0x00000020  1900 0000 4800 00
|   `-> 0x10000108a   lea rsi, [rip + 0x3a7f]     | 0x00000030  524f 0000 0000 00
|       0x100001091   xor edi, edi                | 0x00000040  0000 0000 0100 00
|       0x100001093   call sym.imp.setlocale     ------------------------------------
|       0x100001098   mov r13d, 1                 |   Registers
|       0x10000109e   mov edi, 1                  |  rax 0x00000000          rbx 0
|       0x1000010a3   call sym.imp.isatty         |  rdx 0x00000000          rsi 0
|       0x1000010a8   test eax, eax               |   r8 0x00000000           r9 0
|   ,=< 0x1000010aa   je 0x10000110d              |  r11 0x00000000          r12 0
|   |   0x1000010ac   mov dword [rip + 0x442a],   |  r14 0x00000000          r15 0
|   |   0x1000010b6   lea rdi, [rip + 0x3a54]     |  rbp 0x00000000          rflag
-----------------------------------------------------------------------------------
```

Web User Interface

Start the webserver with =h

Launch the browser with =H

See /m /p /t and /enyo



```
|     0x100001091    xor edi, edi
|     0x100001093    call sym.imp.setlocale
|       ^- 0x10000459e() ; sym.imp.setlocale
|     0x100001098    mov r13d, 1
|     0x10000109e    mov edi, 1
|     0x1000010a3    call sym.imp.isatty
|       ^- 0x100004544() ; sym.imp.isatty
|     0x1000010a8    test eax, eax
| ,=< 0x1000010aa    je 0x10000110d
| |   0x1000010ac    mov dword [rip + 0x442a], 0x50 ; [0x1000054e0:4]=
| |     ^-  ; "P" @ 0x1000054e0
| |   0x1000010b6    lea rdi, [rip + 0x3a54]
| |     ^-  ; 0x100004b11 ; str.COLUMNS
| |     ^-  ; "COLUMNS" @ 0x100004b11
| |   0x1000010bd    call sym.imp.getenv
```

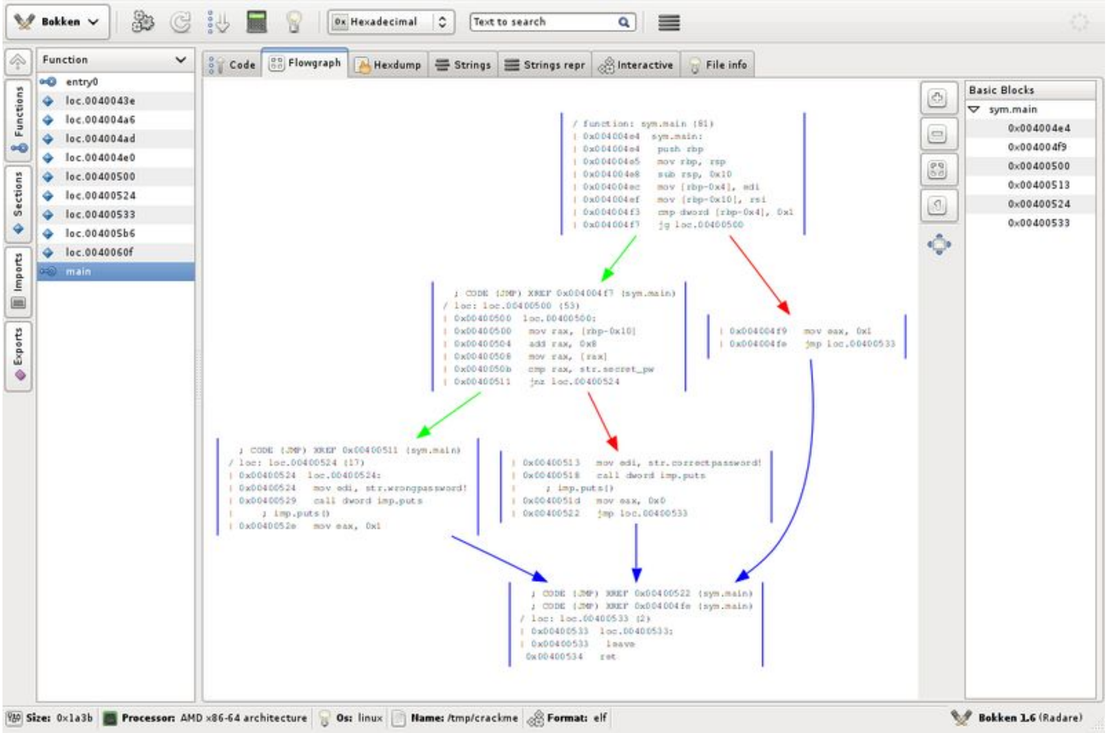
Bokken

Native Python/Gtk GUI

Binaries for Windows

Runs on OSX/Linux too

Author: Hugo Teso



```
/ function: sym.main (81)
| 0x004004e4  sym.main:
| 0x004004e4   push rbp
| 0x004004e5   mov rbp, rsp
| 0x004004e8   sub rsp, 0x10
| 0x004004ec   mov [rbp-0x4], edi
| 0x004004ef   mov [rbp-0x10], rsi
| 0x004004f3   cmp dword [rbp-0x4], 0x1
| 0x004004f7   jg loc.00400500
```

```
  ; CODE (JMP) XREF 0x004004f7 (sym.main)
/ loc: loc.00400500 (53)
| 0x00400500  loc.00400500:
| 0x00400500   mov rax, [rbp-0x10]
| 0x00400504   add rax, 0x8
| 0x00400508   mov rax, [rax]
| 0x0040050b   cmp rax, str.secret_pw
| 0x00400511   jnz loc.00400524
```

```
| 0x004004f9   mov eax, 0x1
| 0x004004fe   jmp loc.00400533
```

```
  ; CODE (JMP) XREF 0x00400511 (sym.main)
/ loc: loc.00400524 (17)
| 0x00400524  loc.00400524:
| 0x00400524   mov edi, str.wrongpassword!
| 0x00400529   call dword imp.puts
|     ; imp.puts()
| 0x0040052e   mov eax, 0x1
```

```
| 0x00400513   mov edi, str.correctpassword!
| 0x00400518   call dword imp.puts
|     ; imp.puts()
| 0x0040051d   mov eax, 0x0
| 0x00400522   jmp loc.00400533
```

```
  ; CODE (JMP) XREF 0x00400522 (sym.main)
  ; CODE (JMP) XREF 0x004004fe (sym.main)
/ loc: loc.00400533 (2)
| 0x00400533  loc.00400533:
| 0x00400533   leave
 0x00400534   ret
```

Size: 0x1a3b **Processor:** AMD x86-64 architecture **Os:** linux **Name:** /tmp/crackme **Format:** elf

Bokken 1.6 (Radare)

Questions?

\o.

Thanks For Watching!