前言

`radare2` 最近越来越流行，已经进入 `github` 前 25了，看到大佬们纷纷推荐，为了紧跟时代潮流，我也决定探究探究这款 `神器`。下面画画重点，以便以后需要用了，可以方便查找。

正文

首先是安装 `radare2` ,直接去官方 `github` 安照指示安装即可。先把源代码下载下来

`https://github.com/radare/radare2`

然后进入源码目录，执行

```
sys/install.sh
```

`radare2` 支持各种各样的平台，文件格式，具体可以看官网描述。它有很多各组件分别进行不同的工作。这些组件是：

- rax2 ---------> 用于数值转换
- rasm2 --------> 反汇编和汇编
- rabin2 --------> 查看文件格式
- radiff2 ------> 对文件进行 diff
- ragg2/ragg2cc ------> 用于更方便的生成shellcode
- rahash2 ------> 各种密码算法， hash算法
- radare2 ------> 整合了上面的工具

rax2

数值转换，程序的 `help` 菜单很明确了：

```
haclh@ubuntu:~$ rax2 -h
Usage: rax2 [options] [expr ...]
  =[base]                 ;  rax2 =10 0x46 -> output in base 10
  int    ->  hex          ;  rax2 10
  hex    ->  int          ;  rax2 0xa
  -int   ->  hex          ;  rax2 -77
  -hex   ->  int          ;  rax2 0xffffffb3
  int    ->  bin          ;  rax2 b30
  int    ->  ternary      ;  rax2 t42
  bin    ->  int          ;  rax2 1010d
  float  ->  hex          ;  rax2 3.33f
  hex    ->  float        ;  rax2 Fx40551ed8
  oct    ->  hex          ;  rax2 35o
  hex    ->  oct          ;  rax2 Ox12 (O is a letter)
  bin    ->  hex          ;  rax2 1100011b
  hex    ->  bin          ;  rax2 Bx63
  hex    ->  ternary      ;  rax2 Tx23
  raw    ->  hex          ;  rax2 -S < /binfile
  hex    ->  raw          ;  rax2 -s 414141
  -l                      ;  append newline to output (for -E/-D/-r/..
  -b     bin -> str       ;  rax2 -b 01000101 01110110
  -B     str -> bin       ;  rax2 -B hello
  -d     force integer    ;  rax2 -d 3 -> 3 instead of 0x3
  -e     swap endianness  ;  rax2 -e 0x33
  -D     base64 decode    ;
  -E     base64 encode    ;
  -f     floating point   ;  rax2 -f 6.3+2.1
  -F     stdin slurp C hex;  rax2 -F < shellcode.c
  -h     help             ;  rax2 -h
  -k     keep base        ;  rax2 -k 33+3 -> 36
  -K     randomart        ;  rax2 -K 0x34 1020304050
  -n     binary number    ;  rax2 -n 0x1234 # 34120000
  -N     binary number    ;  rax2 -N 0x1234 # \x34\x12\x00\x00
  -r     r2 style output  ;  rax2 -r 0x1234
  -s     hexstr -> raw    ;  rax2 -s 43 4a 50
  -S     raw -> hexstr    ;  rax2 -S < /bin/ls > ls.hex
  -t     tstamp -> str    ;  rax2 -t 1234567890
  -x     hash string      ;  rax2 -x linux osx
  -u     units            ;  rax2 -u 389289238 # 317.0M
  -w     signed word      ;  rax2 -w 16 0xffff
  -v     version          ;  rax2 -v
haclh@ubuntu:~$
```

比如输入`rax2 -s 414141`,会返回 `AAAA`

rabin2

对各种文件格式进行解析。

`rabin2 -I hello_pwn` 显示文件的信息

```
haclh@ubuntu:~$ rabin2 -I hello_pwn
arch     x86
binsz    4487
bintype  elf
bits     64
canary   false
class    ELF64
crypto   false
endian   little
havecode true
intrp    /lib64/ld-linux-x86-64.so.2
lang     c
linenum  false
lsyms    false
machine  AMD x86-64 architecture
maxopsz  16
minopsz  1
nx       true
os       linux
pcalign  0
pic      false
relocs   false
relro    partial
rpath    NONE
static   false
stripped true
subsys   linux
va       true
haclh@ubuntu:~$
```

使用 `-l` 显示依赖库。

```
haclh@ubuntu:~$ rabin2 -l hello_pwn
[Linked libraries]
libc.so.6

1 library
haclh@ubuntu:~$
```

使用 `-zz` 显示字符串信息，可以显示 `utf-8` 等宽字节字符串。

```
haclh@ubuntu:~$ rabin2 -zz a.out | grep 分配结点空间
vaddr=0x004005c4 paddr=0x000005c4 ordinal=013 sz=19 len=6 section=.rodata type=utf8 string=分配结点空间 blocks=CJK Unified Ideographs
haclh@ubuntu:~$
```

可以看到显示了长度，所在位置等信息。

通过使用 `-O` 选项可以修改一些文件的信息。

```
haclh@ubuntu:~$ rabin2 -O?
Operation string:
  Change Entrypoint: e/0x8048000
  Dump Symbols: d/s/1024
  Dump Section: d/S/.text
  Resize Section: r/.data/1024
  Remove RPATH: R
```

```
Add Library: a/l/libfoo.dylib
Change Permissions: p/.data/rwx
Show LDID entitlements: C
```

比如修改 `section` 的属性

```
haclh@ubuntu:~$ rabin2 -S a.out | grep text
idx=14 vaddr=0x00400430 paddr=0x00000430 sz=386 vsz=386 perm=--rwx name=.text
haclh@ubuntu:~$ rabin2 -O p/.text/r a.out
wx 02 @ 0x1d60
haclh@ubuntu:~$ rabin2 -S a.out | grep text
idx=14 vaddr=0x00400430 paddr=0x00000430 sz=386 vsz=386 perm=--r-- name=.text
```

rasm2

这个工具用于进行各种平台的汇编和反汇编。该工具的主要选项有。

```
-a 设置汇编和反汇编的架构（比如x86,mips, arm...）
-L  列举支持的架构。
-b  设置 位数
-d, -D  反汇编 提供的 16进制字符串。
```

使用示例:
首先列举支持的架构（使用 `head` 只列举前面几项）

```
haclh@ubuntu:~$ rasm2 -L | head
_dAe  8 16       6502        LGPL3   6502/NES/C64/Tamagotchi/T-1000 CPU
_dA_  8          8051        PD      8051 Intel CPU
_dA_  16 32      arc         GPL3    Argonaut RISC Core
a___  16 32 64   arm.as       LGPL3   as ARM Assembler (use ARM_AS environment)
adAe  16 32 64   arm         BSD     Capstone ARM disassembler
_dA_  16 32 64   arm.gnu     GPL3    Acorn RISC Machine CPU
_d__  16 32      arm.winedbg LGPL2   WineDBG's ARM disassembler
adAe  8 16       avr         GPL     AVR Atmel
adAe  16 32 64   bf          LGPL3   Brainfuck (by pancake, nibble) v4.0.0
_dA_  16         cr16        LGPL3   cr16 disassembly plugin
```

使用 `arm` 插件，汇编 三条 `nop` 指令

```
haclh@ubuntu:~$ rasm2 -a arm "nop;nop;nop;"
0000a0e10000a0e10000a0e1
```

然后我们使用 `-d` 把它反汇编出来

```
haclh@ubuntu:~$ rasm2 -a arm -d 0000a0e10000a0e10000a0e1
mov r0, r0
mov r0, r0
mov r0, r0
```

我可以在命令后面加上 `-r` 打印出在 `radare2`中实现对应的功能，需要使用的命令(`wa` 命令的作用是，汇编给出的指令，并把汇编得到的数据写到相应位置，默认是当前位置)。

```
haclh@ubuntu:~$ rasm2 -a arm -d 0000a0e10000a0e10000a0e1 -r
e asm.arch=arm
e asm.bits=32
"wa mov r0, r0;mov r0, r0;mov r0, r0;"
```

ragg2/ragg2-cc

`radare2` 自己实现的 c 编译器，可以方便的写`shellcode` .
示例一:

代码如下

```
int main() { write (1,"hi\n", 3); exit(0); }
```

使用下面的命令，把它编译成x86 32位代码:

```
ragg2-cc -a x86 -b 32 -d -o test test.c
```

```
haclh@ubuntu:~$ ragg2-cc -a x86 -b 32 -d -o test test.c
==> Compile
gcc -fPIC -fPIE -pie -fpic -m32 -nostdinc -include /usr/include/libr/sflib/linux-x86-32/sflib.h -z execstack -fomit-frame-poin
n-bss -o test.c.tmp -S -Os test.c
gcc -fPIC -fPIE -pie -fpic -m32 -nostdinc -include /usr/include/libr/sflib/linux-x86-32/sflib.h -z execstack -fomit-frame-poin
n-bss -o test.c.tmp -S -Os test.c
============================
==> Assemble
gcc -fPIC -fPIE -pie -fpic -m32 -nostdlib -Os -o test.c.o test.c.s
gcc -fPIC -fPIE -pie -fpic -m32 -nostdlib -Os -o test.c.o test.c.s
/usr/bin/ld: warning: cannot find entry symbol _start; defaulting to 00000000000001d1
==> Link
rabin2 -o 'test.c.text' -O d/S/.text test.c.o
2d
-rw-r--r-- 1 haclh haclh 62 Oct 31 06:22 test.c.text
test
haclh@ubuntu:~$ ./test
hi
haclh@ubuntu:~$ cat test.c
int main() { write (1,"hi\n", 3); exit(0); }
haclh@ubuntu:~$
```

可以生成正常的 `elf` 文件用于测试，可以使用 `-c` 只编译出 `shellcode`

```
haclh@ubuntu:~$ ragg2-cc -a x86 -b 32 -d -c test.c
==> Compile
gcc -fPIC -fPIE -pie -fpic -m32 -nostdinc -include /usr/include/libr/sflib/linux-x86-32/sflib.h -z execstack -fomit-frame-pointer -finline-functions -fno-zero
n-bss -o test.c.tmp -S -Os test.c
gcc -fPIC -fPIE -pie -fpic -m32 -nostdinc -include /usr/include/libr/sflib/linux-x86-32/sflib.h -z execstack -fomit-frame-pointer -finline-functions -fno-zero
n-bss -o test.c.tmp -S -Os test.c
============================
==> Assemble
gcc -fPIC -fPIE -pie -fpic -m32 -nostdlib -Os -o test.c.o test.c.s
gcc -fPIC -fPIE -pie -fpic -m32 -nostdlib -Os -o test.c.o test.c.s
/usr/bin/ld: warning: cannot find entry symbol _start; defaulting to 00000000000001d1
==> Link
rabin2 -o 'test.c.text' -O d/S/.text test.c.o
test.c.text
haclh@ubuntu:~$ r2 test.c.text
 -- Use zoom.byte=entropy and press 'z' in visual mode to zoom out to see the entropy of the whole file
[0x00000000]> pd 30
        ,=< 0x00000000      eb04            jmp 6
        |   0x00000002      68690a00e8      push 0xffffffffe8000a69
            0x00000007      2f              invalid
            0x00000008      0000            add byte [rax], al
            0x0000000a      0005241e0000    add byte [0x00001e34], al
            0x00000010      53              push rbx
            0x00000011      ba03000000      mov edx, 3
            0x00000016      bb01000000      mov ebx, 1
            0x0000001b      8d88d3e1ffff    lea ecx, [rax - 0x1e2d]
            0x00000021      b804000000      mov eax, 4
            0x00000026      53              push rbx
            0x00000027      89db            mov ebx, ebx
            0x00000029      cd80            int 0x80
            0x0000002b      5b              pop rbx
            0x0000002c      31d2            xor edx, edx
            0x0000002e      89d8            mov eax, ebx
            0x00000030      53              push rbx
            0x00000031      89d3            mov ebx, edx
            0x00000033      cd80            int 0x80
            0x00000035      5b              pop rbx
            0x00000036      31c0            xor eax, eax
            0x00000038      5b              pop rbx
            0x00000039      c3              ret
            0x0000003a      8b0424          mov eax, dword [rsp]
            0x0000003d      c3              ret
```

生成的 shellcode 存在于 test.c.text 文件里面，下面是用 radare2 反汇编得到的代码，可以看到使用了 系统调用来实现代码的功能。

使用 ragg2-cc 生成的 shellcode 可以使用 ragg2中的 xor 编码器来编码字符，绕过一些字符限制，比如 \x00。
首先生成shellcode 的16进制表示。

```
ragg2-cc -a x86 -b 32 -d -x test.c
```

```
haclh@ubuntu:~$ ragg2-cc -a x86 -b 32 -d -x test.c
==> Compile
gcc -fPIC -fPIE -pie -fpic -m32 -nostdinc -include /usr/include/libr/sflib/linux-x86-32/sflib.h -z execstack -fomit-frame-pointer -finline-functi
n-bss -o test.c.tmp -S -Os test.c
gcc -fPIC -fPIE -pie -fpic -m32 -nostdinc -include /usr/include/libr/sflib/linux-x86-32/sflib.h -z execstack -fomit-frame-pointer -finline-functi
n-bss -o test.c.tmp -S -Os test.c
============================
==> Assemble
gcc -fPIC -fPIE -pie -fpic -m32 -nostdlib -Os -o test.c.o test.c.s
gcc -fPIC -fPIE -pie -fpic -m32 -nostdlib -Os -o test.c.o test.c.s
/usr/bin/ld: warning: cannot find entry symbol _start; defaulting to 00000000000001d1
==> Link
rabin2 -o 'test.c.text' -O d/S/.text test.c.o
eb0468690a00e82f00000005241e000053ba03000000bb01000008d88d3e1ffffb8040000005389dbcd805b31d289d85389d3cd805b31c05bc38b0424c3
```

然后使用 rasm2 验证下

```
haclh@ubuntu:~$ rasm2 -a x86 -b 32 -d eb0468690a00e82f00000005241e000053ba03000000bb01000008d88d3e1ffffb8040000005389dbcd805b31d289d85389d3cd805b31c05bc38b0424c3
jmp 6
push 0xe8000a69
das
add byte [eax], al
add byte [0x1e24], al
push ebx
mov edx, 3
mov ebx, 1
lea ecx, [eax - 0x1e2d]
mov eax, 4
push ebx
mov ebx, ebx
int 0x80
pop ebx
xor edx, edx
mov eax, ebx
push ebx
mov ebx, edx
int 0x80
pop ebx
xor eax, eax
pop ebx
ret
mov eax, dword [esp]
ret
haclh@ubuntu:~$
```

代码和上面的是一样的。
然后使用 ragg2 使用 xor 编码器编码 shellcode

```
haclh@ubuntu:~$ ragg2 -e xor -c key=32 -B eb0468690a00e82f00000005241e000053ba03000000bb01000008d88d3e1ffffb8040000005389dbcd805b31d289d85389d3cd805b31c05bc38b0424c3
5a3e596a205be8fffffffffc15e4883c60d301e48ffc6e2f9cb2448492a20c80f20202025043e2020739a232020209b21202020ada8f3c1dfdf982420202073a9fbeda07b11f2a9f873a9f3eda07b11e07be3ab2404e3
haclh@ubuntu:~$
```

就是在 shellcode 执行前使用 xor 指令把shellcode 还原。这样就可以消除掉一些坏字符。

ragg2 也有自己 编写 shellcode 的语法。下面是一个示例，具体请看官方文档。

```
haclh@ubuntu:~$ ragg2 -f elf -a x86 -b 32 -o out test.c
haclh@ubuntu:~$ cat test.c
write@syscall(4); //x64 write@syscall(1);
exit@syscall(1); //x64 exit@syscall(60);

main@global(128) {
 .var0 = "hi!\n";
 write(1,.var0, 4);
 exit(0);
}
haclh@ubuntu:~$ ./out
hi!
haclh@ubuntu:~$
```

使用这种方式，我们就能使用最接近 `汇编` 的类c语言 来编写跨平台 `shellcode`

rahash2

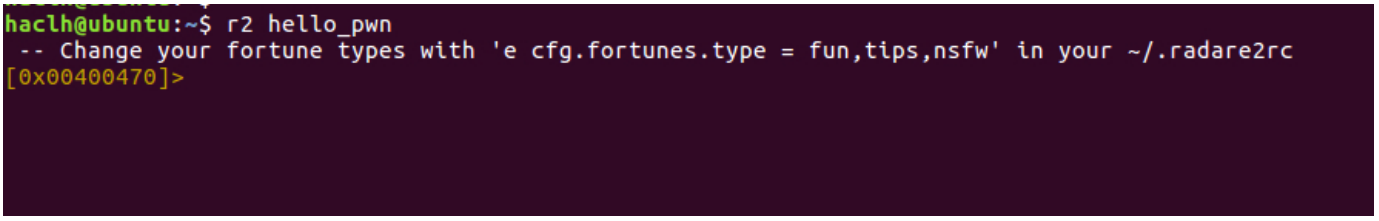
用于使用加密算法，hash算法等计算值

使用`-L` 可以列举支持的算法，比如算算 md5

```
haclh@ubuntu:~$ rahash2 -a md5 -s admin
0x00000000-0x00000004 md5: 21232f297a57a5a743894a0e4a801fc3
```

radare2

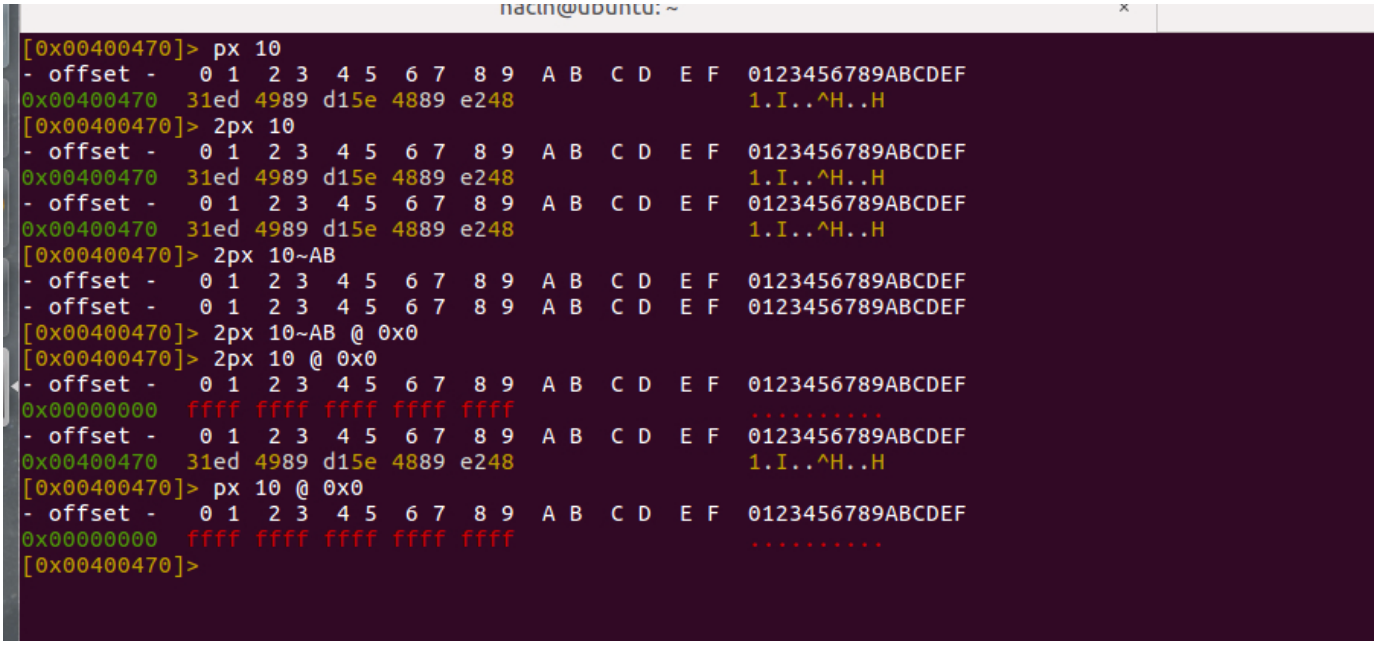
最常用的工具了。整合了上面所有的工具。直接使用 `r2 target_bin` 进入程序。使用`-d` 选项进入调试模式。



`radare2` 中的命令格式为

```
[.][times][cmd][~grep][@[@iter]addr!size][|>pipe] ;
```

`px`表示打印16进制数，默认从当前位置开始。参数控制打印的字节数，下面这张图应该就可以大概解释上面的格式了。



`@ addr` 表示该命令从 `addr` 开始执行。`addr` 不一定是 地址也可以是 `radare2` 中识别的符号，比如 `main`

```
[0x00400470]> px 10 @ main
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00400566  5548 89e5 bf44 0640 00e8                UH...D.@..
[0x00400470]>
```

还有一个重要的东西要记得，在命令的后面加个 `?` ，就可以查看帮助。直接输入`?` 可以查看所有的命令。

```
| pz[?] [len]                print zoom view (see pz? for help)
[0x00400470]> ? | head
Usage: [.][times][cmd][~grep][@[@iter]addr!size][|>pipe] ; ...
Append '?' to any char command to get detailed help
Prefix with number to repeat command N times (f.ex: 3x)
|%var =valueAlias for 'env' command
| *[?] off[=[0x]value]    Pointer read/write data/values (see ?v, wx, wv)
| (macro arg0 arg1)       Manage scripting macros
| .[?] [-|(m)|f|!sh|cmd]  Define macro or load r2, cparse or rlang file
| =[?] [cmd]              Send/Listen for Remote Commands (rap://, http://, <fd>)
| /[?]                    Search for bytes, regexps, patterns, ..
| ![?] [cmd]              Run given command as in system(3)
[0x00400470]> p?
|Usage: p[=68abcdDfiImrstuxz] [arg|len] [@addr]
| p-[?][jh] [mode]              bar|json|histogram blocks (mode: e?search.in)
| p=[?][bep] [blks] [len] [blk]  show entropy/printable chars/chars bars
| p2 [len]                      8x8 2bpp-tiles
| p3 [file]                     print stereogram (3D)
| p6[de] [len]                  base64 decode/encode
| p8[?][j] [len]                8bit hexpair list of bytes
| pa[edD] [arg]                 pa:assemble  pa[dD]:disasm or pae: esil from hexpairs
| pA[n_ops]                     show n_ops address and type
| p[b|B|xb] [len] ([skip])      bindump N bits skipping M
| pb[?] [n]                     bitstream of N bits
| pB[?] [n]                     bitstream of N bytes
| pc[?][p] [len]                output C (or python) format
```

下面按照我们在 `ida` 中使用的功能，来介绍 `radare2`

首先在 用 `ida` 分析程序时，在 `ida` 加载程序后默认会对程序进行分析。`radare2` 相应的功能是以 `a` 开头的。

```
[0x00400470]> a?
|Usage: a[abdefFghoprxstc] [...]
| aa[?]             analyze all (fcns + bbs) (aa0 to avoid sub renaming)
| ab [hexpairs]     analyze bytes
| abb [len]         analyze N basic blocks in [len] (section.size by default)
| ac [cycles]       analyze which op could be executed in [cycles]
| ad[?]             analyze data trampoline (wip)
| ad [from] [to]    analyze data pointers to (from-to)
| ae[?] [expr]      analyze opcode eval expression (see ao)
| af[?]             analyze Functions
| aF                same as above, but using anal.depth=1
| ag[?] [options]   output Graphviz code
| ah[?]             analysis hints (force opcode size, ...)
| ai [addr]         address information (show perms, stack, heap, ...)
| ao[?] [len]       analyze Opcodes (or emulate it)
| aO                Analyze N instructions in M bytes
| ap                find prelude for current offset
| ar[?]             like 'dr' but for the esil vm. (registers)
| as[?] [num]       analyze syscall using dbg.reg
| av[?] [.]         show vtables
| ax[?]             manage refs/xrefs (see also afx?)
Examples:
f ts @ `S*~text:0[3]`; f t @ section..text
f ds @ `S*~data:0[3]`; f d @ section..data
.ad t t+ts @ d:ds
```

注释很简明了。我们使用 `aaa` 就可以进行完整分析了。

```
[0x00400470]> aa?
|Usage: aa[0*?] # see also 'af' and 'afna'
| aa                  alias for 'af@@ sym.*;af@entry0;afva'
| aa*                 analyze all flags starting with sym. (af @@ sym.*)
| aaa[?]              autoname functions after aa (see afna)
| aab                 aab across io.sections.text
| aac [len]           analyze function calls (af @@ `pi len~call[1]`)
| aac* [len]          flag function calls without performing a complete analysis
| aad [len]           analyze data references to code
| aae [len] ([addr])  analyze references with ESIL (optionally to address)
| aaE                 run aef on all functions (same as aef @@f)
| aai[j]              show info of all analysis parameters
| aan                 autoname functions that either start with fcn.* or sym.func.*
| aap                 find and analyze function preludes
| aar[?] [len]        analyze len bytes of instructions for references
| aas [len]           analyze symbols (af @@= `isq~[0]`)
| aat [len]           analyze all consecutive functions in section
| aaT [len]           analyze code after trap-sleds
| aau [len]           list mem areas (larger than len bytes) not covered by functions
| aav [sat]           find values referencing a specific section or map
```

分析前 `radare2` 识别不了函数，分析后就可以正常打印函数代码了（`pdf` 打印函数代码）

```
haclh@ubuntu:~$ r2 hello_pwn
 -- The unix-like reverse engineering framework.
[0x00400470]> pdf
p: Cannot find function at 0x00400470
[0x00400470]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze len bytes of instructions for references (aar)
[x] Analyze function calls (aac)
[x] Use -AA or aaaa to perform additional experimental analysis.
[x] Constructing a function name for fcn.* and sym.func.* functions (aan)
[0x00400470]> paf
Cannot assemble 'f' at line 3
[0x00400470]> pdf
            ;-- section..text:
  (fcn) entry0 41
    entry0 ();
|           0x00400470      31ed           xor ebp, ebp               ; section 14 va=0x00400470 pa=0x00000470 sz=450 vsz=450 rwx=--r-x .text
|           0x00400472      4989d1         mov r9, rdx
|           0x00400475      5e             pop rsi
|           0x00400476      4889e2         mov rdx, rsp
|           0x00400479      4883e4f0       and rsp, 0xfffffffffffffff0
|           0x0040047d      50             push rax
|           0x0040047e      54             push rsp
|           0x0040047f      49c7c0300640.  mov r8, 0x400630
|           0x00400486      48c7c1c00540.  mov rcx, 0x4005c0
|           0x0040048d      48c7c7660540.  mov rdi, main              ; 0x400566
\           0x00400494      e8b7ffffff     call sym.imp.__libc_start_main ; int __libc_start_main(func main, int argc, char **ubp_av, func init, func fini, func rtld_fini
[0x00400470]>
```

有时候我们不需要分析整个 二进制文件，或者有个函数 `radare2`没有识别出来我们可以 `af` 来分析该函数。

```
haclh@ubuntu:~$ r2 hello_pwn
 -- Use 'e asm.offset=true' to show offsets in 16bit segment addressing mode.
[0x00400470]> pdf
p: Cannot find function at 0x00400470
[0x00400470]> af @ 0x00400470
[0x00400470]> pdf
            ;-- entry0:
            ;-- section..text:
/ (fcn) fcn.rip 41
|   fcn.rip ();
|           0x00400470      31ed           xor ebp, ebp               ; section 14 va=0x00400470 pa=0x00000470 sz=450 vs
|           0x00400472      4989d1         mov r9, rdx
|           0x00400475      5e             pop rsi
|           0x00400476      4889e2         mov rdx, rsp
|           0x00400479      4883e4f0       and rsp, 0xfffffffffffffff0
|           0x0040047d      50             push rax
|           0x0040047e      54             push rsp
|           0x0040047f      49c7c0300640.  mov r8, 0x400630
|           0x00400486      48c7c1c00540.  mov rcx, 0x4005c0
|           0x0040048d      48c7c7660540.  mov rdi, 0x400566          ; main
\           0x00400494      e8b7ffffff     call sym.imp.__libc_start_main
[0x00400470]>
```

我们可以使用 `s` 跳转到想要跳转的位置。

跳转到 `main` 函数，并 定义该函数，然后打印函数代码

```
[0x00400470]> s main
[0x00400566]> pdf
p: Cannot find function at 0x00400566
[0x00400566]> af
[0x00400566]> pdf
/ (fcn) main 79
|   main ();
|              ; DATA XREF from 0x0040048d (fcn.rip)
|           0x00400566      55              push rbp
|           0x00400567      4889e5          mov rbp, rsp
|           0x0040056a      bf44064000      mov edi, str.___welcome_to_nuaa_ctf___ ; 0x400644 ; "~~ welcome to nuaa ctf ~~"
|           0x0040056f      e8bcfeffff      call sym.imp.puts           ; int puts(const char *s)
|           0x00400574      bf5e064000      mov edi, str.lets_get_helloworld_for_bof ; 0x40065e ; "lets get helloworld for bof"
|           0x00400579      e8b2feffff      call sym.imp.puts           ; int puts(const char *s)
|           0x0040057e      ba10000000      mov edx, 0x10               ; 16
|           0x00400583      be60106000      mov esi, 0x601060
|           0x00400588      bf00000000      mov edi, 0
|           0x0040058d      b800000000      mov eax, 0
|           0x00400592      e8a9feffff      call sym.imp.read           ; ssize_t read(int fildes, void *buf, size_t nbyte)
|           0x00400597      8b05c70a2000    mov eax, dword [0x00601064] ; [0x601064:4]=0
|           0x0040059d      3d6161756e      cmp eax, 0x6e756161
|       ,=< 0x004005a2      750a            jne 0x4005ae
|       |   0x004005a4      bf40106000      mov edi, str.flag_th1s_Hel10_W0r1d_ ; 0x601040 ; "flag{th1s_Hel10_W0r1d}"
|       |   0x004005a9      e882feffff      call sym.imp.puts           ; int puts(const char *s)
|       `-> 0x004005ae      b800000000      mov eax, 0
|           0x004005b3      5d              pop rbp
\           0x004005b4      c3              ret
[0x00400566]>
```

pd 类命令用于打印汇编信息。

```
[0x00400566]> pd 10
/ (fcn) main 79
|   main ();
|              ; DATA XREF from 0x0040048d (fcn.rip)
|           0x00400566      55              push rbp
|           0x00400567      4889e5          mov rbp, rsp
|           0x0040056a      bf44064000      mov edi, str.___welcome_to_nuaa_ctf___ ; 0x400644 ; "~~ welcome to nuaa ctf ~~"
|           0x0040056f      e8bcfeffff      call sym.imp.puts           ; int puts(const char *s)
|           0x00400574      bf5e064000      mov edi, str.lets_get_helloworld_for_bof ; 0x40065e ; "lets get helloworld for bof"
|           0x00400579      e8b2feffff      call sym.imp.puts           ; int puts(const char *s)
|           0x0040057e      ba10000000      mov edx, 0x10               ; 16
|           0x00400583      be60106000      mov esi, 0x601060
|           0x00400588      bf00000000      mov edi, 0
|           0x0040058d      b800000000      mov eax, 0
[0x00400566]> pdf
/ (fcn) main 79
|   main ();
|              ; DATA XREF from 0x0040048d (fcn.rip)
|           0x00400566      55              push rbp
|           0x00400567      4889e5          mov rbp, rsp
|           0x0040056a      bf44064000      mov edi, str.___welcome_to_nuaa_ctf___ ; 0x400644 ; "~~ welcome to nuaa ctf ~~"
|           0x0040056f      e8bcfeffff      call sym.imp.puts           ; int puts(const char *s)
|           0x00400574      bf5e064000      mov edi, str.lets_get_helloworld_for_bof ; 0x40065e ; "lets get helloworld for bof"
|           0x00400579      e8b2feffff      call sym.imp.puts           ; int puts(const char *s)
|           0x0040057e      ba10000000      mov edx, 0x10               ; 16
|           0x00400583      be60106000      mov esi, 0x601060
|           0x00400588      bf00000000      mov edi, 0
|           0x0040058d      b800000000      mov eax, 0
|           0x00400592      e8a9feffff      call sym.imp.read           ; ssize_t read(int fildes, void *buf, size_t nbyte)
|           0x00400597      8b05c70a2000    mov eax, dword [0x00601064] ; [0x601064:4]=0
|           0x0040059d      3d6161756e      cmp eax, 0x6e756161
|       ,=< 0x004005a2      750a            jne 0x4005ae
|       |   0x004005a4      bf40106000      mov edi, str.flag_th1s_Hel10_W0r1d_ ; 0x601040 ; "flag{th1s_Hel10_W0r1d}"
|       |   0x004005a9      e882feffff      call sym.imp.puts           ; int puts(const char *s)
|       `-> 0x004005ae      b800000000      mov eax, 0
|           0x004005b3      5d              pop rbp
\           0x004005b4      c3              ret
[0x00400566]> pd 5 @ main
/ (fcn) main 79
|   main ();
|              ; DATA XREF from 0x0040048d (fcn.rip)
|           0x00400566      55              push rbp
|           0x00400567      4889e5          mov rbp, rsp
|           0x0040056a      bf44064000      mov edi, str.___welcome_to_nuaa_ctf___ ; 0x400644 ; "~~ welcome to nuaa ctf ~~"
|           0x0040056f      e8bcfeffff      call sym.imp.puts           ; int puts(const char *s)
|           0x00400574      bf5e064000      mov edi, str.lets_get_helloworld_for_bof ; 0x40065e ; "lets get helloworld for bof"
[0x00400566]>
```

具体看帮助。

- 使用 VV 进入 图形化模式（需要是函数范围内）。

- 在图形化模式下，输入？可以查看图形化模式的帮助。

```
Visual Ascii Art graph keybindings:
 :e cmd.gprompt = agft    - show tinygraph in one side
 +/-/0          - zoom in/out/default
 ;              - add comment in current basic block
 .              - center graph to the current node
 :cmd           - run radare command
 '              - toggle graph.comments
 "              - toggle graph.refs
 /              - highlight text
 |              - set cmd.gprompt
 _              - enter hud selector
 >              - show function callgraph (see graph.refs)
 <              - show program callgraph (see graph.refs)
 Home/End       - go to the top/bottom of the canvas
 Page-UP/DOWN   - scroll canvas up/down
 C              - toggle scr.colors
 d              - rename function
 F              - enter flag selector
 g([A-Za-z]*)   - follow jmp/call identified by shortcut (like ;[ga])
 G              - debug trace callgraph (generated with dtc)
 hjkl           - scroll canvas
 HJKL           - move node
 m/M            - change mouse modes
 n/N            - next/previous scr.nkey (function/flag..)
 o              - go/seek to given offset
 p/P            - rotate graph modes (normal, display offsets, minigraph, summary)
 q              - back to Visual mode
 r              - refresh graph
 R              - randomize colors
 s/S            - step / step over
 tab            - select next node
 TAB            - select previous node
 t/f            - follow true/false edges
 u/U            - undo/redo seek
 V              - toggle basicblock / call graphs
 w              - toggle between movements speed 1 and graph.scroll
 x/X            - jump to xref/ref
 y              - toggle node folding/minification
 Y              - toggle tiny graph
 Z              - follow parent node
--press any key--
```

- 使用 `hjkl` 来移动图形
- 使用 `p/P` 切换图形模式
- 在图形模式下使用 `:` 可以输入`radare2` 命令
- 输入 `!`， 在调试的时候应该很有用

- 使用 空格， 切换图形模式和文本模式
- 在文本模式模式下也可以使用 `p` 来切换视图。

剩下的看帮助。

下面介绍如何使用 `radare2` patch程序。首先需要在打开文件时使用 `r2 -w` 来以可写模式打开文件，这样 `pathch` 才能应用到文件（或者在 `radare2` 下使用 `e io.cache=true`, 来允许进行 `patch`, 不过这样的话文件的修改不会影响原文件）

`w` 系列命令用于修改文件。

使用 `wa` 可以使用 汇编指令进行 `patch`

```
[0x00400472]> pd 2
            0x00400472      4989d1          mov r9, rdx
            0x00400475      5e              pop rsi
[0x00400472]> wa nop
Written 1 bytes (nop) = wx 90
[0x00400472]> pd 2
            0x00400472      90              nop
            0x00400473      89d1            mov ecx, edx
[0x00400472]>
```

使用 `"wa nop;nop;nop;nop;"` 可以同时写入多条指令。

双引号不能省

```
Written 1 bytes (nop) = wx 90
[0x00400472]> pd 2
            0x00400472      90              nop
            0x00400473      89d1            mov ecx, edx
[0x00400472]> "wa nop;nop;nop;nop;"
Written 4 bytes (nop;nop;nop;nop;) = wx 90909090
[0x00400472]> pd 4
            0x00400472      90              nop
            0x00400473      90              nop
            0x00400474      90              nop
            0x00400475      90              nop
[0x00400472]>
```

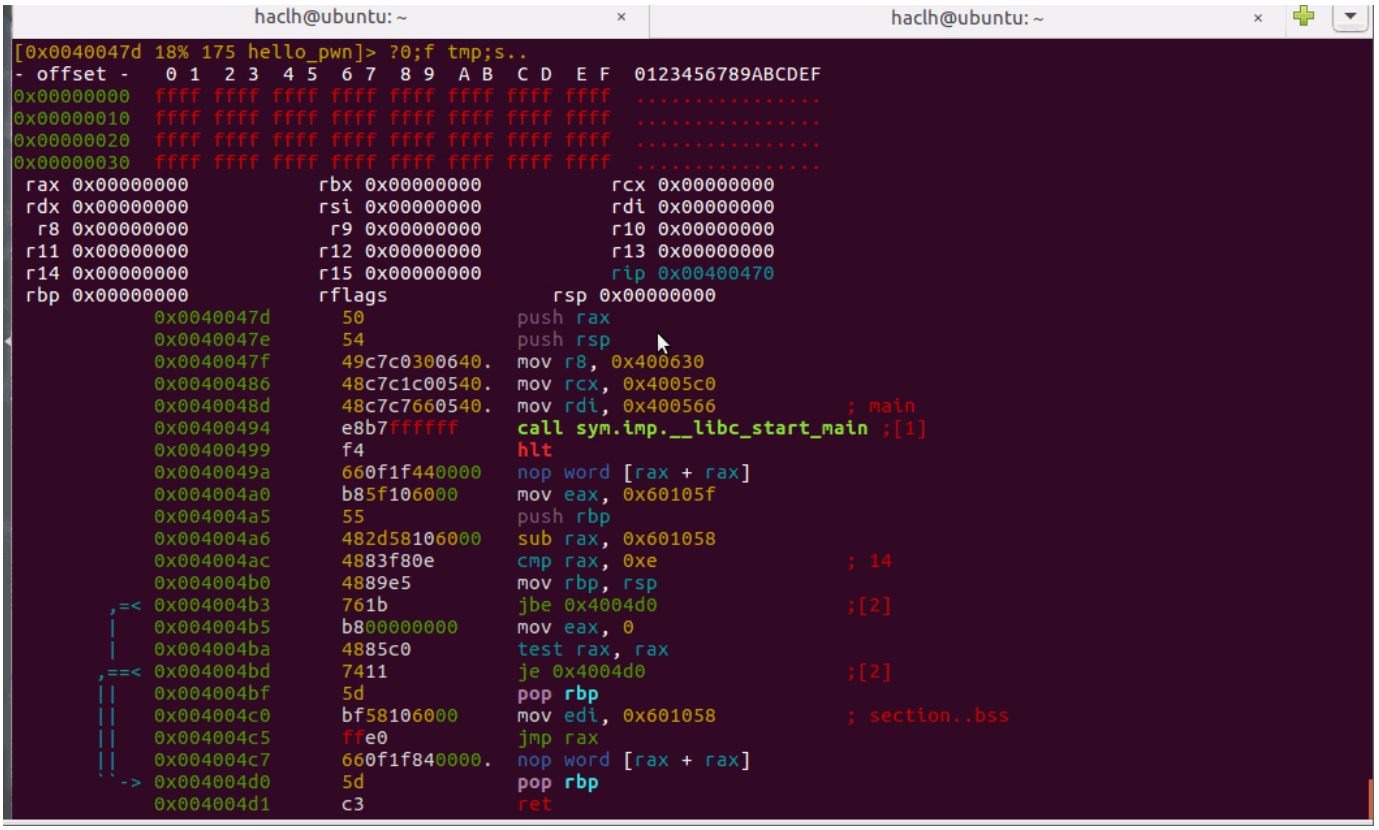
或者可以使用 `wx` 写入 16进制数据

```
[0x00400472]> pd 4
            0x00400472      90              nop
            0x00400473      90              nop
            0x00400474      90              nop
            0x00400475      90              nop
[0x00400472]> wx 00000000
[0x00400472]> pd 4
            0x00400472      0000            add byte [rax], al
            0x00400474      0000            add byte [rax], al
            0x00400476      4889e2          mov rdx, rsp
            0x00400479      4883e4f0        and rsp, 0xfffffffffffffff0
[0x00400472]>
```

其他的请看 `w?` ， 查看帮助。

还可以 `可视化汇编/patch` 程序

输入 `Vp` ， 然后输入 `A`, 就可以了。



使用 / 系列命令可以搜索字符串， rop gadgets等

```
[0x00400476]> /R  pop rsi
  0x00400621                 5e  pop rsi
  0x00400622               415f  pop r15
  0x00400624                 c3  ret

[0x00400476]> / hell
Searching 4 bytes from 0x00000000 to 0xffffffffffffffff: 68 65 6c 6c
Searching 4 bytes in [0x400000-0x4007a4]
hits: 1
Searching 4 bytes in [0x600e10-0x601057]
hits: 0
Searching 4 bytes in [0x601057-0x601068]
hits: 0
0x00400667 hit1_0 .ctf ~~lets get helloworld for bof.
[0x00400476]>
```

查询字符串交叉引用可以依次使用下列方法。

```
[0x00400476]> axt str.lets_get_helloworld_for_bof
[0x00400476]> axf str.lets_get_helloworld_for_bof
[0x00400476]> /r str.lets_get_helloworld_for_bof
[0x00601057-0x00601068] data 0x400574 mov edi, str.lets_get_helloworld_for_bof in unknown function
data 0x400574 mov edi, str.lets_get_helloworld_for_bof in unknown function
data 0x400574 mov edi, str.lets_get_helloworld_for_bof in unknown function
[0x00400476]> aae
[0x00400476]> axf str.lets_get_helloworld_for_bof
[0x00400476]> axt str.lets_get_helloworld_for_bof
data 0x400574 mov edi, str.lets_get_helloworld_for_bof in unknown function
[0x00400476]>
```

`ax?` 系列命令用于管理交叉引用。 `/r` 可以搜索交叉引用, `aae` 是使用`radare2`中的模拟执行功能，动态的检测交叉引用。

下面画重点

radare2中其实也是有 反编译功能，使用 pdc 就可以查看伪代码，虽然和 ida 的还有很大的差距，但是在一些 ida 不支持 f5 的情况下这个功能还是不错的，可以用来看程序的大概逻辑。

```
[0x00400476]> pdc
function fcn.00400476 () {
    loc_0x400476:

       rdx = rsp
       rsp &= 0xfffffffffffffff0

       push rsp
       rax = 0x3e7              //999
       rcx = 0x4005c0
       rdi = 0x400566           //main ; main

       int __libc_start_main(func : unk_format, int argc : 0x00000000 = 4294967295, char ** : unk_format, func : u
nk_format, func : unk_format, func : unk_format, void * stack_end : 0x00000000 = (qword)0xffffffffffffffff)
}
[0x00400476]>
```

在图形化模式下，按下 $ 看看，有惊喜。

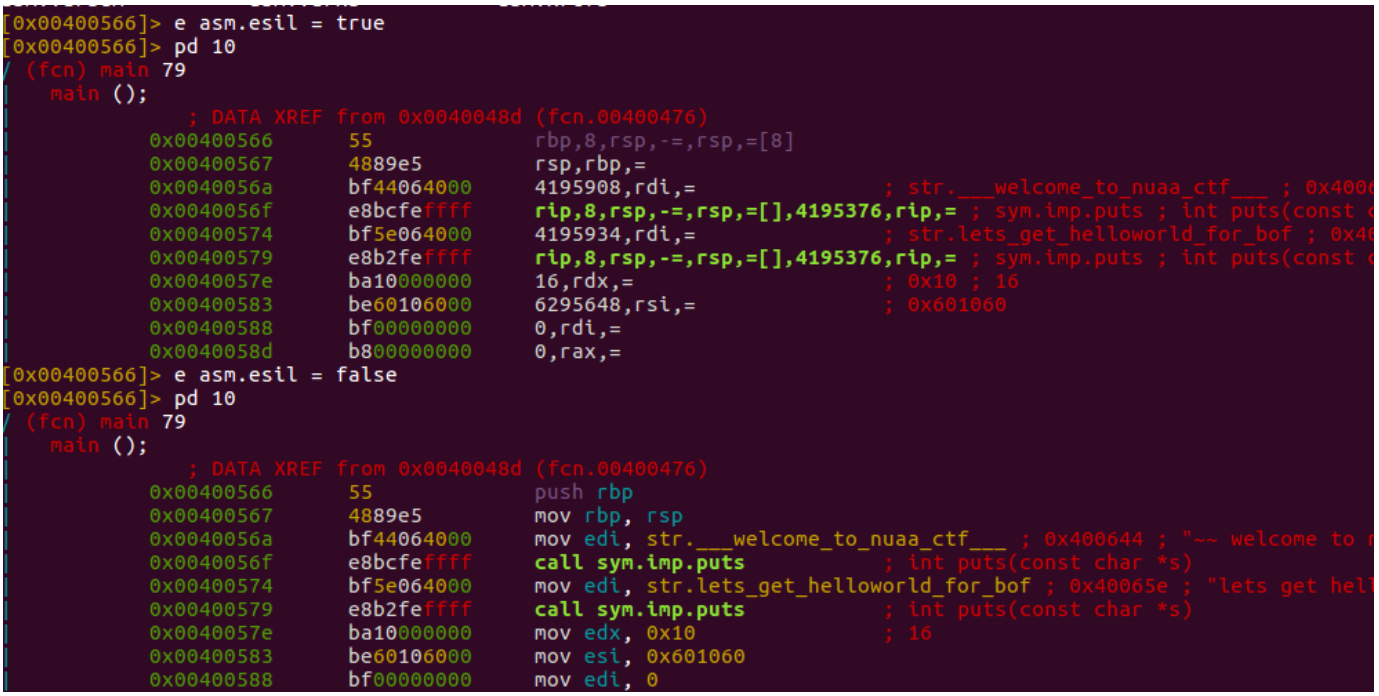
帮我们解析汇编指令，用 c 代码的格式来显示。妈妈再也不用担心我不会汇编了。

radare2和 ida 相比还有一个最大的优势，那就是它自带模拟执行功能。它使用了一种 esil 语言，来定义程序的行为，并且可以根据这个来模拟执行程序代码。

ESIL 的具体语法可以去看官方文档。下面列举两个示例：

```
mov ecx, ebx -> ebx,ecx,=
add ebx, edi ->edi,ebx,+=,$o,of,=,$s,sf,=,$z,zf,=,$c31,cf,=,$p,
```

可以使用 e asm.esil = true显示 esil 代码



和 ESIL相关的命令是

ae?这一类指令。这个我也不熟悉，大概的用法是，使用 ae?这一类指令设置好 指令执行虚拟机的状态，然后设置好模拟执行的终止状态，在停止时，做一些操作，主要用于解密字符串，脱壳等等。

具体事例可以看：

https://blog.xpnsec.com/radare2-using-emulation-to-unpack-metasploit-encoders/

此外 radare2还支持各种语言对他进行调用， 以及拥有大量的插件。

总结

radare2还是很强大的，特别是全平台反编译，全平台模拟执行，各种文件的patch, 修改。感觉在 ida 没法 f5的平台上首选 radare2

参考：

http://radare.org/r/talks.html

https://github.com/radare/radare2book

https://codeload.github.com/radareorg/r2con/

来源： https://www.cnblogs.com/hac425/p/9416727.html