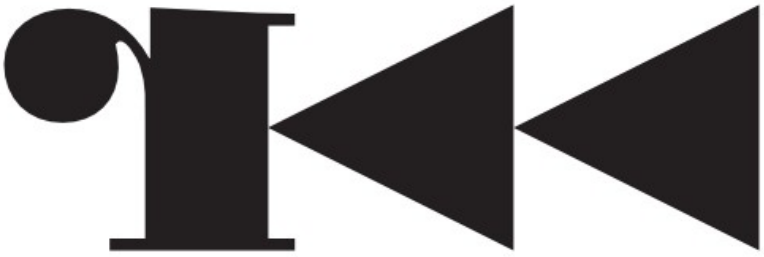


# The Radare2 Book Persian Translation

Revised radare1 book to fit with r2

author : pancake
reviewer : maijin
**translator** : mohsen mostafa jokar

R2 "کتاب"
به کتاب Radare2 خوش آمدید.

# مقدمه

مقدمه
هدف این کتاب پوشش دادن اکثر جنبه‌های radare2 است.چهارچوبی برای مهندسی معکوس و تجزیه و تحلیل فایل‌های باینری.

--pancake

تاریخچه
پروژه Radare در فوریه سال ۲۰۰۶ و با هدف ارائه یک رابط خط دستوری رایگان و ساده برای یک ویرایشگر هگزادسیمال که از آفست های ۶۴ بیتی برای جستجو و بازیابی داده از هارددیسک پشتیبانی کند، آغاز به کار کرد.سپس این پروژه با هدف ارائه یک چهارچوب کامل برای تجزیه و تحلیل فایل‌های باینری با مفاهیم بنیادی *NIX مانند "همه چیز فایل است" ،" برنامه‌های کوچک که با یکدیگر تعامل داشته و از stdin/out استفاده می کنند" یا "آن را ساده نگهدار" رشد کرد.
این تقریباً یک پروژه فردی است اما برخی از کمک‌ها (در منبع، وصله ها، ایده‌ها یا نوع ها) ساخته شده‌اند که از آن‌ها واقعاً قدردانی می کنیم.
این پروژه از یک ویرایشگر هگزادسیمال به عنوان نقطه مرکزی پروژه با اسمبلر/اسمبل کردن مجدد ، تجزیه و تحلیل کد، ویژگی‌های اسکریپت نویسی، تجزیه و تحلیل و نمودار کردن کد و داده ، یکپارچه سازی با یونیکس و... تشکیل شده است.

مرور
هسته ویرایشگر هگزادسیمال و اشکال زدا اجازه می‌دهد تا انواع مختلفی از فایل مانند دسترسی IO به دیسک، شبکه، پلاگین های هسته، دستگاه‌های راه دور، فرایند های اشکل یابی شده ...را باز کرده و به هرکدام از آن‌ها را در صورتی که یک فایل ساده باشند رسیدگی کند.
پیاده‌سازی یک رابط خط دستوری پیشرفته برای انتقال فایل های، تجزیه و تحلیل داده ها، disassembling (ترجمه کردن زبان ماشین به یک زبان اسمبلی)، وصله کردن باینری، مقایسه داده ها، جستجو، جایگزینی، اسکریپت نویسی با زبان‌های Lua، Ruby، Python و Perl و...

Rabin2
استخراج اطلاعات از فایل‌های اجرایی باینری مانند PE، ELF ، کلاس جاوا و MACH-O. این از هسته برای استخراج سمبول ها، وارد کردن، اطلاعات فایل، xrefs، وابستگی کتابخانه، بخش‌ها و... استفاده می کند.

Rasm2
اسمبلر و دی اسمبلر مبتنی بر خط دستور برای معماری های متعدد ( intel[32,64],mips,arm,powerpc,java, msil)

```
$ rasm2 -a java 'nop'
00
$ rasm2 -a x86 -d '90'
nop
$ rasm2 -a x86 -b 32 'mov eax, 33'
b821000000
$ echo 'push eax;nop;nop' | rasm2 -f -
5090
```

rahash2
پیاده‌سازی یک rahash مبتنی بر بلوک برای رشته کوچکی از متن یا دیسک های بزرگ و پشتیبانی از الگوریتم های مختلف مانند sha384, sha256, sha1, crc32, crc16, md5, md4, sha512, par, xor, xorpair, hamdist یا mod255, entropy.
این می‌تواند برای بررسی یکپارچگی یا ردیابی تغییرات بین فایل‌های بزرگ و روگرفت حافظه یا دیسک مورد استفاده قرار بگیرد.

Radiff2
ابزار دودویی diff که الگوریتم های متعدد را پیاده‌سازی می کند.پشتیبانی از تفاوت در سطح بایت یا دلتا برای فایل‌های باینری و تجزیه و تحلیل تفاوت کد برای پیدا کردن تغییرات در بلوک اولیه کد از

تجزیه و تحلیل radare یا آن‌هایی که از اسکریپت idc2rdb rsc با IDA استفاده می کنند.

Rafind2
Rafind2 یک برنامه برای پیدا کردن الگوهای بایت در فایل است.

Ragg2
Ragg2 یک ظاهر برای r_egg است.این برای کامپایل برنامه‌ها به برنامه‌های کوچک باینری برای x86-32/64 و ARM استفاده می شود.

Rarun2
Rarun2 به عنوان یک راه انداز برای اجرای برنامه ها با محیطها، آرگومان ها، مجوزها و دایرکتوری های مختلف و بازنویسی پیش‌فرض های فایل استفاده می‌شود.این می‌تواند برای موارد زیر مفید باشد :

- Crackme
- Fuzzing
- Test suite

دریافت radare2
شما می‌توانید radare را از وب سایت http://radare.org/ یا مخزن Github به آدرس https://github.com/radare/radare2 دریافت کنید.
بسته های باینری برای چندین سیستم عامل و توزیع‌های GNU/Linux وجود دارد (Ubuntu, Maemo,Gentoo, Windows, iPhone و غیره) اما من شما را برای درک بهتر از وابستگی‌ها و داشتن کد منبع و مثال‌های در دسترس، به گرفتن کد منبع و کامپایل آن توسط خودتان تشویق می کنم.
من تلاش می‌کنم که در هر ماه و گاهی اوقات هر شب یک نسخه پایدار جدید را منتشر کنم.
اما همیشه بهترین راه برای استفاده از یک نرم‌افزار حرکت به سمت منبع و گرفتن از مخزن توسعه است که در این مورد radare با ثبات تر از نسخه های "پایدار" است.
برای انجام این کار شما نیاز به Git و تایپ دستور زیر دارید :

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

این احتمالاً مدتی طول خواهد کشید و بنابراین کمی استراحت کنید و سپس خواندن این مقاله را ادامه دهید.
برای به روزرسانی نسخه محلی خود از مخزن، شما نیاز به تایپ دستور زیر در ریشه دایرکتوری 'radare2' که به تازگی ایجاد کرده‌اید دارید :

```
$ git pull
```

در صورتی که شما تغییر محلی از منبع را دارید می‌توانید با استفاده از دستور زیر آن‌ها را برگردانید :

```
$ git reset --hard HEAD
```

یا فقط با یک وصله تغذیه را انجام دهید :

```
$ git diff > radare-foo.patch
```

کامپایل و قابلیت حمل
در حال حاضر هسته radare2 می‌تواند بر روی بسیاری از سیستم‌ها و معماری ها کامپایل شود اما توسعه اصلی بر روی GNU/Linux و GCC انجام می‌شود. کامپایل با TCC و SunStudio نیز شناخته شده است.
مردم اغلب می‌خواهند از radare به عنوان یک اشکال زدا برای مهندسی معکوس استفاده کنند و

این موضوع قابلیت حمل را کمی محدود می‌کند و بنابراین اگر اشکال زدا به پلتفرم مورد علاقه شما منتقل نشده باشد آنوقت لطفاً من را مطلع کرده یا فقط سطح اشکال زدا را با –without-debugger در مرحله ./configure غیرفعال کنید.
امروزه سطح اشکال زدا می‌تواند بر روی
Windows, GNU/Linux (intel32, intel64, mips, arm),FreeBSD,NetBSD,OpenBSD (intel32,intel64)
و طرح هایی برای سولاریس و OSX مورد استفاده قرار بگیرد. و تعدادی پلاگین های IO وجود دارند که از gdb, gdbremote و wine به عنوان طرف دیگر استفاده کنند.
برای ساخت سیستم بر اساس ACR/GMAKE از دستور زیر استفاده کنید :

```
$ ./configure --prefix=/usr
$ gmake
$ sudo gmake install
```

اما یک اسکریپت ساده برای انجام آن به صورت خودکار وجود دارد :

```
$ sys/install.sh
```

کامپایل در ویندوز
راحت ترین راه برای کامپایل چیزها در ویندوز MinGW32 است. W32 ساخته شده و توزیع شده در سایت اصلی radare با استفاده از MinGW32 از یک GNU/Linux تولید شده است و با Wine آزمایش شده است.
برای کامپایل کردن بنویسید :

```
$ sys/mingw32.sh
```

کامپایلر 'i486-mingw32-gcc' تنها چیزی است که من دارم و شما احتمالاً نیاز به تغییر آن دارید. MinGW32 یک برنامه کاربردی تحت کنسول بومی را برای ویندوز تولید می کند.
یکی از راه‌های ممکن دیگر برای کامپایل radare2 در w32 استفاده از Cygwin است که من واقعاً آن را توصیه نمی‌کنم زیرا مشکلات مربوط به کتابخانه Cygwin در بروز مشکلات برنامه را برای اشکال زدایی سخت می کند.

پرچم های خط دستور
هسته به منظور تغییر برخی از پیکربندی ها و یا اجرا با گزینه های مختلف، چندین پرچم را از خط دستور قبول می کند.
در اینجا پیام مربوط به کمک وجود دارد :

```
$ radare2 -h
Usage: r2 [-dDwntLqv] [-P patch] [-p prj] [-a arch] [-b bits] [-i file]
-|file [e k=v-] [c cmd-] [B blocksize-] [s addr-]
Print \x00 after init and every command           0-
a [arch]    set asm.arch-
A           run 'aa' command to analyze all referenced code-
b [bits]    set asm.bits-
B [baddr]   set base address for PIE binaries-
c 'cmd..'   execute radare command-
C           file is host:port (alias for -c+=http://%s/cmd/)-
d           use 'file' as a program to debug-
D [backend] enable debug mode (e cfg.debug=true)-
e k=v       evaluate config var-
f           block size = file size-
h, -hh      show help message, -hh for long-
i [file]    run script file-
k [kernel]  set asm.os variable for asm and anal-
l [lib]     load plugin file-
```

```
L           list supported IO plugins-
m [addr]    map file at given address-
n           do not load file type information-
N           do not load user settings and scripts-
q           quiet mode (no prompt) and quit after -i-
p [prj]     set project file-
P [file]    apply rapatch file and quit-
s [addr]    initial seek-
S           start r2 in sanbox mode-
t           load rabin2 info in thread-
v, -V       show radare2 version (-V show lib versions)-
-w          open file in write mode
```

استفاده اولیه

بسیاری از مردم خواهان یک جلسه ساده از استفاده Radare هستند تا به آن‌ها به درک اینکه چگونه پوسته کار می‌کند و چگونه کارهای رایج مانند دی اسمبل کردن، جستجو، وصله کردن باینری و اشکال زدایی را انجام دهند، کمک کند.

من به شدت شما را تشویق به خواندن بقیه کتاب می‌کنم تا به درک بهتر شما در مورد اینکه چگونه هرچیزی کار می‌کند کمک کرده و مهارت های خود را بهبود ببخشید.منحنی یادگیری برای radare معمولاً در ابتدا کمی شیب دار است.با این حال، بعد از چند ساعت استفاده از آن شما به راحتی درک می‌کنید که بسیاری از چیزها چگونه کار می‌کنند و چگونه ابزارهای مختلف که Radare ارائه می‌دهد را با هم ترکیب کنید :)

مرور یک فایل باینری با استفاده از سه کار ساده انجام می‌شود : جستجو، چاپ و تغییر.

دستور 'seek' به صورت مختصر s است و یک عبارت را به عنوان آرگومان خود قبول می کند.این عبارت می‌تواند چیزی شبیه به 0x25+ , 10 یا [0x100+ptr_table] باشد.اگر شما با فایل‌های مبتنی بر بلوک کار می‌کنید ممکن است ترجیح دهید که اندازه بلوک را به 4K تنظیم کنید یا اندازه مورد نیاز را با b مشخص کرده و با استفاده از دستورهای < و > به جلو یا عقب حرکت کرده و اندازه بلوک را پیدا کنید.

دستور 'print' (مختصر آن : p) یک حرف دوم را برای مشخص کردن حالت چاپ می پذیرد.شایع ترین آن‌ها px برای چاپ در هگزادسیمال و pd برای دی اسمبل هستند.

برای "نوشتن" ابتدا با radare -w فایل را باز کنید.در هنگام باز کردن فایل این باید مشخص شده باشد. سپس شما می‌توانید با استفاده از دستور w رشته و با استفاده از wx یک جفت رشته هگزادسیمال را بنویسید :

```
> w hello world    ;string
> wx 90 90 90     90 ;hexpairs
> wa jmp 0x8048140 ; assemble
> wf inline.bin ; write contents of file
```

اضافه کردن یک ? به دستور نشان دهنده پیام کمک است (به عنوان مثال :p?).

برای ورود به حالت تصویری دکمه V و سپس enter را فشار دهید.برای خروج از حالت تصویری و برگشت به اعلان از دستور q استفاده کنید.

در حالت تصویری شما می‌توانید از کلیدهای جهت نما برای حرک استفاده کنید (به ترتیب چپ، پایین،بالا و راست).شما می‌توانید از این کلیدها در حالت مکان نما(c) نیز استفاده کنید.برای انتخاب کلیدها در حالت مکان نما، در هنگام استفاده از کلیدهای جهت نما به سادگی کلید shift را پایین نگه دارید.

در هنگام استفاده از حالت تصویری شما همچنین می‌توانید عمل درج را با فشار دادن کلید i و سوئیچ بین ستون‌های hex یا string انجام دهید.در پنل hex کلید q را فشار دهید تا به حالت تصویری برگردید.

قالب دستور

قالب کلی برای دستورها چیزی شبیه به زیر است :

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

دستورها توسط یک کاراکتر [a-zA-Z] مشخص می شوند.برای اینکه یک دستور بارها و بارها اجرا شود، به سادگی پیشوند دستور را یک عدد قرار دهید.

```
px # run px
3px # run 3 times 'px'
```

پیشوند ! برای اجرای یک دستور در چهارچوب پوسته استفاده می شود.اگر یک علامت تعجب استفاده شود،دستورات به قلاب system() که در حال حاضر در پلاگین IO بارگذاری شده تعریف شده‌اند فرستاده می شود.به عنوان مثال این در پلاگین ورودی خروجی ptrace استفاده می‌شود که دستورات اشکل زدا را از این رابط قبول می کند.

برخی از مثال‌ها :

```
ds ; call debugger 'step' command
Px 200      @ esp ; show      200   hex   bytes at    esp
Pc > file.c ; dump buffer as a      C byte      array to file
Wx 90 @@ sym.* ; write a nop on every symbol
Pd 2000 | grep eax ; grep opcodes using 'eax' register
Px 20 ; pd 3 ; px 40 ; multiple commands in a single line
```

کاراکتر @ برای مشخص کردن یک آفست موقت که در آن دستور سمت چپ اجرا خواهد شد به کار می رود.
کاراکتر ~ تابع grep داخلی را فعال می‌کند که می‌تواند برای فیلتر کردن خروجی هر دستور به کار رود.استفاده از آن کاملاً ساده است :

```
pd 20~call ; disassemble 20 instructions and grep for 'call'
```

ما می‌توانیم grep را یا برای ستون‌ها و یا سطرها داشته باشیم :

```
pd 20~call:0 ;    get first row
Pd 20~call:1 ;    get second row
Pd 20~call[0] ; get first column
Pd 20~call[1] ; get second column
```

یا حتی ترکیب آن‌ها :

```
pd 20~call[0]:0 ; grep first column of the first row matching 'call'
```

استفاده از تابع grep داخلی یک ویژگی کلیدی برای اسکریپت نویسی Radare است زیرا این می‌تواند برای تکرار لیستی از آفست ها یا داده‌های پردازش شده از دیس اسمبلی، محدوده ها یا دستورات دیگر به کار رود.در اینجا مثالی از استفاده وجود دارد.برای اطلاعات بیشتر بخش ماکروها (تکرار کننده) را ببینید.

عبارات
عبارات یک نمایش ریاضی از یک مقدار ۶۴ بیتی عددی است که می‌تواند در قالب‌های مختلف استفاده شود یا مقایسه شده و یا با تمام دستورها به عنوان یک آرگومان عددی استفاده شود.عبارات از عملیات متعدد اساسی ریاضی و همچنین برخی از باینری ها و منطقی های آن پشتیبانی می کنند.دستوری که برای ارزیابی این عبارات ریاضی استفاده می‌شود ? است.در اینجا برخی از مثال‌ها وجود دارد :

```
[0xB7F9D810]> ? 0X8048000 134512640 0x8048000 01001100000 128.0M 804000:0000
134512640 00000000 134512640.0 0.000 000
[0xB7F9D810]> ? 0X8048000+34 134512674 0x8048022 01001100042 128.0M
804000:0022 134512674 00100010 134512674.0 0.000 000
[0xB7F9D810]> ? 0x8048000+0x34 134512692 0x8048034 01001100064 128.0M
804000:0034 134512692 00110100     134512692.0 0.000 000
[0xB7F9D810]> ? 1+2+3-4*3 -6 0xfffffffffffffffa 0177777777777777777772
17179869183.0G fffff000:0ffa -6
```

عملیات ریاضی پشتیبانی شده عبارت اند از :

```
+  : addition
-  : substraction
*  : multiplication
/  :  division
%  : modulus
>  : shift right
<  : shift left
```

عملیات باینری که باید گفته شوند :

```
\|   : logical OR // ("?     0001010 | 0101001")
\&   : logical AND
```

مقادیر اعداد بیان شده در قالب‌های مختلف هستند :

```
0x033  : hexadecimal
3334   : decimal
sym.fo : resolve flag offset
10K    : Kbytes 10*1024
10M    : Mbytes 10*1024*1024
```

شما همچنین می‌توانید از متغییرها استفاده کرده و به دنبال ایجاد عبـارت هـای پیچیـده‌تر باشـید. در اینجا برخی از مثال‌ها وجود دارند :

```
?@? or      stype @@? ; misc  help  for   '@'   (seek),     '~'   (grep)
      (see  ~??)
?$?  ; show available '$' variables
$$   ; here (current virtual seek)
$l   ; opcode length
$s   ; file size
$j   ; jump address (e.g. jmp 0x10, jz 0x10 => 0x10)
$f   ; jump fail address (e.g. jz 0x10 => next instruction)
$m   ; opcode memory reference (e.g.     mov eax,[0x10] => 0x10)
```

به عنوان مثال :

```
[0x4A13B8C0]> :? $m + $l
140293837812900 0x7f98b45df4a4 03771426427372244 130658.0G 8b45d000:04a4
1402938378129 00 10100100 140293837812900.0    -0.000000


[0x4A13B8C0]> :pd 1 @ +$l 0x4A13B8C2 call 0x4a13c000
```

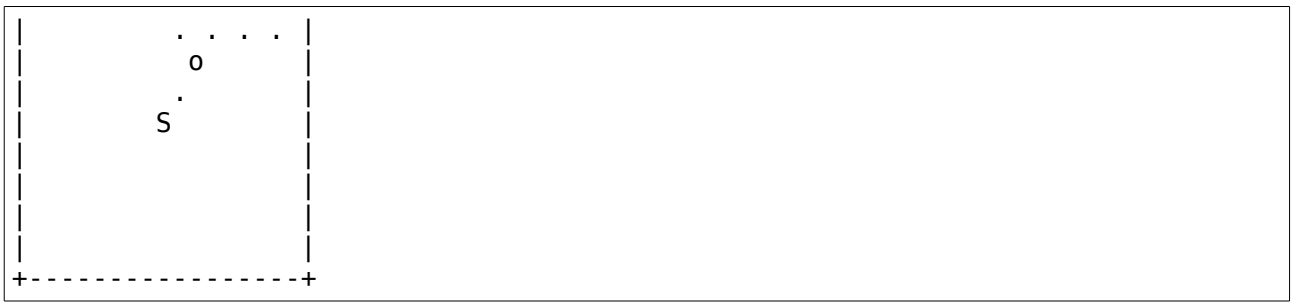
Rax2

ابزار rax2 که همراه با چهارچوب radare است قصد دارد که عبارت حداقلی را برای پوسته ارزیـابی کند. این برای ساخت تغییر بین مقادیر با ممیز شناور، هگزادسیمال، رشته‌های هگز به اسکی، اکتال به عدد صحیح مفید است.این از ترتیب بایت ها در حافظه پشتیبانی کرده و اگر هیـچ آرگومـانی داده نشود می‌تواند به عنوان پوسته استفاده شود.

```
Usage: rax2 [options] [expr ...]
  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
  -b    binstr -> bin     ;  rax2 -b 01000101 01110110
  -B    keep base         ;  rax2 -B 33+3 -> 36
  -d    force integer     ;  rax2 -d 3 -> 3 instead of 0x3
  -e    swap endianness   ;  rax2 -e 0x33
  -f    floating point    ;  rax2 -f 6.3+2.1
  -h    help              ;  rax2 -h
  -k    randomart         ;  rax2 -k 0x34 1020304050
  -n    binary number     ;  rax2 -e 0x1234   # 34120000
  -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
  -v    version           ;  rax2 -V
```

برخی از مثال‌ها :

```
$ rax2      3+0x80
0x83
$ rax2 0x80+3
 131
$ echo 0x80+3 | rax2
131
$ rax2 -s 4142
AB
$ rax2 -S AB
4142
$ rax2 -S < bin.foo
...
$ rax2 -e 33
0x21000000
$ rax2 -e 0x21000000
33
$ rax2 -k 90203010
+--[0x10302090]---+
|            . .Eo|
```



جلسه اساسی اشکال زدا
برای شروع رفع اشکال یک برنامه از پرچم -d استفاده کنید و PID برنامه یا مسیر آن را با آرگومان ها اضافه کنید :

```
$ r2 -d /bin/ls
```

اشکال زدا برنامه ls را در حافظه fork و بارگذاری می‌کند و اجرای آن در ld.so را متوقف می‌کند و بنابراین انتظار نداشته باشید که نقطه ورودی یا کتابخانه نگاشت شده را در این مرحله ببینید.برای تغییر آن شما می‌توانید یک "نقطه شکست" جدید را با اضافه کردن dbg.bep=entry یا dbg.bep=main به radarerc. خود تعریف کنید.
اما مراقب این باشید زیرا بسیاری از ویروس ها و برنامه‌ها می‌توانند قبل از main اجرا شوند.
حالا اعلان اشکال زدا باید ظاهر شود و اگر شما enter را بزنید (دستور تهی) نمایش اولیه از فرایند با روگرفت پشته، ثبات های همه منظوره و دیس اسمبلی از شمارنده برنامه جاری (eip بر روی intel) نمایش داده خواهد شد.

در اینجا لیستی از رایج ترین دستورات برای اشکال زدا وجود دارد :

```
> d?              ; get help on debugger commands
> ds 3            ; step 3 times
> db 0x8048920    ; setup a breakpoint
> db -0x8048920   ; remove a breakpoint
> dc              ; continue process execution
> dcs             ; continue until syscall
> dd              ; manipulate file descriptors
> dm              ; show process maps
> dmp A S rwx     ; change page at A with size S protection permissions
> dr eax=33       ; set register value. Eax = 33
```

راحت ترین راه برای استفاده از اشکال زدا حالت تصویری است.به این ترتیب نیاز نیست که بسیاری از دستورات را در ذهن خود نگه دارید.

```
[0xB7F0C8C0]> V
```

بعد از وارد کردن این دستور یک hexdump از eip جاری نشان داده خواهد شد.حالا p را یک بار فشار دهید تا به اشکال زدا وارد شوید.شما می‌توانید p و P را فشار دهید تا به رایج ترین محیط چاپ چرخش کنید.
از F7 یا s برای وارد شدن و F8 یا S برای خارج شدن استفاده کنید.
با کلید c شما می‌توانید به حالت مکان نما تغییر کنید تا بتوانید محدوده‌ای از بایت ها را تا nop آن‌ها انتخاب کنید یا با استفاده از کلید F2 نقاط شکست را تعیین کنید.
در حالت تصویری شما می‌توانید دستورات را با : وارد کنید تا محتویات بافر را همانند زیر روگرفت کنید

```
x @ esi
```

برای دریافت کمک در محیط تصویری ? را فشار دهید.
در این مرحله رایج ترین دستور !reg است که می‌تواند برای گرفتن یا تنظیم مقادیر ثبات هـای همـه منظوره استفاده شود.شما همچنیـن می‌توانیـد سـخت‌افزار و ثبـات هـای گسـترش یـافته/شـناور را دستکاری کنید.

# پیکربندی

پیکربندی
هسته در هنگام اجرا ~/.radare2rc را می‌خواند و بنابراین شما می‌توانید با دستور e آن را به روش مورد علاقه خود راه اندازی کنید.
برای جلوگیری از تجزیه و تحلیل این فایل از -n استفاده کنید و برای گرفتن یک خروجی تمیز برای استفاده از radare در حالت دسته‌ای ممکن است بهتر باشد که با -v اضافه گویی را حذف کنید.
تمام پیکربندی radare با استفاده از دستور eval انجام می‌شود که به کاربر اجازه تغییر برخی از متغییرها را از یک جدول hash داخلی که شامل رشته‌های جفت جفت است می دهد.
رایج ترین پیکربندی شبیه به زیر است :

```
$ cat ~/.radarerc
e scr.color = true
e dbg.bep  = loader
```

این پیکربندی همچنین می‌تواند با استفاده از پرچم -e در هنگام بارگذاری radare تعریف شود و بنابراین شما می‌توانید پیکربندی های داخلی مختلفی را از خط دستور و بدون نیاز به تغییر فایل rc داشته باشید.

```
$ radare2 -n -e scr.color=true -e asm.syntax=intel -d /bin/ls
```

تمام پیکربندی ها در یک جدول hash که با نام های مختلف (.cfg ,.file ,.dbg ,..) گروه بندی شده است ذخیره می شود.
برای گرفتن یک لیست از متغییرهای پیکربندی فقط e را در اعلان بنویسید.تمام دستورات اساسی می‌توانند به یک کاراکتر کاهش پیدا کنند.شما همچنین می‌توانید یک لیست از متغییرهای پیکربندی را با تمام کردن آرگومان دستور با یک نقطه ارزیابی کنید.
دو رابط پیشرفته برای کمک به کاربران وجود دارد تا به صورت تعاملی این جدول hash را پیکربندی کنند.یکی از آن‌ها emenu نام دارد و یک پوسته را به منظور تغییر متغیرها فراهم می کند.
برای گرفتن کمک در مورد این دستور شما می‌توانید e? را تایپ کنید :

```
Usage: e[?] [var[=value]]
e?              show this help
e?asm.bytes     show description
e??             list config vars with description
e               list config vars
e-              reset config vars
e*              dump config vars in r commands
e!a             invert the boolean value of 'a' var
er [key]        set config key as readonly . no way back
ec [k] [color] set color for given key (prompt, offset,...)
e a             value of var 'a'
e a=b           set var 'a' the 'b' value
env [k[=v]]     get/set environment variable
```

یک رابط e ساده‌تر وجود دارد که از حالت تصویری در دسترس است، بعد از وارد شدن به این حالت بنویسید Ve :

```
eval  spaces:
> anal
asm
scr
asm
bin
cfg
diff
dir
```

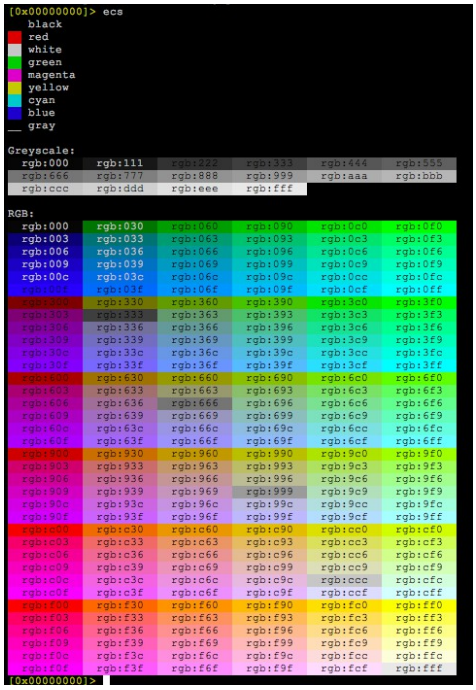
```
dbg
cmd
fs
hex
http
graph
hud
scr
search
io
```

بیشتر درخت‌های eval کاملاً پایدار هستند.
من در اینجا شما را کمی به کار کردن با این رابط تشویق می‌کنم تا نیازهای خود را برآورده کنید.

رنگ‌ها
دسترسی به کنسول توسط یک API پیچیده شده است که اجازه می‌دهد تا خروجی هر دستور به صورت ANSI، کنسول w32 یا HTML (بیشتر ncurses,pango) نشان داده شود.این اجازه می‌دهد تا هسته به اندازه کافی برای اجرای بر روی محیط های محدود مانند هسته یا دستگاه‌های تعبیه شده انعطاف پذیر باشد و به ما اجازه دهد تا بازخورد را از نرم‌افزار در قالب مورد علاقه دریافت کنیم.
برای شروع ما رنگ‌ها را در فایل rc به طور پیش‌فرض فعال می‌کنیم :

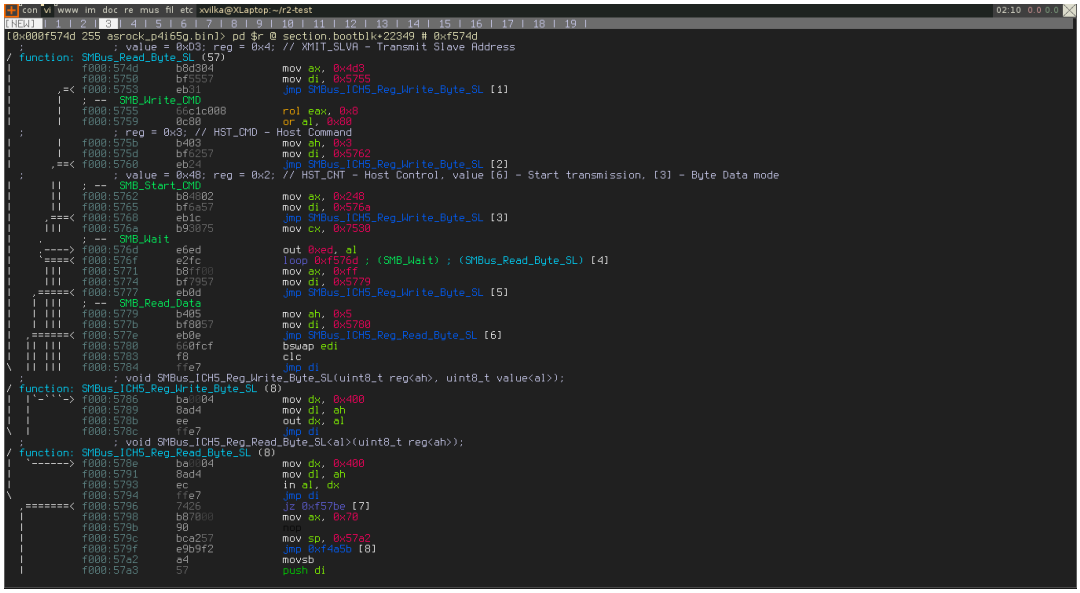
```
$ echo 'e scr.color=true' >> ~/.radare2rc
```

شما می‌توانید رنگ هایی که تقریباً در هر عنصر از دیس اسمبلی استفاده می‌شود را پیکربندی کنید.r2 از رنگ‌های rgb در ترمینال یونیکس پشتیبانی می‌کند و با استفاده از دستور ec اجازه تغییر پالت های رنگ کنسول را می‌دهد.
برای گرفتن یک لیست از تمام عنصرها ec را تایپ کنید.برای نشان دادن یک پالت رنگ به منظور انتخاب رنگ از آن ecs را تایپ کنید :



قالب xvilka :

```
ec fname    rgb:0cf
ec label    rgb:0f3
ec math     rgb:660
ec bin      rgb:f90
ec call     rgb:f00
ec jmp      rgb:03f
ec cjmp     rgb:33c
ec offset  rgb:366
ec comment rgb:0cf
ec push     rgb:0c0
ec pop      rgb:0c0
ec cmp      rgb:060
ec nop      rgb:000
ec b0x00    rgb:444
ec b0x7f    rgb:555
ec b0xff    rgb:666
ec btext    rgb:777
ec other    rgb:bbb
ec num      rgb:f03
ec reg      rgb:6f0
ec fline    rgb:fc0
ec flow     rgb:0f0
```



متغییرهای پیکربندی رایج
در اینجا یک لیست از متغییرهای رایج پیکربندی وجود دارد، شما می‌توانید لیست کامل را با استفاده از دستور e و بدون هیچ آرگومانی یا با استفاده از e cfg. دریافت کنید(با یک نقطه به پایان می‌رسد تا لیست تمام متغییرهای پیکربندی را از فضای cfg. نشان دهد).شما با استفاده از e?? cfg می‌توانید بر روی هر متغییر پیکربندی کمک دریافت کنید.به عنوان مثال :

```
asm.arch
```

معماری مورد استفاده برای دیس اسمبلی توسط دستورهای pd,pD و تجزیه و تحلیل کد توسط دستور e تعریف می شود.در حال حاضر این intel32 , intel64 , mips , arm16 , arm , java , csr , sparc , ppc , msil و m68k را اجرا می کند.
اضافه کردن معماری جدید برای دیس اسمبلی و تجزیه و تحلیل کد کاملاً ساده است و به طوری که

یک رابط اقتباس شده برای دی اسمبلر GNU و بقیه برای udis86 یا آن‌هایی که دست ساز هستند وجود دارد.

```
asm.bits
```

این متغیر asm.arch را یک بار (در radare1) تغییر خواهد داد و برعکس (توسط asm.arch تعیین می شود).این اندازه ثبات ها را برای انتخاب معماری در بیت نشان می‌دهد. این ۶۴،۳۲،۱۶،۸ است :

```
asm.syntax
```

نوع گرامر که در هنگام استفاده از دیس اسمبلی استفاده می‌شود را تعریف کنید.این در حال حاضر تنها دیس اسمبلی udis86 را برای x86 هدف قرار می‌دهد ( ۳۲/۶۴ بیت).مقادیر پشتیبانی شده intel یا att هستند.

```
asm.pseudo
```

مقادیر منطقی مشخص می‌کنند که کدام رشته موتور دیس اسمبلی استفاده شود (یک نوع بومی آن توسط معماری تعریف شده است) یا کدام فیلتر شبه کدهای رشته را نشان دهد.برای مثال، eax=ebx به جای mov eax, ebx.

```
asm.os
```

سیستم عامل هدف مربوط به فایل باینری برای تجزیه و تحلیل را تعریف می کند.این به طور خودکار توسط rabin -rI تعریف می‌شود و این برای سوئیچ کردن بین جدول های مختلف syscall مفید است و بر روی سیستم عامل وابستگی مختلف را انجام می دهد.

```
asm.flags
```

اگر برابر با true تعریف شود آنوقت ستون پرچم ها در دیس اسمبلی را نشان می دهد.

```
asm.linescall
```

به منظور نمایش گرافیکی پرش ها و فراخوانی ها در بلوک جاری، خطوط را در سمت چپ آفست و در قالب چاپی دیس اسمبلی (pd,pD) رسم کنید.

```
asm.linesout
```

هنگامی که به صورت true تعریف شود، آنوقت خطوط پرش را در بلوک جاری که تا خارج از این بلوک ادامه دارد ترسیم می کند.

```
asm.linestyle
```

می‌تواند مقادیر true یا false را بگیرد و اگر false باشد امکان تجزیه و تحلیل خط را از بالا به پایین می‌دهد و اگر true باشد پایین به بالا.false مقدار پیش‌فرض و مطلوب برای خوانایی است.

```
asm.offset
```

مقدار منطقی آدرس آفست یک opcode دیس اسمبلی را نمایش یا مخفی می کند.

```
asm.profile
```

حجم اطلاعاتی که به کاربر در دیس اسمبلی نشان داده می‌شود را تنظیم می کند. و می‌تواند مقادیر default , simple ,gas , smart , debug , full را بگیرد.
این ارزیابی مقادیر دیگر asm را تغییر می‌دهد تا خصوصیت مجازی سازی را برای موتور دیس اسمبلر تغییر دهد.asm.profile simple فقط offset+opcode را نشان می‌دهد و debug اطلاعات در مورد opcodes ردیابی شده، اشاره گر پشته و غیره را نشان می دهد.

```
asm.trace
```

اطلاعات ردیابی را در سمت چپ هر opcode نشان می‌دهد (دنباله عدد و شمارنده).این برای خواندن آثار اجرای یک برنامه مفید است.

```
asm.bytes
```

مقدار منطقی که بایت های دیس اسمبل شده opcode را نشان داده یا مخفی می کند.

```
cfg.bigendian
```

انتخاب مقدار endian که true برای بزرگ و false برای کوچک است.

```
file.analyze
```

در هنگام بارگذاری باینری بعد از حل و فصل سمبول ها .af @@ sym. و .af* @ entrypoint را اجرا می‌کند تا حداکثر اطلاعات در مورد تجزیه و تحلیل کد برنامه را مشخص کند.هنگامی که فایل باز است این استفاده نمی‌شود و بنابراین از قبل بارگذاری شده است.این گزینه نیاز دارد که فایل‌های file.id و file.flag برابر با true باشند.

```
scr.color
```

این متغیر منطقی اجازه فعال یا غیرفعال کردن خروجی رنگی شده را می دهد.

```
scr.seek
```

این متغیر یک عبارت، یک اشاره گر (eg.eip) و غیره را قبول می کند.Radare به طور خودکار تلاش خواهد کرد تا مطمئن شود که این مقدار همیشه در چهارچوب صفحه نمایش است.

```
cfg.fortunes
```

پیام 'fortune' را در ابتدای برنامه فعال یا غیرفعال می کند.

# دستورات اولیه

دستورات اولیه
بیشتر نام دستورات در Radare از نام کاری که انجام می‌دهند مشتق شده است.همانطور که آن‌ها کوتاه هستند برای به خاطر آوردن نیز آسان هستند.بنابراین تمام دستورات تک حرفی هستند.زیر دستورات یا دستورات مرتبط که شرح داده شدند از دو حرف استفاده می کنند.به عنوان مثال، / foo برای جستجوی رشته‌های ساده یا /x9090 برای جستجوی رشته‌های جفت زوجی.
قالب یک دستور معتبر (همانطور که در فصل "قالب دستور" توضیح داده شده است) چیزی شبیه به زیر است :

```
[[.][times][cmd][~grep][@[@iter]addr!size][|>pipe] ;...
> 3s+1024 ;seeks three times 1024 from the current seek
```

اگر دستور با یک ! شروع شود آنوقت رشته به پلاگین IO بارگذاری شده رد می‌شود (به عنوان مثال اشکال زدا).اگر هیچ پلاگینی دستور را به کار نگیرد آنوقت posix_system() فراخوانی می‌شود تا دستور شما را در پوسته تحویل بگیرد.شما همچنین می‌توانید با قرار دادن دو علامت !! به عنوان پیشوند مطمئن شوید که دستور شما به طور مستقیم به پوسته تحویل داده می شود.

```
> !help ; handled by the debugger or shell
> !!ls  ; runs ls in the shell
```

[arg]-part بستگی به دستور خاص دارد.به عنوان یک قاعده کلی، بیشتر دستورها یک عدد را به عنوان آرگومان گرفته تا تعداد بایت ها برای کار کردن را به جای اندازه بلوک مشخص کنند.دستورات دیگر عبارات ریاضی یا رشته‌ها را قبول می کنند.

```
> px 0x17         ; show 0x17 bytes in hexa at current seek
> s base+0x33     ; seeks to flag 'base' plus 0x33
> /lib            ; search for 'lib' string.
```

دستور @ برای مشخص کردن محل یک آفست موقت / جستجو برای اینکه کدام دستور اجرا شده است استفاده می شود.این کاملاً مفید است و بنابراین شما لازم نیست که تمام وقت خود را به جستجو بپردازید.

```
> p8 10 @ 0x4010  ; show 10 bytes at offset 0x4010
> f patata @ 0x10 ; set 'patata' flag at offset 0x10
```

با استفاده از دستور @@ شما می‌توانید یک دستور را بر روی لیستی از پرچم های مطابق اجرا کنید.شما می‌توانید آن را به عنوان یک عملیات foreach در نظر بگیرید :

```
> s 0
> / lib ; search 'lib' string
> p8 20 @@ hit0_* ; show 20 hexpairs at each search hit
```

> برای تغییر مسیر خروجی یک دستور به یک فایل استفاده می شود(اگر در حال حاضر وجود داشته باشد آن به 0 خلاصه می شود).

```
> pr > dump.bin ; dump 'raw' bytes of current block to 'dump.bin' file
> f > flags.txt ; dump flag list to 'flags.txt'
```

| (لوله) رفتاری شبیه به چیزی که شما در پوسته *NIX استفاده می‌کنید دارد : از خروجی یک دستور به عنوان ورودی برای دستور دیگر استفاده کنید.

```
[0x4A13B8C0]> f | grep section | grep text
0x0805f3b0 512 section._text
0x080d24b0 512 section._text_end
```

با استفاده ; شما می‌توانید چندین دستور را در یک خط الحاق کنید :

```
> px ; dr
```

جستجو
جستجو با استفاده از دستور s انجام می شود.این یک عبارت ریاضی را به عنوان آرگومان قبول می‌کند که می‌تواند متشکل از عملیات shift ، عملیات اساسی ریاضی یا عملیات دسترسی به حافظه باشد.

```
[0x00000000]> s?
Usage:      s[+-] [addr]
s           print current     address
S 0x320     seek to this address
s-          undo seek
S+          redo seek
S*          list undo seek history
S++         seek blocksize bytes forward
s--         seek blocksize bytes backward
S+ 512      seek 512 bytes forward
s- 512      seek 512 bytes backward
sg/sG       seek begin(sg) or end (sG) of section or file
s.hexoff    seek honoring a base from core->offset
Sa [[+-]a] [asz] seek asz (or bsize) aligned to addr
sn/sp       seek next/prev scr.nkey
S/ DATA     search for next occurrence of 'DATA'
s/x 9091    search for next occurrence of \x90\x91
Sb          seek aligned to bb start
So [num]    seek to N next opcode(s)
Sf          seek to next function (f->addr+f->size)
SC str      seek to comment matching given string
Sr pc       seek to register

> 3s++ ; 3 times block-seeking
> s 10+0x80 ; seek at 0x80+10
```

اگر می‌خواهید نتیجه یک عبارت ریاضی را بررسی کنید شما می‌توانید با استفاده از دستور ? آن را ارزیابی کنید.به سادگی عبارت را به عنوان آرگومان رد کنید.نتیجه می‌تواند به صورت هگزادسیمال، دسیمال، اکتال یا باینری نمایش داده می شود.

```
> ? 0x100+200 0x1C8 ; 456d ; 710o ; 1100 1000
```

در محیط تصویری شما می‌توانید u (که به معنی undo است) یا U (که به معنی redo است) را فشار داده و جستجو را انجام دهید.

اندازه بلوک
اندازه بلوک نمایش پیش‌فرض اندازه برای Radare است.تمام دستورات با این محدودیت کار خواهند کرد، اما شما همیشه می‌توانید به صورت موقت اندازه بلوکی که فقط یک آرگومان عددی را به دستور print میدهد تغییر دهید. به عنوان مثال :

```
0xB7F9D810]> b?
Usage:      b[f] [arg]
b           display     current     block size
B+3         increase    blocksize   by    3
```

```
b-16     decrement blocksize by 3
b33      set block size to 33
b eip+4  numeric  argument can be an expression
bf foo   set block size to flag size
bm 1M    set max block size
```

دستور b برای تغییر اندازه بلوک استفاده می‌شود :

```
[0x00000000]> b 0x100   ; block size = 0x100
[0x00000000]> b+16      ; … = 0x110
[0x00000000]> b-32      ; … = 0xf0
```

دستور bf برای تغییر اندازه بلوک که توسط یک پرچم مشخص شده است استفاده می شود.به عنوان مثال، در سمبول ها اندازه بلوک پرچم نشان دهنده اندازه تابع است.

```
[0x00000000]> bf sym.main   ; block size = sizeof(sym.main)
[0x00000000]> pd @ sym.main ; disassemble sym.main
...
```

شما می‌توانید این دو عملیات را به صورت یکی(pdf) انجام دهید :

```
[0x00000000]> pdf @ sym.main
```

بخش‌ها
Firmware، بوت لودرها و فایل‌های باینری معمولاً بخش‌های مختلفی از یک باینری را در آدرس‌های مختلف در حافظه بارگذاری می کنند.
برای نشان دادن این رفتار، radare دستور S را ارایه می‌کند.
در اینجا پیام مربوط به کمک وجود دارد :

```
[0xB7EE8810]> S?
Usage:      S[?-.*=adlr] [...]
S               ; list sections
S.              ; show current section name
S?              ; show this help message
S*              ; list sections (in radare commands)
S=              ; list sections (in nice ascii-art bars)
Sa [-] [arch] [bits] [[off]] ; Specify arch and bits for given section
Sd [file]       ; dump current section to a file (see dmd)
Sl  [file]      ; load contents of file into current section (see dml)
Sr [name]       ; rename section on current seek
S [off] [vaddr] [sz] [vsz] [name] [rwx] ; add new section
S-[id|0xoff|*]  ; remove this section definition
```

در این روش شما می‌توانید یک بخش را در یک خط مشخص کنید :

```
S [off] [vaddr]   [sz] [vsz] [name] [rwx] ; add new section
```

به عنوان مثال :

```
[0x00404888]> S 0x00000100 0x00400000 0x0001ae08 0001ae08 test rwx
```

نمایش اطلاعات بخش :

```
[0x00404888]> S ; list sections
```

```
[00].0x00000238 r-- va=0x00400238 sz=0x0000001c vsz=0000001c .interp
[01].0x00000254 r-- va=0x00400254 sz=0x00000020 vsz=00000020 .note.ABI_tag
[02].0x00000274 r-- va=0x00400274 sz=0x00000024 vsz=00000024 .note.gnu.build_id
[03].0x00000298 r-- va=0x00400298 sz=0x00000068 vsz=00000068 .gnu.hash
[04].0x00000300 r-- va=0x00400300 sz=0x00000c18 vsz=00000c18 .dynsym


[0xB7EEA810]> S= ; list sections (in nice ascii-art bars)

...
25 0x0001a600 |-----------------------------#| 0x0001a608 --- .gnu_debuglink
26 0x0001a608 |-----------------------------#| 0x0001a706 --- .shstrtab
27*0x00000000 |##############################| 0x0001ae08 rwx ehdr
=> 0x00004888 |-----^------------------------| 0x00004988
```

سه خط اول بخش‌ها هستند و آخرین خط(که با => شروع می شود) محل جاری جستجو است.
برای حذف کردن تعریف یک بخش به سادگی نام بخش را با - شروع کنید :

```
[0xB7EE8810]> S -.dynsym
```

نگاشت فایل‌ها
IO Radare به شما اجازه نگاشت تقریبی محتویات فایل‌ها را به فضای IO مشابه در هنگامی که باینری را در آفست های تصادفی بارگذاری کرده‌اید، می‌دهد.این برای باز کردن چندین فایل در یک نما یا شبیه سازی یک محیط ایستا همانند چیزی که شما در هنگام استفاده از یک اشکال زدا که برنامه و تمام کتابخانه‌های آن در حافظه بارگذاری شده‌اند و قابل دسترسی هستند مفید است.
با استفاده از دستور S شما قادر به تعریف آدرس‌های پایه مختلف برای هر کتابخانه بارگذاری شده هستید.
نگاشت فایل‌ها با استفاده از دستور o (به معنی open) انجام می‌شود.اجازه دهید که کمک را بخوانیم :

```
[0x00000000]> o?
Usage: o[conm-    ] [file]    ([offset])
o                  list opened files
oc                 [file] open core file, like relaunching r2
oo                 reopen current file (kill+fork in debugger)
oo+                reopen current file in read-write
o 4                priorize io on fd 4 (bring to front)
o-1                close file index 1
o /bin/ls          open /bin/ls file in read-only
o+/bin/ls          open /bin/ls file in read-write mode
o /bin/ls 0x4000   map file at 0x4000
on /bin/ls 0x4000  map raw file at 0x4000 (no r_bin involved)
om[?]              create, list, remove IO maps
```

اجازه دهید یک طرح ساده را آماده سازی کنیم :

```
$ rabin2 -l /bin/ls
[Linked libraries]
libselinux.so.1
librt.so.1
libacl.so.1
libc.so.6
4 libraries
```

یک فایل را نگاشت کنید :

```
[0x00001190]> o /bin/zsh 0x499999
```

فایل‌های نگاشت شده را لیست کنید :

```
[0x00000000]> o
- 6 /bin/ls @ 0x0 ; r
- 10 /lib/ld-linux.so.2 @ 0x100000000 ; r
- 14 /bin/zsh @ 0x499999 ; r
```

برخی  از مقادیر هگزادسیمال را از /bin/zsh چاپ کنید :

```
[0x00000000]> px @ 0x499999
```

برای از حالت نگاشت خارج کردن این فایل‌ها به سادگی از دستور -o استفاده کنید و شاخص فایل را به عنوان آرگومان بدهید :

```
[0x00000000]> o-14
```

3.5 حالت‌های چاپ
یکی از ویژگی‌های کلیدی radare این است که اطلاعات را در قالب‌های مختلف نشان می دهد.هدف ارائه مجموعه‌ای از انتخاب های نمایش است تا داده‌های باینری را به بهترین حالت تفسیر کند.
داده‌های باینری می‌توانند به صورت اعداد صحیح، اعداد کوتاه، اعداد بزرگ، اعداد شناور، برچسب زمانی، رشته‌های هگز یا قالب‌های پیچیده‌تر شبیه ساختارهای C، دیس اسمبلی، دی کامپال، پردازنده های خارجی و... باشند.
در اینجا لیستی از حالت‌های چاپ در دسترس با استفاده از p? لیست شده است :

```
[0x08049AD0]> p?
Usage:      p[=68abcdDfiImrstuxz][arg|len]
p=[bep?] [blks]   show entropy/printable chars/chars bars
p2[len]           8x8 2bpp-tiles
p6[de][len]       base64      decode/encode
p8[len]           8bit  hexpair     list  of    bytes
pa[ed][hex asm]   assemble (pa) or disasm (pad) or esil (pae) from hexpairs
p[bB][len]        bitstream of N bytes
pc[p] [len]       output C (or python) format
p[dD][lf][l]      disassemble N opcodes/bytes (see pd?)
pf[?|.nam][fmt]   print formatted data (pf.name, pf.name $<expr>)
p[iI][df] [len]   print N instructions/bytes (f=func) (see pi? And pdi)
pm[magic]         print libmagic data (pm? For more information)
Pr [len]          print N raw bytes
p[kK] [len]       print key in randomart (K is for mosaic)
ps[pwz] [len]     print pascal/wide/zero-terminated strings
pt[dn?]     [len] print different timestamps
pu[w] [len]       print N url encoded bytes (w=wide)
pv[jh] [mode]     bar|json|histogram blocks (mode: e?search.in)
p[xx][owq] [len]  hexdump of n bytes (o=octal, w=32bit, q=64bit)
pz [len]          print zoom view (see pz? for help)
pwd               display      current working directory
```

۳.۵.۱ هگزا دسیمال
راه کاربر پسند :

```
[0x00404888]>px
```

```
-offset     -0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00404888 31ed 4989 d15e 4889 e248 83e4 f050 5449 1.I..^H..H...PTI
0x00404898 c7c0 4024 4100 48c7 c1b0 2341 0048 c7c7 ..@$A.H...#A.H..
0x004048a8 d028 4000 e83f dcff fff4 6690 662e 0f1f .(@..?....f.f...
```

نشان دادن رونوشت کلمات هگزادسیمال (۳۲ بیت) :

```
[0x00404888]> pxw
0x00404888 0x8949ed31 0x89485ed1 0xe48348e2 0x495450f0 1.I..^H..H...PTI
0x00404898 0x2440c0c7 0xc7480041 0x4123b0c1 0xc7c74800 ..@$A.H...#A.H..
0x004048a8 0x004028d0 0xffdc3fe8 0x9066f4ff 0x1f0f2e66 .(@..?....f.f...
[0x00404888]> e cfg.bigendian
false
[0x00404888]> e cfg.bigendian = true
[0x00404888]> pxw
0x00404888 0x31ed4989 0xd15e4889 0xe24883e4 0xf0505449  1.I..^H..H...PTI
0x00404898 0xc7c04024 0x410048c7 0xc1b02341 0x0048c7c7  ..@$A.H...#A.H..
0x004048a8 0xd0284000 0xe83fdcff 0xfff46690 0x662e0f1f  .(@..?....f.f...
```

یک لیست hexpair هشت بیتی از بایت ها :

```
[0x00404888]> p8 16
31ed4989d15e4889e24883e4f0505449
```

یک روگرفت چهار کلمه‌ای هگزادسیمال (۶۴ بیت) :

```
[0x08049A80]> pxq
0x00001390 0x65625f6b63617473 0x646e6962006e6967     stack_begin.bind
0x000013a0 0x616d6f6474786574 0x7469727766006e69     textdomain.fwrit
0x000013b0 0x6b636f6c6e755f65  0x6d63727473006465    e_unlocked.strcm
...
```

۳.۵.۲ قالب‌های داده
حالت‌های چاپ timestamp (یک دنباله از کاراکترها یا اطلاعات کد شده) که در حال حاضر پشــتیبانی می‌شوند :

```
[0x00404888]> pt?
|Usage: pt[dn?]
| pt                          print unix  time  (32   bit   cfg.big_endian)
| ptd                         print dos   time  (32   bit   cfg.big_endian)
| ptn                         print ntfs  time  (64   bit   !cfg.big_endian)
| pt?                         show  help  message
```

به عنوان مثال شما می‌توانید بافر جاری را به عنوان timestamp در زمان ntfs مشاهده کنید :

```
[0x08048000]> eval cfg.bigendian =  false
[0x08048000]> pt 4 29:04:32948 23:12:36 +0000
[0x08048000]> eval cfg.bigendian = true
[0x08048000]> pt 4
20:05:13001 09:29:21 +0000
```

همانطور که می‌بینید endianness بر روی قالب نمایش تأثیر گذاشته است.در هنگام چاپ کـردن یـک timestamp شما می‌توانید نتایج را به صورت سالانه grep کنید. به عنوان مثال :

```
[0x08048000]> pt | grep 1974 | wc -l
15
[0x08048000]> pt | grep 2022
27:04:2022  16:15:43    +0000
```

قالب پیش‌فرض زمان می‌تواند توسط متغیر cfg.datefmt پیکربندی شود.تعاریف فیلد قـالب شـناخته شده strftime(3) را دنبال می کنند.
گزیده ای از صفحه man مربوط به strftime(3) :

```
%a        The abbreviated name of the day of the week according to the current
locale.
%A       The full name of the day of the week according to the current locale.
%b     The abbreviated month name according to the current locale.
%B     The full month name according to the current locale.
%c     The preferred date and time representation for the current locale.
%C     The century number (year/100) as a 2-digit integer. (SU)
%d     The day of the month as a decimal number (range 01 to 31).
%D     Equivalent to %m/%d/%y . (Yecch—for Americans only. Americans should
note that in other countrie
S %d/%m/%y is rather common. This means that in international context this
format is ambiguous and s hould not be used.) (SU)
%e Like %d, the day of the month as a decimal number, but a leading zero is
replaced by a space.(SU)
%E     Modifier: use alternative format, see below.  (SU)
%F     Equivalent to %Y-%m-%d (the ISO 8601 date format). (C99)
%G     The ISO 8601 week-based year (see NOTES) with century as a decimal
number.     The 4-digit year corresponding to the ISO week number (see %V).This
has the same format and value as%Y, except that if the ISO week number belongs
to the previous or next year, that year is used instead.(TZ)
%g    Like %G, but without century, that is, with a 2-digit year (00-99). (TZ)
%h    Equivalent to %b. (SU)
%H    The hour as a decimal number using a 24-hour clock (range 00 to 23).
%I    The hour as a decimal number using a 12-hour clock (range 01 to 12).
%j    The day of the year as a decimal number (range 001 to 366).
%k    The hour (24-hour clock) as a decimal number (range 0 to 23); single
digits are preceded by a blank. (See also %H.) (TZ)
%l    The hour (12-hour clock) as a decimal number (range 1 to 12); single
digits are preceded by a blank. (See also %I.) (TZ)
%m    The month as a decimal number (range 01 to 12).
%M    The minute as a decimal number (range 00 to 59).
%n      A newline character.(SU)
%O     Modifier: use alternative format, see below. (SU)
%p      Either "AM" or "PM" according to the given time value, or the
corresponding strings for the current    l
ocale.  Noon is treated as "PM" and midnight as "AM".
%P    Like %p but in lowercase: "am" or "pm" or a corresponding string for the
current locale.(GNU)
%r    The time in a.m. or p.m. notation. In the POSIX locale this is equivalent
to %I:%M:%S%p.(SU)
%R    The time in 24-hour notation (%H:%M).(SU) For a version including the
seconds,see %T below.
%s    The number of seconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC).
(TZ)
%S    The second as a decimal number (range 00 to 60). (The range is up to 60
to allow for occasional leap seconds.)
%t    A tab character. (SU)
```

```
%T    The time in 24-hour notation (%H:%M:%S).(SU)
%u    The day of the week as a decimal, range 1 to 7, Monday being 1. See also
      %w.(SU)
%U    The week number of the current year as a decimal number, range 00 to 53,
starting with the first S unday as the first day of week 01. See also %V and
%W.
%V    The ISO 8601 week number (see NOTES) of the current year as a decimal
number,range 01 to 53, where week 1 is the first week that has at least 4 days
in the new year. See also %U and%W.(U)
%w    The day of the week as a decimal, range 0 to 6, Sunday being 0. See also
%u.
%W    The week number of the current year as a decimal number, range 00 to 53,
starting with the first M onday as the first day of week 01.
%x    The preferred date representation for the current locale without the
time.
%X    The preferred time representation for the current locale without the
date.
%y    The year as a decimal number without a century (range 00 to 99).
%Y    The year as a decimal number including the century.
%z    The +hhmm or -hhmm numeric timezone (that is, the hour and minute offset
from UTC).  (SU)
%Z    The timezone name or abbreviation.
%+    The date and time in date(1) format. (TZ) (Not supported in glibc2.)
%%    A literal '%' character.
```

۳.۵.۳ نوع های اولیه

برای تمام نوع های پایه حالت‌های چاپ در دسترس وجـود دارد.اگـر شـما بـه سـاختارهای پیچیـده‌تر علاقه‌مند هستید فقط تایپ کنید : pf?

در اینجا لیستی از حالت‌های چاپ ( pf?) برای نوع های اولیه وجود دارد :

در اینجا لیستی از حالت‌های چاپ (pf?) برای نوع های اولیه وجود دارد :

```
Usage:      pf[.key[.field[=value]]|[     val]]|[times][format]   [arg0 arg1
      ...]
Examples:
            pf    10xiz pointer     length      string
            pf    {array_size}b     @     array_base
            pf.
            #     list  all   formats
            pf.obj      xxdz  prev  next  size  name
            pf.obj                                                        #
      run   stored      format
            pf.obj.name                         #     show  string      inside
      object
            pf.obj.size=33          #     set   new   size
Format      chars:
e     -     temporally  swap  endian
f     -     float value (4    bytes)
c     -     char  (signed     byte)
b     -     byte  (unsigned)
B     -     show  10    first bytes of    buffer
i     -     %i    integer     value (4    bytes)
w     -     word  (2    bytes unsigned    short in    hex)
q     -     quadword    (8    bytes)
p     -     pointer     reference   (2,   4     or    8     bytes)
d     -     0x%08x      hexadecimal value (4    bytes)
D     -     disassemble one   opcode
x     -     0x%08x      hexadecimal value and   flag  (fd   @     addr)
z     -     \0    terminated  string
```

```
Z     -     \0    terminated  wide  string
s     -     32bit pointer     to    string      (4    bytes)
S     -     64bit pointer     to    string      (8    bytes)
*     -     next  char  is    pointer     (honors     asm.bits)
+     -     toggle      show  flags for   each  offset
:     -     skip  4     bytes
.     -     skip  1     byte
```

اجازه دهید به چند مثال نگاه کنیم :

```
[0x4A13B8C0]> pf i
0x00404888  = 837634441
[0x4A13B8C0]> pf
0x00404888  = 837634432.000000
```

### ۳.۵.۴ منبع (C ، ASM)

قالب‌های معتبر چاپ کد عبارت اند از : JSON, C,Python,Cstring(pcj, pc,pcp, pcs)pc C pcs string pcj json pcJ

pcw words (4 byte) pcd dwords (8 byte) python javascriptpcp.

```
[0xB7F8E810]> pc 32
#define_BUFFER_SIZE 32
Unsigned char buffer[_BUFFER_SIZE] = {
0x89, 0xe0, 0xe8, 0x49, 0x02, 0x00, 0x00, 0x89, 0xc7, 0xe8, 0xe2, 0xff, 0xff,
      0xff, 0x81, 0xc3, 0xd6, 0xa7,
0x01, 0x00, 0x8b, 0x83, 0x00, 0xff, 0xff, 0xff, 0x5a, 0x8d, 0x24, 0x84, 0x29,
      0xc2  };
[0x7fcd6a891630]> pcs
"\x48\x89\xe7\xe8\x68\x39\x00\x00\x49\x89\xc4\x8b\x05\xef\x16\x22\x00\x5a\x48\x
8d\x24\xc4\x29\xc2\x52\
x48\x89\xd6\x49\x89\xe5\x48\x83\xe4\xf0\x48\x8b\x3d\x06\x1a
```

### ۳.۵.۵ رشته

در هنگام انجام مهندسی معکوس یک برنامه رشته‌ها معمولاً یکی از مهمترین نقاط ورودی هستند زیرا آن‌ها معمولاً مرجع اطلاعات در مورد کارهای توابع هستند (,asserts debug یا info messages).
بنابراین Radare از قالب‌های مختلف رشته پشتیبانی می‌کند :

```
[0x00404888]> ps?
|Usage: ps[zpw] [N]
|     ps    =     print string
|     psb   =     print strings     in    current     block
|     psx   =     show  string      with  scaped      chars
|     psz   =     print zero  terminated  string
|     psp   =     print pascal      string
|     psw   =     print wide  string
```

اغلب رشته‌ها خاتمه یافته با صفر خواهند بود.در اینجا یک مثال وجود دارد که با استفاده از اشکال زدا اجرای برنامه ادامه پیدا کرده تا وقتی که "syscallopen اجرا شود.هنگامی که ما کنترل روند را بازیابی می کنیم، ما آرگومان هایی که به syscall توسط %ebx رد شده است را دریافت می کنیم.در مورد فراخوانی 'open' این پارامتر یک رشته خاتمه یافته با صفر است که می‌تواند با استفاده از psz بازرسی شود.

```
[0x4A13B8C0]> dcs open
0x4a14fc24 syscall(5)  open ( 0x4a151c91 0x00000000 0x00000000 ) = 0xfffffda
```

```
[0x4A13B8C0]> dr
eax  0xffffffda  esi   0xffffffff    eip 0x4a14fc24
ebx  0x4a151c91 edi    0x4a151be1  oeax  0x00000005
ecx 0x00000000  esp    0xbfbedb1c  eflags      0x200246
edx 0x00000000  ebp    0xbfbedbb0  cPaZstIdor0 (PZI)
[0x4A13B8C0]>
[0x4A13B8C0]> psz @ 0x4a151c91
/etc/ld.so.cache
```

### 3.5.6 چاپ حافظه

همچنین با استفاده از دستور pf امکان چاپ کردن انواع داده‌های بسته بندی شده وجود دارد.

```
[0xB7F08810]> pf xxS @ rsp
0x7fff0d29da30 = 0x00000001
0x7fff0d29da34 = 0x00000000
0x7fff0d29da38 = 0x7fff0d29da38 -> 0x0d29f7ee  /bin/ls
```

به عنوان مثال این می‌تواند برای نگاه کردن به آرگومان های رد شده به یک تابع مورد استفاده قرار بگیرد.برای رسیدن به این، به سادگی یک "قالب رشته حافظه" را به عنوان آرگومان بـه pf رد کنیـد و با استفاده از @ به صورت موقت موقعیت فعلی جستجو/آفست را تغییر دهید.
همچنین این امکان وجود دارد که با استفاده از pf آرایه ای از ساختارها را تعریف کنیـد.بـرای انجـام این کار، پیشوند رشته را به مقدار عددی قرار دهید.
شما همچنین می‌توانید برای هر فیلد از ساختار با استفاده از اضافه کردن آن‌ها به عنوان یک لیست از آرگومان ها که با فاصله از هم جدا شده‌اند یک نام را تعریف کنید.

```
[0x4A13B8C0]> pf 2*xw pointer type @ esp
0x00404888 [0] {
                  pointer     :
(*0xffffffff8949ed31)    type :     0x00404888  =     0x8949ed31
                  0x00404890  =     0x48e2
}
0x00404892 [1] {
(*0x50f0e483)         pointer :     0x00404892  =     0x50f0e483
                              type  :     0x0040489a  =     0x2440
}
```

یک مثال عملی استفاده از pf بر روی پلاگین باینری Gstreamer است :

```
$ radare ~/.gstreamer-0.10/plugins/libgstflumms.so
[0x000028A0]> seek      sym.gst_plugin_desc
[0x000185E0]> pf iissxssssss major minor name desc _init version  \
license     source      package     origin
major :     0x000185e0  =     0
minor :     0x000185e4  =     10
name  :     0x000185e8  =     0x000185e8  flumms
desc  :     0x000185ec  =     0x000185ec  Fluendo     MMS   source
_init :     0x000185f0  =     0x00002940
version     :     0x000185f4  =     0x000185f4  0.10.15.1
license     :     0x000185f8  =     0x000185f8  unknown
source      :     0x000185fc  =     0x000185fc  gst-fluendo-mms
package     :     0x00018600  =     0x00018600  Fluendo     MMS   source
origin      :     0x00018604  =     0x00018604  http://www.fluendo.com
```

### ۳.۵.۷ دیس اسمبلی

دستور pd برای دیس اسمبل کردن کد استفاده می شود.این یک مقدار عددی را به منظور مشخص کردن اینکه چه مقدار opcode باید دیس اسمبل شود قبول می کند.دستور pD نیز مشـابه بـه همیـن است اما به جای تعدادی دستورالعمل این تعداد معینی از بایت ها را دی کامپایل می کند.

```
d : disassembly N opcodes count of opcodes
D: asm.arch disassembler bsize bytes
[0x00404888]> pd 1
      ;-- entry0:
      0x00404888  31ed xor ebp, ebp
```

### ۳.۵.۸ انتخاب معماری

نوع معماری برای دیس اسمبلی توسط متغییر ارزیابی asm.arch تعریف می شود.شما می‌توانید با استفاده از e asm.arch = ? تمام متغییرهای معماری را لیست کنید.

```
[0xB7F08810]> e asm.arch =   ?
_d 16       8051 pd 8051     intel cpu
_d 16 32    arc gpl3 argonaut risc core
ad 16 32 64 arm gpl3 acorn risc machine cpu
_d 16 32 64 arm.cs bsd capstone arm disassembler
_d 16 32    arm.winedbg lgpl2 winedbg's arm disassembler
_d 16 32    avr gpl avr atmel
ad 32       bf lgpl3 brainfuck
_d 16       cr16 LGPL3 cr16 disassembly plugin
_d 16       csr PD Cambridge Silicon Radio (CSR)
ad 32 64    dalvik lgpl3 androidvm dalvik
ad 16 dcpu16  pd mojang's dcpu-16
_d 32 64    ebc LGPL3 EFI Bytecode
_d 8 gb     LGPL3 GameBoy(TM) (z80-like)
_d 16       h8300 LGPL3 H8/300 disassembly plugin
_d 8        i8080 BSD Intel 8080 CPU
ad 32       java Apache Java bytecode
_d 16 32    m68k BSD Motorola 68000
_d 32       malbolge LGPL3 Malbolge Ternary VM
ad 32 64    mips gpl3 mips cpu
_d 16 32 64 mips.cs bsd capstone mips disassembler
_d 16 32 64 msil pd .net microsoft intermediate language
_d 32       nios2 gpl3 nios ii embedded processor
_d 32 64    ppc gpl3 powerpc
_d 32 64    ppc.cs bsd capstone powerpc disassembler
ad          rar lgpl3 rar vm
_d 32       sh gpl3 superh-4 cpu
_d 32 64    sparc gpl3 scalable processor architecture
_d 32       tms320 lgplv3 tms320 dsp family
_d 32       ws lgpl3 whitespace esotheric vm
_d 16 32 64 x86 bsd udis86 x86-16,32,64
_d 16 32 64 x86.cs bsd capstone x86 disassembler
a_ 32 64    x86.nz lgpl3 x86 handmade assembler
ad 32       x86.olly gpl2 ollydbg x86 disassembler
ad 8        z80 nc-gpl2 zilog z80
```

### ۳.۵.۹ پیکربندی دیس اسمبلر

چندین گزینه وجود دارد که می‌تواند برای پیکربندی خروجی دیس اسمبلر استفاده شود، تمام این گزینه های می‌تواند با استفاده از e? Asm. شرح داده شود.

```
asm.os: Select operating system (kernel) (linux, darwin,w32,..)
asm.bytes: Display the bytes of each instruction
asm.cmtflgrefs: Show comment flags associated to branch referece
asm.cmtright: Show comments at right of disassembly if they fit in screen
asm.comments: Show comments in disassembly view
asm.decode: Use code analysis as a disassembler
asm.dwarf: Show dwarf comment at disassembly
```

```
asm.esil: Show ESIL instead of mnemonic
asm.filter: Replace numbers in disassembly using flags containing a dot in the
name in disassemb ly
asm.flags: Show flags
asm.lbytes: Align disasm bytes to left
asm.lines: If enabled show ascii-art lines at disassembly
asm.linescall:    Enable call lines
asm.linesout: If enabled show out of block lines
asm.linesright: If enabled show lines before opcode instead of offset
asm.linesstyle: If enabled iterate the jump list backwards
asm.lineswide: If enabled put an space between lines
asm.middle: Allow disassembling jumps in the middle of an instruction
asm.offset: Show offsets at disassembly
asm.pseudo: Enable pseudo syntax
asm.size: Show size of opcodes in disassembly (pd)
asm.stackptr: Show stack pointer at disassembly
asm.cycles: Show cpu-cycles taken by instruction at disassembly
asm.tabs: Use tabs in disassembly
asm.trace: Show execution traces for each opcode
asm.ucase: Use uppercase syntax at disassembly
asm.varsub: Substitute variables in disassembly
asm.arch: Set the arch to be usedd by asm
asm.parser: Set the asm parser to use
asm.segoff: Show segmented address in prompt (x86-16)
asm.cpu: Set the kind of asm.arch cpu
asm.profile: configure disassembler (default, simple, gas, smart, debug, full)
asm.xrefs: Show xrefs in disassembly
asm.functions: Show functions in disassembly
asm.syntax: Select assembly syntax
asm.nbytes: Number of bytes for each opcode at disassembly
asm.bytespace: Separate hex bytes with a whitespace
asm.bits: Word size in bits at assembler
asm.lineswidth: Number of columns for program flow arrows
```

### ۳.۵.۱۰ گرامر دیس اسمبلی

گرامر متغییر برای تاثیرگذاری در نوع نمایش اسمبلی که موتور اسمبلر در خروجی نشان می‌دهد استفاده می شود.

```
e asm.syntax = intel
e asm.syntax = att
```

شما همچنین می‌توانید asm.pseudo که یک نمایش تجربی از شبه کد است و asm.esil که خروجی ESIL (رشته های قابل ارزیابی زبان میانی) است را بررسی کنید.هدف آن نمایش یک خروجی قابل خواندن از هر opcode است.این نمایش‌ها می‌توانند به منظور شبیه سازی کد ارزیابی شوند.

### پرچم ها

پرچم ها شبیه به بوک مارک ها هستند.آن ها نشان دهنده یک آفست خاص در فایل هستند.پرچم ها می‌توانند در "فضای پرچم" گروه بندی شوند.فضای پرچم چیزی شبیه به فضای نام برای پرچم است.آنها برای گروه بندی پرچم ها با ویژگی‌های یا نوع مشابه استفاده می شوند.برخی از مثال‌های فضای پرچم می‌توانند بخش ها، ثبات ها و سمبول ها باشند.
برای ساخت یک پرچم عبارت زیر را بنویسید :

```
[0x4A13B8C0]> f flag_name @ offset
```

شما می‌توانید یک پرچم را به وسیله یک - در ابتدای نام حذف کنید.بسیاری از دستورها - را به عنوان پیشوند آرگومان و به عنوان یک راه برای حذف موارد قبول می کنند.

```
[0x4A13B8C0]> f -flag_name
```

برای سوئیچ بین فضاهای جدید پرچم یا ساخت آن از دستور fs استفاده کنید :

```
[0x4A13B8C0]> fs ; list flag spaces
00    symbols
01    imports
02    sections
03    strings
04    regs
05    maps
[0x4A13B8C0]> fs symbols ; select only flags in symbols flagspace
[0x4A13B8C0]> f ; list only flags in symbols flagspace
[0x4A13B8C0]> fs *      ; select all flagspaces
```

شما با استفاده از fr می توایند پرچم را تغییر نام دهید.

نوشتن

Radare می‌تواند فایل‌های باینری بارگذاری شده را در راه‌های مختلف دستکاری کند.شما می‌توانید فایل را تغییر اندازه داده، بایت ها را انتقال و کپی/چسباندن کنید، بایت های جدید را درج کنید (داده ها را به انتهای بلوک یا فایل انتقال دهید) و یا به سادگی بایت ها را در یک آدرس، محتوای یک فایل، یک رشته طولانی یا حتی اسمبلی درون خطی یک opcode بازنویسی کنید.

برای تغییر اندازه از دستور r استفاده کنید که یک آرگومان عددی را قبول می کند.یک مقدار مثبت اندازه جدید را به فایل تنظیم می کند.یک مقدار منفی N بایت از جستجوی جاری را جدا کرده و اندازه فایل را پایین می آورد.

```
r 1024      ; resize the file to 1024 bytes
r -10 @ 33 ; strip 10 bytes at offset 33
```

برای نوشتن بایت ها از دستور w استفاده کنید.این قالب‌های ورودی متعدد مانند اسمبلی درون خطی، dwordها، فایل ها، فایل‌های hexpair و رشته‌های گسترده را قبول می کند:

```
[0x00404888]> w?
|Usage: w[x] [str] [<file] [<<EOF] [@addr]
| w foobar        write string 'foobar'
| wh r2           whereis/which shell command
| wr 10           write 10 random bytes
| ww foobar       write wide string 'f\x00o\x00o\x00b\x00a\x00r\x00'
| wa push ebp     write opcode, separated by ';' (use '"' around the command)
| waf file        assemble file and write bytes
| wA r 0          alter/modify opcode at current seek (see wA?)
| wb 010203       fill current block with cyclic hexpairs
| wc[ir*?]        write cache undo/commit/reset/list (io.cache)
| wx 9090         write two intel nops
| wv eip+34       write 32-64 bit value
| wo? Hex         write in block with operation. 'wo?' fmi
| wm f0ff         set binary mask hexpair to be used as cyclic write mask
| ws pstring      write 1 byte for length and then the string
| wf -|file       write contents of file at current offset
| wF -|file       write contents of hexpairs file here
| wp -|file       apply radare patch file. See wp? fmi
| wt file [sz]       write to file (from current seek, blocksize or sz bytes)
```

برخی از مثال‌ها :

```
[0x00000000]> wx 123456 @ 0x8048300
[0x00000000]> wv 0x8048123 @ 0x8049100
[0x00000000]> wa jmp 0x8048320
```

۳.۸.۱ نوشتن با عمل
دستور wo (عمل نوشتن) انواع مختلفی از عملیات را قبول می‌کند که می‌تواند بر روی بلوک فعلی اعمال شوند.به عنوان مثال یک XOR, ADD, SUB …

```
[0x4A13B8C0]> wo?
|Usage: wo[asmdxoArl24] [hexpairs] @ addr[:bsize]
|Example:
| wox 0x90     ; xor cur block with 0x90
| wox 90         ; xor cur block with 0x90
| wox 0x0203 ; xor cur block with 0203
| woa 02 03 ; add [0203][0203][...] to curblk
| woe 02 03
|Supported operations:
| wow ==  write looped value (alias for 'wb')
| woa  +=  addition
| wos   -=  substraction
| wom *=  multiply
| wod  /=   divide
| wox  ^=  xor
| woo  |=   or
| woA &= and
| woR random bytes (alias for 'wr $b' )
| wor >>= shift right
| wol <<= shift left
| wo2 2= 2 byte endian swap
| wo4 4=4 byte endian swap
```

با این روش پیاده‌سازی یک الگوریتم رمزگذاری با استفاده از هسته اولیه radare ممکن است.
جلسه زیر یک xor(90) + addition(01 02) را انجام می‌دهد :

```
[0x7fcd6a891630]> px
- offset    - 0 1 2 3 4 5 6 7 8 9 A B C D E F        0123456789ABCDEF
0x7fcd6a891630    4889    e7e8    6839    0000    4989    c48b    05ef    1622
H...h9..I......"
0x7fcd6a891640 005a 488d 24c4 29c2 5248 89d6 4989 e548         .ZH.$.).RH..I..H
0x7fcd6a891650    83e4    f048    8b3d    061a    2200    498d    4cd5    1049
...H.=..".I.L..I
0x7fcd6a891660    8d55    0831    ede8    06e2    0000    488d    15cf    e600
.U.1......H.....

0x7fcd6a891630]> wox 90
[0x7fcd6a891630]> px
- offset    -0 1 2 3 4 5 6 7 8 9 A B C D E F               0123456789ABCDEF
0x7fcd6a891630    d819    7778    d919    541b    90ca    d81d    c2d8    1946
..wx..T........F
0x7fcd6a891640    1374    60d8    b290    d91d    1dc5    98a1    9090    d81d
.t`.............
0x7fcd6a891650     90dc     197c     9f8f     1490     d81d     95d9     9f8f
1490              ...|............
0x7fcd6a891660     13d7     9491     9f8f     1490     13ff     9491     9f8f
1490              ..............
```

```
[0x7fcd6a891630]> woa 01 02
[0x7fcd6a891630]> px
- offset -      0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x7fcd6a891630 d91b 787a 91cc d91f 1476 61da 1ec7 99a3  ..xz.....va.....
0x7fcd6a891640 91de 1a7e d91f 96db 14d9 9593 1401 9593  ...~............
0x7fcd6a891650 c4da 1a6d e89a d959 9192 9159 1cb1 d959  ...m...Y...Y...Y
0x7fcd6a891660 9192 79cb 81da 1652 81da 1456 a252 7c77  ..y ....R...V.R|w
```

حرکت/چسباندن
شما می‌توانید بایت ها را در محیط تصویری با استفاده از کلید y و Y حرکت/بچسبانید که نام مستعاری برای دستورهای y و yy پوسته هستند.این دستورات بر روی بافر داخلی عمل می‌کنند که N بایت شمارش شده را از جستجوی فعلی ذخیره می کند.شما می‌توانید با استفاده از دستور yy جستجوی فعلی را دوباره ذخیره کنید.

```
[0x00000000]> y?
|Usage: y[ptxy] [len] [[@]addr]
| y                 show yank buffer information (srcoff len bytes)
| y 16              copy 16 bytes into clipboard
| y 16 0x200        copy 16 bytes into clipboard from 0x200
| y 16 @ 0x200      copy 16 bytes into clipboard from 0x200
| yp                print contents of clipboard
| yx                print contents of clipboard in hexadecimal
| yt 64 0x200       copy 64 bytes from current seek to 0x200
| yf 64 0x200 file  copy 64 bytes from 0x200 (opens w/io), use -1 for all bytes
| yfa file          copy copy all bytes from from file (opens w/io)
| yy 0x3344         paste clipboard
```

نمونه :

```
[0x00000000]> s 0x100   ; seek at 0x100
[0x00000100]> y 100           ; yanks 100 bytes from here
[0x00000200]> s 0x200   ; seek 0x200
[0x00000200]> yy        ; pastes 100 bytes
```

با استفاده از دستور yt شما می‌توانید یک حرکت و چسباندن را در یک خط انجام دهید.نحوه استفاده همانند زیر است :

```
[0x4A13B8C0]> x
Offset       0 1 2 3 4 5 6 7 8 9 A B     0123456789AB
0x4A13B8C0, 89e0 e839 0700 0089 c7e8 e2ff ...9........
0x4A13B8CC, ffff 81c3 eea6  0100 8b83 08ff     ............
0x4A13B8D8,  ffff   5a8d 2484  29c2            ..Z.$.).
[0x4A13B8C0]> yt 8 0x4A13B8CC @ 0x4A13B8C0
[0x4A13B8C0]> x
Offset  0 1 2 3 4 5 6 7 8 9 A B    0123456789AB
0x4A13B8C0,  89e0 e839 0700 0089 c7e8 e2ff     ...9........
0x4A13B8CC, 89e0 e839 0700 0089 8b83 08ff...9........
0x4A13B8D8,  ffff 5a8d 2484 29c2               ..Z.$.).
```

مقایسه بایت ها
شما می‌توانید با استفاده از دستور c داده‌ها را مقایسه کنید.این یک ورودی را در قالب‌های مختلف پذیرفته و ورودی را در مقابل بایت ها جاری مقایسه می کند.

```
[0x00404888]> c?
| Usage: c[?dfx] [argument]
| c [string]        Compares a plain with escaped chars string
| cc [at] [(at)]    Compares in two hexdump columns of block size
| c4 [value]        Compare a doubleword from a math expression
| c8 [value]        Compare a quadword from a math expression
| cx [hexpair]      Compare hexpair string
| cX [addr]         Like 'cc' but using hexdiff output
| cf [file]         Compare contents of file at current seek
| cg[o] [file]      Graphdiff current file and [file]
| cu [addr] @at     Compare memory hexdumps of $$ and dst in unified diff
| cw[us?] […]       Compare memory watchers
| cat [file]        Show contents of file (see pwd, ls)
| cl|cls|clear      Clear screen, (clear0 to goto 0, 0 only)
```

یک مثال از مقایسه حافظه :

```
[0x08048000]> p8 4
7f 45 4c 46
[0x08048000]> cx 7f 45 90 46
Compare 3/4 equal bytes
0x00000002 (byte=03) 90 ' '   -> 4c 'L'
[0x08048000]>
```

یک زیر دستور دیگر از c (مقایسه) دستور cc است که یک استاندارد برای مقایسه کد است.

```
[0x4A13B8C0]> cc 0x39e8e089 @ 0x4A13B8C0
[0x08049A80]> cc sym.main2 @ sym.main
```

c8 یک کلمه چهارتایی را از محل فعلی (0x00000000) با عبارت ریاضی مقایسه می‌کند

```
[0x00000000]> c8 4
Compare 1/8 equal bytes (0%)
0x00000000 (byte=01) 7f ' '   -> 04 ' '
0x00000001 (byte=02) 45 'E' -> 00 ' '
0x00000002 (byte=03) 4c 'L' -> 00 ' '
```

عدد پارامتر با استفاده از نام پرچم و غیره همچنین می‌تواند یک عبارت ریاضی باشد :

```
[0x00000000]> cx 7f469046
Compare 2/4 equal bytes
0x00000001 (byte=02) 45 'E'   -> 46 'F'
0x00000002 (byte=03) 4c 'L' -> 90 ' '
```

ما می‌توانیم مقایسه بلوک جاری را با فایلی که از قبل بر روی دیسک رونوشت گرفته‌ایم با استفاده از دستور مقایسه انجام دهیم.

```
r2 /bin/true
[0x08049A80]> s 0
[0x08048000]> cf /bin/true
Compare 512/512 equal bytes
```

# حالت تصویری

حالت تصویری
حالت تصویری یک رابط کاربر پسند برای اعلان خط دستور Radare است که کلیدهای جهت نما را قبول می‌کند و یک جهت نما برای انتخاب بایت ها و برخی از مجموعه کلیدها برای سهولت در استفاده از اشکال زدا دارد.
در این حالت شما با استفاده از کلید e (ارزیابی) می‌توانید پیکربندی را به یک روش آسان تغییر دهید یا می‌توانید با فشار دادن کلید t پرچم ها را ردیابی کرده و در فضای پرچم به این طرف و آنطرف بروید.
برای دریافت کمک در مورد تمام ترکیب های کلید در حالت تصویری شما می‌توانید ? را فشار دهید :

```
Visual mode help:
?         show this help or manpage in cursor mode
_         enter the hud
.         seek to program counter
/         in cursor mode search in current block
:cmd      run radare command
;[-]cmt   add/remove comment
/*+-[]    change block size, [] = resize hex.cols
>||<      seek aligned to block size
i/a/A     (i)nsert hex, (a)ssemble code, visual (A)ssembler
b/B       toggle breakpoint / automatic block size
c/C       toggle (c)ursor and (C)olors
d[f?]     define function, data, code, ..
D         enter visual diff mode (set diff.from/to)
e         edit eval configuration variables
f/F       set/unset flag
gG        go seek to begin and end of file (0-$s)
Hjkl      move around (or HJKL) (left-down-up-right)
mK/'K     mark/go to Key (any key)
M         walk the mounted filesystems
n/N       seek next/prev function/flag/hit (scr.nkey)
o         go/seek to given offset
p/p       rotate print modes (hex, disasm, debug, words, buf)
q         back to radare shell
R         randomize color palette (ecr)
sS        step/    step over
t         track flags (browse symbols, functions..)
T         browse anal info and comments
V         visual code analysis menu
V/W       (V)iew graph using cmd.graph (agv?), open (W)ebUI
uU         undo/redo seek
X         show xrefs to seek between them
yY        copy and paste selection
z         toggle zoom mode
Enter     follow address of jump/call
Function  Keys: (See 'e key.'), defaults to:
  F2        toggle breakpoint
  F7        single step
  F8        step over
  F9        continue
```

از حالت تصویری شما می‌توانید با استفاده از کلید های I و c به حالت‌های درج و مکان نما سوئیچ کنید.

مکان نمای تصویری
برای ظاهر شدن مکان نما یا ناپدید شدن آن حروف کوچک c را فشار دهید.مکان نما برای انتخاب

محدود های از بایت ها یا فقط اشاره به یک بایت برای پرچم استفاده مـی شـود(بـرای سـاخت یـک پرچم جدید در جایی که مکان نما به آن اشاره می‌کند کلید f را فشار دهید).
اگر شما یک محدوده از بایت ها را انتخاب کردید کلید i را فشار داده و سپس آرایه بایت را بـا بـایت هایی که انتخاب کرده‌اید بازنویسی کنید.به عنوان مثال :

```
<select     10 bytes in visual mode using upper hjkl>
<press 'I' and then '12 34'>
```

۱۰ بایت انتخاب شده تبدیل خواهد شد به : 12 34 12 34 12 34 12 34 12 34 محدوده بایت های انتخاب شده می‌توانند با همدیگر و همراه با کلید d استفاده شوند تا نوع داده بایت های انتخاب شده را به یک رشته، کد یا آرایه ای از بایت ها تغییر دهد.
این برای افزایش دیس اسمبلی و متاداده یا فقط تراز کردن کد در صورتی که بایت ها با کد ترکیب شده‌اند مفید است.
در حالت مکان نما شما می‌توانید اندازه بلوک را به سادگی با انتقال آن به مکـانی کـه می‌خواهیـد و فشاردادن _ تنظیم کنید و سپس اندازه بلوک تغییر دهید.

درج تصویری
حالت درج به شما امکان نوشتن بایت ها در سطح اندک را می‌دهد کـه بیشـتر شـبیه ویرایشـگرهای هگزادسیمال رایج است.در این حالت شما می‌توانید با فشار دادن <tab> بین سـتون‌های hexa و ascii از روگرفت هگزادسیمال حرکت کنید.
برای برگشتن به حالت طبیعی <tab> را فشار داده تا به نمایش هگزادسیمال برگشته و سـپس q را فشار دهید.(توجه: اگر شما q را در حالت ascii فشار دهید، این به جای خـروج از ایـن حـالت q را درج می کند).
در حالت تصویری کلیدهای دیگری نیز برای درج و نوشتن داده‌ها وجود دارند.درواقـع بـا فشـار دادن کلید I از شما برای یک جفت رشته در مبنای شانزده درخواست می‌شود یا با استفاده از a جایی که مکان نما قرار دارد می‌توانید اسمبلی بنویسید.

Xrefs تصویری
Radare برای رابط تصویری و کد اسمبلی ویژگی‌های کاربر پسند زیادی را پیاده‌سازی می کند.یکی از آن‌ها کلید x است که یک منو را برای انتخاب xref (داده یا کد)در مقابل محل فعلـی ارائـه داده و به آنجا پرش می کند.به عنوان مثال، هنگام فشار دادن x و نگاه کردن به آن XREF :

```
| .....-->   ; CODE (CALL) XREF from 0x00402b98 (fcn.004028d0)
| .....-->   ; CODE (CALL) XREF from 0x00402ba0 (fcn.004028d0)
| .....-->   ; CODE (CALL) XREF from 0x00402ba9 (fcn.004028d0)
| .....-->   ; CODE (CALL) XREF from 0x00402bd5 (fcn.004028d0)
| .....-->   ; CODE (CALL) XREF from 0x00402beb (fcn.004028d0)
| .....-->   ; CODE (CALL) XREF from 0x00402c25 (fcn.004028d0)
| .....-->   ; CODE (CALL) XREF from 0x00402c31 (fcn.004028d0)
| .....-->   ; CODE (CALL) XREF from 0x00402c40 (fcn.004028d0)
| .....-->   ; CODE (CALL) XREF from 0x00402c51 (fcn.004028d0)
```

بعد از فشار دادن x

```
[GOTO XREF]>
[0]   CODE  (CALL)      XREF  0x00402b98  (loc.00402b38)
[1]   CODE  (CALL)      XREF  0x00402ba0  (loc.00402b38)
[2]   CODE  (CALL)      XREF  0x00402ba9  (loc.00402b38)
[3]   CODE  (CALL)      XREF  0x00402bd5  (loc.00402b38)
[4]   CODE  (CALL)      XREF  0x00402beb  (loc.00402b38)
[5]   CODE  (CALL)      XREF  0x00402c25  (loc.00402b38)
[6]   CODE  (CALL)      XREF  0x00402c31  (loc.00402b38)
```

```
[7]   CODE  (CALL)      XREF  0x00402c40  (loc.00402b38)
[8]   CODE  (CALL)      XREF  0x00402c51  (loc.00402b38)
[9]   CODE  (CALL)      XREF  0x00402c60  (loc.00402b38)
```

تمام فراخوانی ها و پرش ها شماره گذاری شده‌اند (1,2,3...) این اعداد مجموعه‌ای از کلیدها برای جستجو از محیط تصویری هستند.تمام تاریخچه های مربوط به جستجو ذخیره مـی شـوند، بـا فشـار دادن کلید u شما به عقب برخواهید گشت :)

# جستجوی بایت ها

جستجوی بایت ها
موتور جستجوی radare مبتنی بر کارهای انجام شده توسط esteve است که تعـدادی ویژگـی بـه آن اضافه شده است که امکان جستجوی چندین کلمه کلیدی با ماسک‌های باینری و مشـخص کـردن نتایج به صورت خودکار را می دهد.

این دستور قدرتمند / است :

```
[0x00000000]> /
Usage:      /[amx/]     [arg]
/ foo\x00               search for string `foo\0`
/w foo                        search for wide string `f\0o\0o\0`
/wi foo                       search for wide string ignoring case `f\0o\0o\0`
/! ff                   search for first occurrence not matching
/i foo                        search for string `foo` ignoring case
/e /E.F/i               match regular expression
/x ff0033               search for hex string
/x ff..33               search for hex string ignoring some nibbles
/x ff43 ffd0            search for hexpair with mask
/d 101112               search for a deltified sequence of bytes
/!x 00                  inverse hexa search (find first byte != 0x00)
/c jmp [esp]            search for asm code (see search.asmstr)
/a jmp eax              assemble opcode and search its bytes
/A                      search for AES expanded keys
/r sym.printf           analyze opcode reference an offset
/R                      search for ROP gadgets
/P                      show offset of previous instruction
/m magicfile            search for matching magic file (use blocksize)
/p patternsize          search for pattern of given size
/z min max              search for strings of given size
/v[?248] num            look for a asm.bigendian 32bit value
//                      repeat last search
/b                      search backwards
```

با Radare هرچیزی به عنوان یک فایل به کار گرفته می‌شود و فرقی نمی‌کند که ایـن یـک سـوکت، یک دستگاه از راه دور، یک فرایند حافظه و... باشد.

جستجوی اولیه
یک جستجوی اولیه برای یک رشته ساده در کل یک فایل چیزی شبیه به زیر است :

```
$ r2 -c "/ lib" -q /bin/ls
Searching 3 bytes from 0x00400000 to 0x0041ae08: 6c 69 62
Hits: 9
0x00400239  hit0_0      "lib64/ld-linux-x86-64.so.2"
0x00400f19  hit0_1      "libselinux.so.1"
0x00400fae  hit0_2      "librt.so.1"
0x00400fc7  hit0_3      "libacl.so.1"
0x00401004  hit0_4      "libc.so.6"
0x004013ce  hit0_5      "libc_start_main"
0x00416542  hit0_6      "libs/"
0x00417160  hit0_7      "lib/xstrtol.c"
0x00417578  hit0_8      "lib"
```

r2 -q // حالت بی سروصدا (بدون اعلان) و خارج شدن بعد از -i .

همانطور که می بینید، Radare یک پرچم hit را برای هر نتیجه جستجو که پیدا می‌کند ایجاد می کند.شما می‌توانید با استفاده از دستور ps و با روش زیر رشته‌ها را در این آفست ها به تصویر بکشید :

```
[0x00404888]> / ls
...
[0x00404888]> ps @ hit0_0
lseek
```

ما همچنین می‌توانیم با استفاده از /w رشته‌های طولانی را جستجو کنیم (آن هایی که شامل صفر بین هر حرف هستند) :

```
[0x00000000]> /w Hello
0 results found.
```

همچنین امکان ترکیب دنباله ای از هگزادسیمال در رشته جستجو وجود دارد :

```
[0x00000000]> / \x7FELF
```

اما اگر شما بخواهید یک جستجوی هگزا دسیمال را انجام دهید، احتمالاً یک ورودی هگزادسیمال را با /x ترجیح می‌دهید :

```
[0x00000000]> /x 7F454C46
```

هنگامی که جستجو انجام شد، جستجو در فضای پرچم search ذخیره می‌شود :

```
[0x00000000]> f
0x00000135  512   hit0_0
0x00000b71  512   hit0_1
0x00000bad  512   hit0_2
0x00000bdd  512   hit0_3
0x00000bfb  512   hit0_4
0x00000f2a  512   hit0_5
```

برای پاک کردن این پرچم ها، شما می‌توانید از دستور *f@-hit استفاده کنید.
گاهی اوقات که برای مدت طولانی با فایل یکسان کار می‌کنید نیاز به راه اندازی آخرین جستجو بیش از یکبار دارید و احتمالاً ترجیح می‌دهید که از دستور // به جای نوشتن دوباره تمام دستورات استفاده کنید.

```
[0x00000f2a]> //  ; repeat last search
```

پیکربندی جستجو
موتور جستجو می‌تواند توسط رابط e پیکربندی شود :

```
Configuration:
e cmd.hit = x          ; command to execute on every search hit
e search.distance = 0  ; search string distance
e search.in = [foo]    ; boundaries to raw, block, file, section)
e search.align = 4     ; only catch aligned search hits
e search.from = 0      ; start address
e search.to = 0        ; end address
e search.asmstr = 0    ; search string instead of assembly
e search.flags = true  ; if enabled store flags on keyword hits
```

متغییر search.align برای مشخص کردن اینکه تنها جستجوی valid باید در این تراز متناسب باشد استفاده می شود.شما می‌توانید با استفاده از e search.align=4 تنها چهاربایت آدرس همتراز را پیدا کنید.
متغیر منطقی search.flag در هنگام پیدا کردن چیزی پرچم های راه اندازی موتور را ایجاد می کند.اگر جستجو توسط کاربر و به وسیله C^ متوقف شود آنوقت پرچم search_stop ایجاد می شود.

جستجوی الگو
دستور search به شما اجازه تکرار الگوهای جستجو را در مقابل IO می‌دهد تا قادر به شناسایی تکرار توالی از بایت ها بدون مشخص کردن آن‌ها باشید.تنها ویژگی انجام این جستجو این است که به صورت دستی حداقل طول این الگوها را تعریف می کنید.

در اینجا یک مثال وجود دارد :

```
[0x00000000]> /p 10
```

خروجی دستور الگوهای مختلف پیدا شده و تعداد مرتبه‌ای که تکرار شده‌اند را نشان می‌دهد .

خودکار سازی
متغیر cmd.hit برای تعریف یک دستور که در هنگام دستیابی شدن توسط یک موتور جستجو اجرا می‌شود به کار می رود.اگر شما می‌خواهید بیش از یک دستور را اجرا کنید از ; یا script-file-name. برای درج یک فایل به عنوان اسکریپت استفاده کنید.

برای مثال :

```
[0x00404888]> e cmd.hit = p8 8
[0x00404888]> / lib
Searching 3 bytes from 0x00400000 to 0x0041ae08: 6c 69 62
Hits: 9
0x00400239  hit4_0      "lib64/ld-linux-x86-64.so.2"
31ed4989d15e4889
0x00400f19  hit4_1      "libselinux.so.1"
31ed4989d15e4889
0x00400fae  hit4_2      "librt.so.1"
31ed4989d15e4889
0x00400fc7  hit4_3      "libacl.so.1"
31ed4989d15e4889
0x00401004  hit4_4      "libc.so.6"
31ed4989d15e4889
0x004013ce  hit4_5      "libc_start_main"
31ed4989d15e4889
0x00416542  hit4_6      "libs/"
31ed4989d15e4889
0x00417160  hit4_7      "lib/xstrtol.c"
31ed4989d15e4889
0x00417578  hit4_8      "lib"
31ed4989d15e4889
```

جستجوی رو به عقب
برای جستجوی رو به عقب از b/ استفاده کنید.

جستجو در اسمبلی
اگر شما می‌خواهید نوع خاصی از opcodes را جستجو کنید یا از c/ و یا a/ استفاده کنید :

```
/c jmp [esp]            search for asm code
[0x00404888]> /c jmp qword   [rdx]
f hit_0     @ 0x0040e50d      # 2: jmp qword [rdx]
f hit_1     @ 0x00418dbb      # 2: jmp qword [rdx]
f hit_2     @ 0x00418fcb      # 3: jmp qword [rdx]
f hit_3     @ 0x004196ab      # 6: jmp qword [rdx]
f hit_4     @ 0x00419bf3      # 3: jmp qword [rdx]
f hit_5     @ 0x00419c1b      # 3: jmp qword [rdx]
f hit_6     @ 0x00419c43      # 3: jmp qword [rdx]
/a jmp eax        assemble opcode and search its bytes
[0x00404888]> /a jmp eax
Hits: 1
0x004048e7 hit3_0 ffe00f1f8000000000b8
```

جستجو کلیدهای AES

با تشکر از Victor Muoz که من پشتیبانی از الگوریتمی که او برای پیدا کردن کلیدهای گسترش یافته AES توسعه داده بود را اضافه کردم.این جستجو را از محل فعلی تا cfg.limit یا پایان فایل اجرا می‌کند.شما همیشه می‌توانید جستجو را با فشار دادن کلید C^ متوقف کنید :

```
$ sudo r2 /dev/mem
[0x00000000]> /A
0 AES keys found
```

# دیس اسمبل کردن

دیس اسمبل کردن
دیس اسمبل کردن در Radare فقط یک راه برای نشان دادن یک دسته از بایت ها است.سپس این می‌تواند به عنوان یک حالت چاپ با دستور 'p' به کار گرفته شود.
در زمان های قدیم هنگامی که هسته Radare کوچک‌تر بود.دیس اسمبلر توسط یک فایل خارجی rsc به کار گرفته می‌شود و بنابراین Radare بلوک جاری را به یک فایل روگرفت می‌کند و اسکریپت objdump را در یک راه مناسب برای دیس اسمبل کردن Intel، ARM و غیره فراخوانی می کند.
بدیهی است که این یک راه حلی است که کار می‌کند اما برای تکرار کردن همان کار به تعداد زیاد پردازنده زیادی را می‌گیرد زیرا هیچ مکانی برای ذخیره سازی وجود ندارد و پیمایش کاملاً آهسته است.
امروزه دیس اسمبلر یکی از اصول اولیه در Radare است که به شما امکان انتخاب انواع مختلف معماری برای دیس اسمبلی کردن را با دستور 'pd' می دهد.
دستور 'pd' یک آرگومان عددی را به منظور مشخص کردن اینکه چه تعداد opcode از بلوک جاری را شما برای دیس اسمبلی می‌خواهید می پذیرد.بسیاری از دستورات در Radare با اندازه بلوک محدود شده اند.بنابراین اگر شما دیس اسمبل کردن بایت های بیشتری را بخواهید باید از دستور 'b' برای مشخص کردن اندازه جدید بلوک استفاده کنید.

```
[0x00000000]> b 100     ; set block size to 100
[0x00000000]> pd        ; disassemble 100 bytes
[0x00000000]> pd 3      ; disassemble 3 opcodes
[0x00000000]> pD 30     ; disassemble 30 bytes
```

دستور 'pD' شبیه به 'pd' کار می‌کند اما به جای تعداد opcodeها تعداد بایت ها را می گیرد.
گرامر 'pseudo' به زبان انسان نزدیک‌تر است اما اگر شما تعداد زیادی کد را خوانده باشید این می‌تواند آزاردهنده باشد :

```
[0xB7FB8810]> e asm.pseudo = true
[0xB7FB8810]> pd 3
0x00404888  31ed  ebp = 0
0x0040488a  4989d1 r9 = rdx
0x0040488d  5e pop      rsi
[0xB7FB8810]> e asm.syntax=intel
[0xB7FB8810]> pd 3
0xB7FB8810, mov eax, esp
0xB7FB8812  call 0xb7fb8a60
0xB7FB8817  add edi, eax
[0xB7FB8810]> e asm.syntax=att
[0xB7FB8810]> pd 3
0xB7FB8810, mov %esp, %eax
0xB7FB8812  call 0xb7fb8a60
0xB7FB8817  add %eax, %edi
```

اضافه کردن متاداده
کار کردن بر روی فایل‌های باینری باعث یادداشت برداری و تعریف اطلاعات در بالای فایل‌های کاملاً مهم می‌شود.Radare راه‌های متعددی را برای دریافت و به دست آوردن این اطلاعات از انواع مختلفی از فایل‌ها فراهم می کند.پیروی از چند اصل *nix نوشتن یک برنامه کوچک در اسکریپت که از objdump, otool استفاده می‌کند تا اطلاعات را از باینری دریافت کرده و آن را در Radare وارد کند را کاملاً آسان می کند.
شما می‌توانید به یکی از این اسکریپت ها بانام 'idc2r.py' که همراه با Radare توزیع می‌شود نگاه کنید :
این اسکریپت به صورت 'idc2r.py file.idc>file.r2' فراخوانی می‌شود.این یک فایل IDC که از پایگاه داده IDA صادر شده است را می‌خواند و توضیحات و نام توابع را وارد می کند.
ما با استفاده از دستور '.' از Radare می‌توانیم 'file.r2' را وارد کنیم (شبیه به پوسته) :

```
[0x00000000]> . file.r2
```

دستور '.' برای تفسیر داده‌ها از منابع خارجی مانند فایل ها، برنامه و غیره استفاده می‌شود.به همین ترتیب ما می‌توانیم کار مشابه را بدون نوشتن فایل انجام دهیم :

```
[0x00000000]> .!idc2r.py < file.idc
```

دستور 'C' یکی از دستوراتی است که برای مدیریت توضیحات و تبدیل داده‌ها استفاده می‌شود، به طوری که شما می‌توانید یک محدوده از بایت ها را تعریف کرده تا به عنوان کد یا رشته تفسیر شوند.همچنین امکان تعریف پرچم ها و اجرای کد در یک دنباله خاص وجود دارد تا توضیحات را از یک فایل خارجی یا پایگاه داده واکشی کند.

در اینجا کمک وجود دارد.

```
[0x00404cc0]> C?
|Usage: C[-Lcvsdfm?] [...]
| C*                                                List meta info in r2
commands
| C- [len] [@][addr]                              delete metadata at given
address range
| CL[-] [addr|file:line [addr] ]   show 'code line' information (bininfo)
| Cl file:line [addr]              add comment with line information
| CC[-] [comment-text]       add/remove comment. Use CC! To edit with $EDITOR
| Cca[-at]|[at] [text]       add/remove comment at given address
| Cv[-] offset reg name      add var substitution
| Cs[-] [size] [[addr]]       add string
| Ch[-] [size] [@addr]        hide data
| Cd[-] [size]                hexdump data
| Cf[-] [sz] [fmt..]          format memory (see pf?)
| Cm[-]     [sz] [fmt..]      magic parse (see pm?)
[0x00404cc0]>
[0x00000000]> Cca 0x0000002 this guy seems legit
[0x00000000]> pd 2
 ;this guy seems legit   [rax], al   add  0000      0x00000000
  0x00000002      0000  add   [rax], al
```

دستور 'C' به شما اجازه تغییر نوع داده را می دهد.سه نوع اصلی عبارت اند از : کد(دیس اسمبلی که از asm.arch استفاده می کند)، داده (آرایه ای از بایت) یا رشته.
در حالت تصویری مدیریت آسان‌تر است زیرا این از کلید 'd' برای تغییر نوع داده استفاده می کند.با استفاده از مکان نما محدوده بایت ها را انتخاب کنید (کلید 'c' برای تغییر به حالت مکان نما و کلیدهای جهت نما برای انتخاب) و سپس 'ds' را برای تبدیل به رشته فشار دهید.
شما می‌توانید با استفاده از دستور Cs از پوسته نیز این کار را انجام دهید :

```
[0x00000000]> f string_foo @ 0x800
[0x00000000]> Cs 10 @ string_foo
```

تا کردن/آشکار کردن کاملاً نابهنگام است اما این ایده از مفهوم 'folder' از VIM می آید.به طوری که شما می‌توانید محدوده‌ای از بایت ها را در حالت دیس اسمبلی انتخاب کرده و با فشار دادن '<' بایت ها را در یک خط جا کنید یا از '>' برای آشکار کردن آن‌ها استفاده کنید.این کار فقط برای سهولت خوانایی کد استفاده می شود.
دستور Cm برای تعریف یک رشته با قالب حافظه استفاده می‌شود (همین کار با استفاده از دستور pf انجام می شود). در اینجا یک مثال وجود دارد :

```
[0x7fd9f13ae630]> Cf 16 2xi foo bar
[0x7fd9f13ae630]> pd
```

```
;-- rip:
0X7fd9f13ae630 format 2xi foo bar {
     0X7fd9f13ae630 [0] {
                         foo : 0x7fd9f13ae630 = 0xe8e78948
                         bar : 0x7fd9f13ae634  = 14696
      }
      0X7fd9f13ae638 [1] {
                         foo : 0x7fd9f13ae638 = 0x8bc48949
                         bar : 0x7fd9f13ae63c  = 571928325
      }
      } 16
            0X7fd9f13ae633 e868390000    call  0x7fd9f13b1fa0
            0X7fd9f13b1fa0() ; rip
            0x7fd9f13ae638   4989c4           mov   r12,  rax
```

به این ترتیب تعریف ساختار با استفاده از تنها یک خط امکان‌پذیر است.برای اطلاعات بیشتر 'print memory' را ببینید.
همچنین در محیط گرافیکی با فشار دادن کلید 'd' (تبدیل داده)، تمام دستورات C* می‌تواند قابل دسترسی شود.

# Rabin2

Rabin2

تحت این نام که شبیه به عربی است، Radare قدرت یک ابـزار فوق‌العـاده بـرای مـدیریت فایل‌هـای باینری و گرفتن اطلاعات به منظور نمایش آن در خط دستور یا وارد کـردن آن در هسـته را مخفـی کرده است.

Rabin2 قادر به اداره فایل با قالب‌های متعدد ماننـد Java CLASS, ELF, PE, MACH-O و غیـره را دارد و این قادر به وارد کردن/صادر کردن سمبول، وابستگی‌های کتابخـانه ای، رشـته‌ها و بخش‌هـای داده،xrefs، آدرس نقطه ورود،بخش ها، نوع معماری و غیره است.

```
$ rabin2    -h
Usage:      rabin2 [-AcdehHiIjlLMqrRsSvVxzZ] [-@ addr] [-a arch] [-b bits] [-B
addr] [-c F:C:D] [-f str] [-m addr] [-n str] [-N len] [-o str] [-O str] file
-@ [addr]         show section, symbol or import at addr
-A                list archs
-a [arch]         set arch (x86, arm, .. or <arch>_<bits>)
-b [bits]         set bits (32, 64  ...)
-B [addr]         override base address (pie bins)
-c [fmt:C:D]      create [elf,mach0,pe] with Code and Data hexpairs (see -a)
-C                list classes
-d                show debug/dwarf information
-e                entrypoint
-f [str]          select sub-bin named str
-g                same as -SMRevsiz (show all info)
-h                this help
-H                header fields
-i                imports (symbols imported from libraries)
-I                binary info
-j                output in json
-l                linked libraries
-L                list supported bin plugins
-m [addr]         show source line at addr
-M                main (show address of main symbol)
-n [str]          show section, symbol or import named str
-N [minlen]       force minimum number of chars per string (see -z)
-o [str]          output file/folder for write operations (out by default)
-O [str]          write/extract operations (-O help)
-q                be quiet, just show fewer data
-r                radare output
-R                relocations
-s                symbols (exports)
-S                sections
-v                use vaddr in radare output (or show version if no file)
-x                extract bins contained in file
-z                strings (from data section)
-zz               strings (from raw bins [e bin.rawstr=1])
-Z                guess size of binary program
```

هویت فایل

شناسـایی فایـل از طریـق پرچـم -I انجـام مـی شـود، ایـن اطلاعـات را در مـورد کلاس بـاینری، رمزگذاری، OS، نوع و غیره نشان می دهد.

```
$ rabin2 -I /bin/ls
```

```
File        /bin/ls
type        EXEC (Executable file)
pic         false
has_va      true
root        elf
class       ELF64
lang        c
arch        x86
bits        64
machine     AMD   x86-64      architecture
os          linux
subsys      linux
endian      little
strip       true
static      false
linenum     false
lsyms       false
relocs      false
rpath       NONE
```

همانطور که گفته شد ما پرچم -r را برای استفاده از تمام اطلاعات در Radare اضافه کردیم :

```
$ rabin2 -Ir /bin/ls
e       file.type=elf
e       cfg.bigendian=false
e       asm.os=linux
e       asm.arch=x86
e       anal.arch=x86
e       asm.bits=64
e       asm.dwarf=true
```

نقطه ورودی
پرچم "-e" به ما اجازه دانستن نقطه ورودی برنامه را می دهد.

```
$ rabin2 -e /bin/ls
[Entrypoints]
addr=0x00004888 off=0x00004888     baddr=0x00000000
1     entrypoints
$     rabin2      -er /bin/ls
fs    symbols
f     entry0 @ 0x00004888
s     entry0
```

وارد کردن
Rabin2 قادر به گرفتن تمام اشیاء وارد شده و همچنین آفست آن‌ها در PLT است، این اطلاعات کاملاً مفید هستند و به عنوان مثال برای تشخیص اینکه کدام تابع توسط یک دستورالعمل فراخوانی صدا زده شده است.

```
$ rabin2 -i /bin/ls |head
[Imports]
Ordinal=001  plt=0x000021b0        bind=GLOBAL          type=FUNC
name=__ctype_toupper_loc
Ordinal=002  plt=0x000021c0        bind=GLOBAL          type=FUNC name=__uflow
Ordinal=003  plt=0x000021d0        bind=GLOBAL          type=FUNC name=getenv
ordinal=004 plt=0x000021e0    bind=GLOBAL type=FUNC   name=sigprocmask
ordinal=005 plt=0x000021f0    bind=GLOBAL type=FUNC   name=raise
ordinal=006 plt=0x00002210    bind=GLOBAL type=FUNC   name=localtime
```

```
ordinal=007 plt=0x00002220    bind=GLOBAL type=FUNC   name=__mempcpy_chk
ordinal=008 plt=0x00002230    bind=GLOBAL type=FUNC   name=abort
ordinal=009 plt=0x00002240    bind=GLOBAL type=FUNC   name=__errno_location
(...)
```

سمبول ها (نقل و انتقال)
در rabin لیست سمبول ها بسیار مشابه به نقل و انتقال ها کار می کند.

```
$ rabin2 -s /bin/ls | head
[Symbols]
addr=0x0021a610  off=0x0021a610  ord=114  fwd=NONE  sz=8  bind=GLOBAL  type=OBJEC
name=stdout
addr=0x0021a600  off=0x0021a600  ord=115  fwd=NONE  sz=0  bind=GLOBAL  type=NOTYPE
name=_edat a
addr=0x0021b388  off=0x0021b388  ord=116  fwd=NONE  sz=0  bind=GLOBAL  type=NOTYPE
name=_end
addr=0x0021a600  off=0x0021a600  ord=117  fwd=NONE  sz=8  bind=GLOBAL  type=OBJECT
name=__prog
name
addr=0x0021a630  off=0x0021a630  ord=119  fwd=NONE  sz=8  bind=UNKNOWN  type=OBJECT
name=prog
ram_invocation_name
addr=0x0021a600  off=0x0021a600  ord=121  fwd=NONE  sz=0  bind=GLOBAL  type=NOTYPE
name=__bss_
start
addr=0x0021a630  off=0x0021a630  ord=122  fwd=NONE  sz=8  bind=GLOBAL  type=OBJECT
name=__prog
name_full
addr=0x0021a600  off=0x0021a600  ord=123  fwd=NONE  sz=8  bind=UNKNOWN  type=OBJECT
name=prog
ram_invocation_short_name
addr=0x00002178 off=0x00002178     ord=124  fwd=NONE  sz=0  bind=GLOBAL  type=FUNC
name=_init
```

با -r هسته Radare می‌تواند به صورت خودکار تمام این سمبول هـا را پرچـم کـرده و توابـع و بلـوک های داده را تعریف کند.

```
$ rabin2 -sr /bin/ls
fs symbols
cd 8 @ 0x0021a610
f sym.stdout      8     0x0021a610
f sym._edata      0     0x0021a600
f sym._end  0     0x0021b388
cd 8 @ 0x0021a600
f sym.__progname 8 0x0021a600
cd 8 @ 0x0021a630
f sym.program_invocation_name 8 0x0021a630
f sym.__bss_start 0 0x0021a600
```

کتابخانه‌ها
Rabin2 می‌تواند با استفاده از پرچم -l کتابخانه‌های استفاده شده توسط یک فایل باینری را لیسـت کند.

```
$ rabin2 -l /bin/ls
[Linked libraries]
libselinux.so.1
```

```
librt.so.1
libacl.so.1
libc.so.6
4 libraries
```

اگر شما خروجی 'rabin2 -l' و 'ldd' را مقایسه کنید شما متوجه خواهید شد که rabin کتابخانه‌های کمتری را در مقایسه با 'ldd' لیست خواهد کرد.دلیل آن این است که rabin وابستگی‌های کتابخانه ای لیست شده را همراهی نمی‌کند و فقط آن‌هایی که در فایل باینری لیست شده‌اند را نمایش خواهد داد.

رشته‌ها
پرچم -z برای لیست کردن تمام رشته‌های قرار داده شده در بخش .rodata برای فایل‌های باینری ELF و .text برای PE به کار می رود.

```
$ rabin2 -z /bin/ls |head
addr=0x00012487 off=0x00012487 ordinal=000 sz=9 len=9 section=.rodata type=A
string=src/ls.c
addr=0x00012490 off=0x00012490 ordinal=001 sz=26 len=26 section=.rodata type=A
string=sort_typ
E != sort_version
addr=0x000124aa   off=0x000124aa    ordinal=002 sz=5  len=5 section=.rodata
type=A string=    %lu
addr=0x000124b0   off=0x000124b0    ordinal=003 sz=7  len=14
      section=.rodata type=W string=%*lu ?
addr=0x000124ba   off=0x000124ba    ordinal=004 sz=8  len=8 section=.rodata
type=A string=%s  %*s
addr=0x000124c5   off=0x000124c5    ordinal=005 sz=10 len=10
      section=.rodata type=A string=%*s, %*
s
addr=0x000124cf   off=0x000124cf    ordinal=006 sz=5  len=5 section=.rodata
type=A string=    ->
addr=0x000124d4   off=0x000124d4    ordinal=007 sz=17 len=17
      section=.rodata type=A string=cannot
access      %s
addr=0x000124e5   off=0x000124e5    ordinal=008 sz=29 len=29
      section=.rodata type=A string=cannot     r
Ead symbolic link %s
addr=0x00012502   off=0x00012502    ordinal=009 sz=10 len=10
      section=.rodata type=A string=unlabel ed
```

با استفاده از -r تمام این اطلاعات به دستورات Radare2 تبدیل خواهند شد که یک فضای پرچم با نام "strings" را ایجاد می‌کند که با پرچم ها برای تمام این رشته‌ها پر شده است.علاوه بر این آن‌ها را به جای کد به عنوان رشته پالایش می کند.

```
$ rabin2 -zr /bin/ls |head
fs strings
f str.src_ls.c    9     @     0x00012487
cs 9 @ 0x00012487
f str.sort_type__sort_version 26    @     0x00012490
cs 26 @ 0x00012490
f str._lu 5 @ 0x000124aa
cs 5 @ 0x000124aa
f str.__lu_ 14 @ 0x000124b0
cs 7 @ 0x000124b0
f str._s__s 8 @ 0x000124ba
```

```
(...)
```

بخش‌های برنامه
Rabin2 به ما اطلاعات کامل در مورد بخش‌های برنامه را می دهد.همانطور که در مثال بعد خواهیم دید ما می‌توانیم شاخص، آفست، اندازه، چینش، نوع و مجوز آن‌ها را بدانیم.

```
$ rabin2 -S /bin/ls
[Sections]
idx=00      addr=0x00000238   off=0x00000238 sz=28 vsz=28  perm=-r--
name=.interp
idx=01      addr=0x00000254   off=0x00000254 sz=32 vsz=32  perm=-r--
name=.note.ABI_tag
idx=02      addr=0x00000274   off=0x00000274 sz=36 vsz=36  perm=-r--
name=.note.gnu.build_id
idx=03      addr=0x00000298   off=0x00000298 sz=104  vsz=104 perm=-r--
name=.gnu.hash
idx=04      addr=0x00000300   off=0x00000300 sz=3096 vsz=3096 perm=-r--
name=.dynsym
idx=05      addr=0x00000f18   off=0x00000f18 sz=1427 vsz=1427 perm=-r--
      name=.dynstr
idx=06      addr=0x000014ac   off=0x000014ac sz=258  vsz=258 perm=-r--
name=.gnu.version
idx=07      addr=0x000015b0   off=0x000015b0 sz=160  vsz=160 perm=-r--
name=.gnu.version_r
idx=08      addr=0x00001650   off=0x00001650 sz=168  vsz=168 perm=-r--
name=.rela.dyn
idx=09      addr=0x000016f8   off=0x000016f8 sz=2688 vsz=2688 perm=-r--
name=.rela.plt
idx=10      addr=0x00002178   off=0x00002178 sz=26 vsz=26 perm=-r-x name=.init
idx=11      addr=0x000021a0   off=0x000021a0 sz=1808 vsz=1808    perm=-r-x
name=.plt
idx=12      addr=0x000028b0   off=0x000028b0 sz=64444 vsz=64444 perm=-r-x
name=.text
idx=13      addr=0x0001246c   off=0x0001246c sz=9    vsz=9 perm=-r-x name=.fini
idx=14      addr=0x00012480   off=0x00012480 sz=20764 vsz=20764 perm=-r--
name=.rodata
idx=15      addr=0x0001759c   off=0x0001759c sz=1820 vsz=1820 perm=-r--
name=.eh_frame_hdr
idx=16      addr=0x00017cb8   off=0x00017cb8 sz=8460 vsz=8460    perm=-r--
name=.eh_frame
idx=17      addr=0x00019dd8   off=0x00019dd8 sz=8    vsz=8 perm=-rw-
name=.init_array
idx=18      addr=0x00019de0   off=0x00019de0 sz=8    vsz=8 perm=-rw-
name=.fini_array
idx=19      addr=0x00019de8   off=0x00019de8 sz=8    vsz=8 perm=-rw- name=.jcr
idx=20      addr=0x00019df0   off=0x00019df0 sz=512 vsz=512 perm=-rw-
name=.dynamic
idx=21      addr=0x00019ff0   off=0x00019ff0 sz=16 vsz=16 perm=-rw- name=.got
idx=22      addr=0x0001a000   off=0x0001a000 sz=920 vsz=920 perm=-rw-
name=.got.plt
idx=23      addr=0x0001a3a0   off=0x0001a3a0 sz=608 vsz=608 perm=-rw-
name=.data
idx=24      addr=0x0001a600   off=0x0001a600 sz=3464 vsz=3464 perm=-rw-
name=.bss
idx=25      addr=0x0001a600   off=0x0001a600 sz=8     vsz=8 perm=----
name=.gnu_debuglink
idx=26      addr=0x0001a608   off=0x0001a608 sz=254 vsz=254 perm=----
      name=.shstrtab
27 sections
```

همچنین با استفاده -r برنامه Radare می‌تواند شـروع و پایـان هـر بخـش را پرچـم گـذاری می‌کنـد و همچنین هر بخش را با اطلاعات قبلی توضیح گذاری می کند.

```
$ rabin2 -Sr /bin/ls
Fs sections
S 0x00000238      0x00000238 0x0000001c 0x0000001c .interp 4
F section..interp 28 0x00000238
F section_end..interp 0 0x00000254
CC [00] va=0x00000238 pa=0x00000238 sz=28 vsz=28 rwx=-r-- .interp @ 0x00000238
S 0x00000254      0x00000254 0x00000020 0x00000020 .note.ABI_tag 4
F section..note.ABI_tag 32 0x00000254
F section_end..note.ABI_tag   0 0x00000274
CC [01] va=0x00000254 pa=0x00000254 sz=32 vsz=32 rwx=-r--  .note.ABI_tag    @
0x00000254
S 0x00000274 0x00000274 0x00000024 0x00000024 .note.gnu.build_id 4
F section..note.gnu.build_id 36 0x00000274
F section_end..note.gnu.build_id 0 0x00000298
CC [02] va=0x00000274 pa=0x00000274 sz=36 vsz=36 rwx=-r-- .note.gnu.build_id @
0x00000274
S 0x00000298 0x00000298 0x00000068 0x00000068 .gnu.hash 4
F section..gnu.hash 104 0x00000298
F section_end..gnu.hash 0 0x00000300
CC [03] va=0x00000298 pa=0x00000298 sz=104 vsz=104 rwx=-r-- .gnu.hash @
0x00000298
S 0x00000300 0x00000300 0x00000c18 0x00000c18 .dynsym 4
F section..dynsym 3096 0x00000300
F section_end..dynsym 0 0x00000f18
CC [04] va=0x00000300 pa=0x00000300 sz=3096 vsz=3096 rwx=-r-- .dynsym @
0x00000300
S 0x00000f18 0x00000f18 0x00000593 0x00000593 .dynstr 4
F section..dynstr 1427 0x00000f18
F section_end..dynstr   0 0x000014ab
CC [05] va=0x00000f18 pa=0x00000f18 sz=1427 vsz=1427 rwx=-r-- .dynstr @
0x00000f18
S 0x000014ac 0x000014ac 0x00000102 0x00000102 .gnu.version 4
F section..gnu.version  258 0x000014ac
F section_end..gnu.version 0 0x000015ae (...)
```

# Rasm2

Rasm2

اسمبلر/دی اسمبلر خطی.در ابتدا rasm بـرای وصـله هـای بـاینری و گرفتـن بـایت هـای یـک opcode مشخص طراحی شده بود .در اینجا کمک وجود دارد :

```
$ rasm2 -h
Usage:      rasm2 [-CdDehLBvw] [-a arch] [-b bits] [-o addr] [-s syntax]
        [-f file] [-F fil:ter] [-i skip] [-l len] 'code'|hex|-
-a [arch]         Set architecture to assemble/disassemble (see -L)
-b [bits]         Set cpu register size (8, 16, 32, 64) (RASM2_BITS)
-c [cpu]          Select specific CPU (depends on arch)
-C                Output in C format
-d, -D            Disassemble from hexpair bytes (-D show hexpairs)
-e                Use big endian instead of little endian
-f [file]         Read data from file
-F [in:out]       Specify input and/or output filters (att2intel,x86.pseudo)
-h                Show this help
-i [len]          ignore/skip N bytes of the input buffer
-k [kernel]       Select operating system (linux, windows, darwin,     ..)
-l [len]          Input/Output length
-L                List supported asm plugins
-o [offset]       Sets tart address for code (default 0)
-O [file]         Output file name (rasm2 -Bf a.asm -O a)
-s [syntax]       Select syntax (intel, att)
-B                Binary input/output (-l is mandatory for binary input)
-v                Show version information
-w                What's this instruction for? Describe opcode
If '-l' value is greater than output length, output is padded with nops
If the last argument is '-' reads from stdin
```

Asm از پلاگین هایی پشتیبانی می‌کند که می توانند با -L لیست شوند

```
$ rasm2 -L
_d 16         8051 pd 8051 intel cpu
_d 16 32      arc gpl3 argonaut risc core
ad 16 32 64   arm gpl3 acorn risc machine cpu
_d 16 32 64   arm.cs bsd capstone arm disassembler
_d 16 32      arm.winedbg lgpl2 winedbg's arm disassembler
_d 16 32      avr gpl avr atmel
ad 32         bf lgpl3 brainfuck
_d 16         cr16 lgpl3 cr16 disassembly plugin
_d 16         csr pd cambridge silicon radio (csr)
ad 32 64      dalvik lgpl3 androidvm dalvik
ad 16         dcpu16 pd mojang's dcpu-16
_d 32 64      ebc lgpl3 efi bytecode
_d 8          gb lgpl3 gameboy(tm) (z80-like)
_d 16         h8300 lgpl3 h8/300 disassembly   plugin
_d 8          i8080 bsd intel 8080 cpu
ad 32         java apache java bytecode
_d 32         m68k bsd motorola 68000
_d 32         malbolge lgpl3 malbolge ternary vm
ad 32 64      mips gpl3 mips cpu
_d 16 32 64   mips.cs bsd capstone mips disassembler
_d 16 32 64   msil pd .net microsoft intermediate language
_d 32         nios2 gpl3 nios ii embedded processor
_d 32 64      ppc gpl3 powerpc
_d 32 64      ppc.cs bsd capstone powerpc disassembler
ad rar        lgpl3 rar vm
_d 32         sh gpl3 superh-4 cpu
_d 32 64      sparc gpl3 scalable processor architecture
_d 32       t ms320      lgplv3 tms320 dsp family
```

```
_d 32       ws lgpl3 whitespace esotheric vm
_d 16 32 64 x86 bsd udis86 x86-16,32,64
_d 16 32 64 x86.cs bsd capstone x86 disassembler
a_ 32 64    x86.nz lgpl3 x86 handmade assembler
ad 32       x86.olly gpl2 ollydbg x86 disassembler
ad 8        z80 nc-gpl2 zilog z80
```

اسمبل
استفاده از rasm2 از پوسته بسیار شایع است.این یک ابزار بسیار خوب برای کپی کردن و چسباندن هگزادسیمال ها است که نشان دهنده opcode هستند.

```
$ rasm2 -a x86 -b 32 'mov eax, 33'
b821000000
$ echo 'push eax;nop;nop' | rasm2 -f -
5090
```

Rasm2 از هسته Radare برای نوشتن بایت ها با استفاده از دستور wa استفاده می کند.
اسـمبل کـردن بـرای X86(گرامـر اینتـل)، olly(گرامـر olly)، arm، powerpc و java امکان‌پـذیر است.برای گرامر اینتل Rasm تلاش می‌کند تا از NASM یـا GAS اسـتفاده کنـد.شـما بـا اسـتفاده از متغیر محیطی SYNTAX می‌توانید گرامر مورد علاقه خود را انتخاب کنید :intel یا att.
در دایرکتوری منبع Rasm چند مثال وجود دارد که یک فایل خام را با استفاده از rasm از یک فایل که در این opcodeها شرح داده شده است اسمبل می کند.

```
$ cat selfstop.rasm
;
; Self-Stop shellcode written in rasm for x86
;
; --pancake
;
.arch x86
.equ base 0x8048000
.org 0x8048000    ; the offset where we inject the 5 byte jmp
selfstop:
            push  0x8048000
            pusha
            mov   eax,  20
            int   0x80
            mov   ebx,  eax
            mov   ecx,  19
            mov   eax,  37
            int   0x80
            popa
            ret
;
;     The   call  injection
;
            ret
[0x00000000]> e asm.bits = 32
[0x00000000]> wx `!rasm2 -f a.rasm`
[0x00000000]> pd 20
                            0X00000000 6800800408          push  0x8048000    ;
0x08048000
                      0x00000005          60    pushad
                      0x00000006          b814000000  mov   eax,      0x14     ;
0x00000014
                      0x0000000b          cd80        int   0x80
                      syscall[0x80][0]=?  0x0000000d  89c3 mov ebx, eax
                      0x0000000f          b913000000  mov   ecx,      0x13     ;
```

```
0x00000013
                    0x00000014          b825000000  mov eax, 0x25 ; 0x00000025

                    0x00000019          cd80        int 0x80
                    syscall[0x80][0]=?  0X0000001b  61 popad
                    0x0000001c          c3    ret
                    0x0000001d          c3    ret
```

دیس اسمبل
به همان شیوه ای که اسمبلر rasm کـار می‌کنـد، بـا اسـتفاده از پرچـم -d شـما می‌توانـد یـک رشـته هگزادسیمال را دیس اسمبل کنید :

```
$ rasm2 -a x86 -b 32 -d '90'
nop
```

# تجزیه و تحلیل

تجزیه و تحلیل
سه دستور مختلف برای تجزیه و تحلیل داده‌ها و کد و استخراج اطلاعات شبیه به اشاره گرها، ارجاعات رشته ای، بلوک های اولیه، استخراج اطلاعات opcode، اطلاعات پرش، xrefs و غیره وجود دارد.
این عملیات توسط دستور a که از analysis گرفته شده است انجام می‌شوند :

```
|Usage: a[?adfFghoprsx]
| a8 [hexpairs]         analyze bytes
| aa                    analyze all (fcns + bbs)
| ad                    analyze data trampoline (wip)
| ad [from] [to]        analyze data pointers to (from-to)
| ae [expr]             analyze opcode eval expression (see ao)
| af[rnbcsl?+-*]        analyze Functions
| aF                    same as above, but using graph.depth=1
| ag[?acgdlf]           output Graphviz code
| ah[?lba-]             analysis hints (force opcode size, ...)
| ao[e?]    [len]       analyze Opcodes (or emulate it)
| ap                    find and analyze function preludes
| ar[?ld-*]             manage refs/xrefs (see also afr?)
| as [num]              analyze syscall using dbg.reg
| at[trd+-*?] [.]       analyze execution Traces
|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
```

تجزیه و تحلیل کد
تجزیه و تحلیل کد یک تکنیک معمول برای استخراج اطلاعات از کد اسمبلی است.Radare چندین ساختار داده داخلی را برای شناسایی بلوک های اولیه، درخت توابع، استخراج اطلاعات سطح opcode و غیره ذخیره می کند.
یک دستور رایج تجزیه و تحلیل radare2 همانند زیر استفاده می‌شود :

```
[0x08048440]> aa
[0x08048440]> pdf @ main

       ; DATA XREF from 0x08048457 (entry0)
/ (fcn) fcn.08048648 141
|           ;-- main:
|     0x08048648 8d4c2404     lea   ecx,  [esp+0x4]
|     0x0804864c 83e4f0       and   esp,  0xfffffff0
|     0x0804864f ff71fc       push  dword [ecx-0x4]
|     0x08048652 55           push  ebp
|     ; CODE (CALL) XREF from 0x08048734 (fcn.080486e5)
|     0x08048653 89e5         mov ebp, esp
|     0x08048655 83ec28       sub   esp,  0x28
|     0x08048658 894df4       mov   [ebp-0xc],  ecx
|     0x0804865b 895df8       mov   [ebp-0x8],  ebx
|     0x0804865e 8975fc       mov   [ebp-0x4],  esi
|     0x08048661 8b19         mov   ebx,  [ecx]
|     0x08048663 8b7104       mov   esi,  [ecx+0x4]
|     0x08048666 c744240c000. mov   dword [esp+0xc],  0x0
|     0x0804866e c7442408010. mov   dword [esp+0x8],  0x1   ; 0x00000001
            0X08048676 c7442404000. mov   dword [esp+0x4],  0x0
|     0x0804867e c7042400000. mov   dword [esp],      0x0
|     0x08048685 e852fdffff   call  sym..imp.ptrace
|     sym..imp.ptrace(unk,    unk)
|     0x0804868a 85c0         test  eax,  eax
|     ,=< 0x0804868c 7911           jns   0x804869f
```

```
|     | 0x0804868e c70424cf870.     mov   dword [esp],
      str.Don_tuseadebuguer_ ; 0x080487cf||     | 0x08048695 e882fdffff call
      sym..imp.puts
|     | sym..imp.puts()
|     | 0x0804869a e80dfdffff call  sym..imp.abort
|     | sym..imp.abort()
              `-> 0x0804869f 83fb02  cmp   ebx,  0x2
|      ,==< 0x080486a2 7411   je    0x80486b5
|     |0x080486a4 c704240c880.      mov   dword [esp],
str.Youmustgiveapasswordforusethisprogram_ ; 0
x0804880c
|     | 0x080486ab e86cfdffff call sym..imp.puts
|     | sym..imp.puts()
|     | 0x080486b0 e8f7fcffff call sym..imp.abort
|     | sym..imp.abort()
|     `--> 0x080486b5 8b4604  mov   eax,  [esi+0x4]
|     0x080486b8 890424       mov   [esp],      eax
|     0x080486bb e8e5feffff   call  fcn.080485a5
|     fcn.080485a5() ; fcn.080484c6+223
|     0x080486c0 b800000000   mov   eax,  0x0
|     0x080486c5 8b4df4       mov   ecx,  [ebp-0xc]
|     0x080486c8 8b5df8       mov   ebx,  [ebp-0x8]
|     0x080486cb 8b75fc       mov   esi,  [ebp-0x4]
|     0x080486ce 89ec         mov   esp,  ebp
|     0x080486d0 5d           pop   ebp
|     0x080486d1 8d61fc       lea   esp,  [ecx-0x4]
\     0x080486d4 c3           ret
```

# Rahash2

Rahash2
محاسبه کردن یک هش کنترلی از بلوک جاری با استفاده از دستور # بسیار آسان است.

```
$ radare2 /bin/ls
[0x08049790]> bf entry0
[0x08049790]> #md5
d2994c75adaa58392f953a448de5fba7
```

همچنین به همین روش شما می‌توانید الگوریتم های هش دیگر که توسط rahash پشتیبانی می‌شوند را محاسبه کنید : md4, md5, crc16, crc32, sha1, sha256, sha384, sha512, par, xor, xorpair, mod255, hamdist, entropy, all

دستور # می‌تواند یک آرگومان عددی را قبول کند که طول بایت هایی که هش شده‌اند را تعریف می کند.

```
[0x08049A80]> #md5 32
9b9012b00ef7a94b5824105b7aaad83b
[0x08049A80]> #md5 64
a71b087d8166c99869c9781e2edcf183
[0x08049A80]> #md5 1024
a933cc94cd705f09a41ecc80c0041def
[0x08049A80]>
```

ابزار Rahash2
ابزار rahash توسط Radare به منظور تحقق بخشیدن به این محاسبات مورد استفاده قرار می گیرد.

```
$ rahash2 -h
Usage:rahash2 [-rBhLkv] [-b sz] [-a algo] [-s str] [-f from] [-t to] [file] ...
-a algo           comma separated list of algorithms (default is 'sha256')
-b bsize          specify the size of the block (instead of full file)
-B                show per-block hash
-e                swap endian (use little endian)
-f from           start hashing at given address
-i num            repeat hash N iterations
-S seed           use given seed (hexa or s:string) use ^ to prefix
-k                show hash using the openssh's randomkey algorithm
-q                run in quiet mode (only show results)
-L                list all available algorithms (see -a)
-r                output radare commands
-s string         hash this string instead of files
-t to             stop hashing at given address
-v                show version information
```

این اجازه محاسبه هش از رشته‌ها یا فایل‌ها را می دهد.

```
$ rahash2 -q -a md5 -s 'hello world'
5eb63bbbe01eeed093cb22bb8f5acdc3
```

همچنین هش کامل تمام محتویات یک فایل نیز ممکن است اما این را برای فایل‌های بزرگ مانند دیسک ها انجام ندهید زیرا rahash قبل از محاسبه کنترلی به جای اینکه این کار را به تدریج انجام دهد بافر را در حافظه ذخیره می کند.

```
$ rahash2 -a all /bin/ls
/bin/ls:    0x00000000-0x0001ae08   md5:  b5607b4dc7d896c0fab5c4a308239161
/bin/ls:    0x00000000-0x0001ae08   sha1:
      c8f5032c2dce807c9182597082b94f01a3bec495
sha256: 978317d58e3ed046305df92a19f7    0x00000000-0x0001ae08    bin/ls:/
 d3e0bfcb 3c70cad 979f24fe e289ed1d266b0
/bin/ls:    0x00000000-0x0001ae08   sha384:
9e946efdbebb4e0ca00c86129ce2a71ee734ac30 b620336c38
1aa929dd222709e4cf7a800b25fbc7d06fe3b184933845
/bin/ls:    0x00000000-0x0001ae08   sha512:
076806cedb5281fd15c21e493e12655c55c5253
7fc1f36e641b57648f7512282c03264cf5402b1b15cf03a20c9a60edfd2b4f76d4905fcec777c29
7d3134f41f
/bin/ls:    0x00000000-0x0001ae08   crc16:      4b83
/bin/ls:    0x00000000-0x0001ae08   crc32:      6e316348
/bin/ls:    0x00000000-0x0001ae08   md4:  3a75f925a6a197d26bc650213f12b074
/bin/ls:    0x00000000-0x0001ae08   xor:  3e
/bin/ls:    0x00000000-0x0001ae08   xorpair:    59
/bin/ls:    0x00000000-0x0001ae08   parity:     01
/bin/ls:    0x00000000-0x0001ae08   entropy:    0567f925
/bin/ls:    0x00000000-0x0001ae08   hamdist:    00
/bin/ls:    0x00000000-0x0001ae08   pcprint:    23
/bin/ls:    0x00000000-0x0001ae08   mod255:     1e
/bin/ls:    0x00000000-0x0001ae08   xxhash:     138c936d
/bin/ls:    0x00000000-0x0001ae08   adler32:    fca7131b
```

# اشکال زدا

اشکال زدا
اشکال زدا در radare به عنوان پلاگین IO پیاده‌سازی می شود.این برای ساخت یا اتصال به یک فرایند دو آدرس مختلف را اداره می‌کند : dbg:// و pid://
برنامه‌های مختلفی برای معماری های مختلف و سیستم عامل ها مانند GNU/Linux, Windows, MacOSX, (Net,Free,Open)BSD و Solaris وجود دارد.
فرایند حافظه توسط Radare به عنوان یک فایل ساده تفسیر می شود.بنابراین تمام صفحه های نگاشت شده همانند برنامه‌ها و کتابخانه‌ها می‌توانند به عنوان کد، ساختار و غیره تفسیر شوند.
بقیه ارتباط بین Radare و لایه اشکال زدا توسط فراخوان system() پوشانده می‌شود که یک رشته را به عنوان آرگومان دریافت کرده و دستور گرفته شده را اجرا می کند.نتیجه عملیات در کنسول خروجی بافر می‌شود و این محتوا می‌تواند توسط یک زبان اسکریپت نویسی به کار گرفته شود.
به همین دلیل است که Radare می‌تواند برای فراخوانی system() یک و دو علامت تعجب را به کار بگیرد.

```
[0x00000000]> ds
[0x00000000]> !!ls
```

دو علامت تعجب به Radare می‌گویند که از لیست پلاگین برای پیدا کردن یک پلاگین IO که این دستور را به کار می‌گیرد صرف نظر کن و آن را به طور مستقیم در پوسته اجرا کن.تنها یک علامت لیست کردن پلاگین IO را انجام می دهد.

دستورات اشکال زدا معمولاً بین معماری و سیستم عامل قابل حمل هستند.اما radare تلاش می‌کند که آن‌ها را برای تمام معماری ها و سیتم عامل ها در شل کد تزریق کند یا استثناها را در یک روش خاص به کار گیرد.به عنوان مثال در mips هیچ ویژگی گام به گامی توسط سخت‌افزار وجود ندارد و بنابراین radare با استفاده از ترکیبی از تجزیه و تحلیل کد و نقاط شکست نرم افزاری برای دور زدن این محدودیت پیاده‌سازی مربوط به خود را دارد.
برای دریافت کمک اولیه از اشکال زدا می‌توانید d? را تایپ کنید :

```
Usage:      d[sbhcrbo] [arg]
dh [handler]      list or set debugger handler
dh [handler]      transplant process to a new handler
dd                file descriptors (!fd in r1)
ds[ol] n          step, over, source line
do                open process (reload, alias for 'oo')
dk [sig][=act] list,   send, get, set, signal handlers of child
di[s] [arg..]     inject code on running process and execute it (see gs)
dp[=*?t][pid]     list, attach to process or thread id
dc[?]             continue execution. dc? for more
dr[?]             cpu registers, dr? for extended help
db[?]             breakpoints
dbt               display backtrace
dt[?r] [tag]      display instruction traces (dtr=reset)
dm[?*]            show memory maps
dw [pid]          block prompt until pid dies
```