

R2 Basics

Maxime Morin

2014-10-22

GroundWork

r2 basics

Each command is associated to a single letter. The rest are subcommands.

- p? => print display help on print commands)
- pd => print disassembly of the current block size
- pd? => print disassembly help
- pdf => print disassembly of a function

```
[0x00404890]> 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) 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  
[0x00404890]> █
```

Figure 1: p? => print display help on print commands)

```

[0x00404890]> pd 15
;-- entry0:
0x00404890 31ed      xor ebp, ebp
0x00404892 4989d1    mov r9, rdx
0x00404895 5e        pop rsi
0x00404896 4889e2    mov rdx, rsp
0x00404899 4883e4f0  and rsp, 0xfffffffffffffff0
0x0040489d 50        push rax
0x0040489e 54        push rsp
0x0040489f 49c7c0d01e4. mov r8, 0x411ed0 ; 0x00411ed0
0x004048a6 48c7c1601e4. mov rcx, 0x411e60 ; 0x00411e60
0x004048ad 48c7c7c0284. mov rdi, 0x4028c0 ; main
0x004048b4 e837dcffff call sym.imp.__libc_start_main
0x004024f0(unk, unk, unk) ; sym.imp.__libc_start_main
0x004048b9 f4        hlt
0x004048ba 660f1f440000 o16 nop [rax+rax]
0x004048c0 b8ffa56100 mov eax, 0x61a5ff ; 0x0061a5ff
0x004048c5 55        push rbp
[0x00404890]>

```

Figure 2: pd => print disassembly of the current block size

```

[0x00404890]> pdf
/ (fcn) entry0 42
|
| 0x00404890 31ed      xor ebp, ebp
| 0x00404892 4989d1    mov r9, rdx
| 0x00404895 5e        pop rsi
| 0x00404896 4889e2    mov rdx, rsp
| 0x00404899 4883e4f0  and rsp, 0xfffffffffffffff0
| 0x0040489d 50        push rax
| 0x0040489e 54        push rsp
| 0x0040489f 49c7c0d01e4. mov r8, 0x411ed0 ; 0x00411ed0
| 0x004048a6 48c7c1601e4. mov rcx, 0x411e60 ; 0x00411e60
| 0x004048ad 48c7c7c0284. mov rdi, 0x4028c0 ; main
| 0x004048b4 e837dcffff call sym.imp.__libc_start_main
| sym.imp.__libc_start_main(unk, unk)
\ 0x004048b9 f4        hlt
[0x00404890]>

```

Figure 3: pdf => print disassembly of a function

To get help on any command just append ? example:

```
[0x00404890]> w?
| Usage: w[x] [str] [<File] [<E0F] [@addr]
| wc          list all write changes
| w[1248][+-][n] increment/decrement byte,word...
| 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)
| wd [off] [n] duplicate N bytes from offset at current seek (memcpy) (see y?
| )
| 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)
[0x00404890]>
```

Figure 4: w? => display help on print commands

More help type ? to get the main help:

- man radare2, radare2 -h (same with the other tools)
- ????: Help on Expressions
- \$??: Help on Variables
- ?@?: Help on Offset

Hashing: Fingerprint for a sample (#)

Hashing is a common method used to uniquely identify malware. The malicious software is run through a hashing program that produces a unique hash that identifies that Malware (like a fingerprint). MD5, SHA1, SHA512 are the most commonly used. The fingerprint will be used for research and sharing instead of sharing the binary. It can also be used for researching over the Internet to see if the file has already been identified.

To calculate the hash of a program you can either use `r2` or the stand-alone program `rahash2`

Rahash2

- Display list of algorithm available `rahash2 -L`
- Calculate the sha1 `rahash2 -a sha1 program.exe`

Radare2

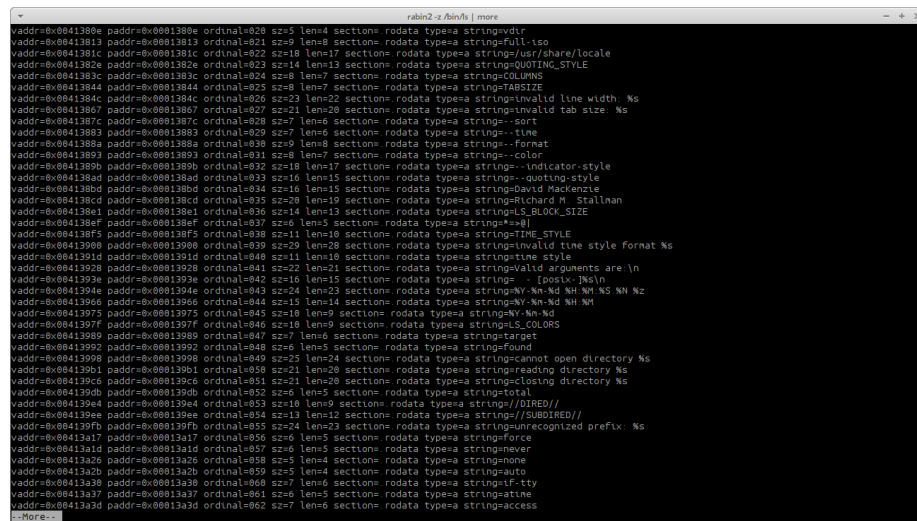
- Display list of algorithm available `[0x00404888]>##`
- Calculate the sha1 `[0x00404888]>#sha1 $s @ 0 // Compute md5 (#md5) of size of file ($s) at offset 0`

Quick strings fetching

A string in a file is a sequence of characters such as “Abracadabra!”. Searching through the strings can give some information about the functionality of a program.

To quickly display the strings contained in a binary you can use either r2 or rabin2: which is the dedicated command to get information about binaries.

- Rabin2
 - Display strings inside .data section (like gnu strings does) `rabin2 -zz file`



```
rabin2 -zz file
vaddr=0x0041380e paddr=0x0001380e ordinal=020 sz=5 len=4 section= rodata type=a string=vdtr
vaddr=0x00013813 paddr=0x00013813 ordinal=021 sz=9 len=8 section= rodata type=a string=full-tao
vaddr=0x0001381c paddr=0x0001381c ordinal=022 sz=18 len=17 section= rodata type=a string=/usr/share/locale
vaddr=0x0001382e paddr=0x0001382e ordinal=023 sz=14 len=13 section= rodata type=a string=QUOTINC_STYLE
vaddr=0x0001383c paddr=0x0001383c ordinal=024 sz=8 len=7 section= rodata type=a string=COLUMNS
vaddr=0x00013844 paddr=0x00013844 ordinal=025 sz=8 len=7 section= rodata type=a string=TAGSIZE
vaddr=0x0001384c paddr=0x0001384c ordinal=026 sz=23 len=22 section= rodata type=a string=invalid line width: %s
vaddr=0x00013867 paddr=0x00013867 ordinal=027 sz=21 len=20 section= rodata type=a string=invalid tab size: %s
vaddr=0x0001387c paddr=0x0001387c ordinal=028 sz=7 len=6 section= rodata type=a string=-sort
vaddr=0x00013883 paddr=0x00013883 ordinal=029 sz=7 len=6 section= rodata type=a string=-time
vaddr=0x0001388a paddr=0x0001388a ordinal=030 sz=9 len=8 section= rodata type=a string=-format
vaddr=0x00013893 paddr=0x00013893 ordinal=031 sz=8 len=7 section= rodata type=a string=-color
vaddr=0x0001389b paddr=0x0001389b ordinal=032 sz=18 len=17 section= rodata type=a string=-indicator-style
vaddr=0x000138ad paddr=0x000138ad ordinal=033 sz=16 len=15 section= rodata type=a string=-quoting-style
vaddr=0x000138bd paddr=0x000138bd ordinal=034 sz=16 len=15 section= rodata type=a string=David MacKenzie
vaddr=0x000138cd paddr=0x000138cd ordinal=035 sz=28 len=19 section= rodata type=a string=Richard W. Stallman
vaddr=0x000138e1 paddr=0x000138e1 ordinal=036 sz=14 len=13 section= rodata type=a string=L5_BLOCK_SIZE
vaddr=0x000138ef paddr=0x000138ef ordinal=037 sz=6 len=5 section= rodata type=a string==e|
vaddr=0x000138f5 paddr=0x000138f5 ordinal=038 sz=11 len=10 section= rodata type=a string=TIME_STYLE
vaddr=0x00013908 paddr=0x00013908 ordinal=039 sz=20 len=18 section= rodata type=a string=invalid time style format %s
vaddr=0x0001391d paddr=0x0001391d ordinal=040 sz=11 len=10 section= rodata type=a string=--time style
vaddr=0x0001392b paddr=0x0001392b ordinal=041 sz=22 len=21 section= rodata type=a string=valid arguments are: \n
vaddr=0x0001392e paddr=0x0001392e ordinal=042 sz=16 len=15 section= rodata type=a string= [posix] %s\n
vaddr=0x0001394e paddr=0x0001394e ordinal=043 sz=24 len=23 section= rodata type=a string=NY-Nd NH NS NN Nz
vaddr=0x00013966 paddr=0x00013966 ordinal=044 sz=15 len=14 section= rodata type=a string=NY-Nd NH NH
vaddr=0x00013975 paddr=0x00013975 ordinal=045 sz=10 len=9 section= rodata type=a string=NY-Nd
vaddr=0x0001397f paddr=0x0001397f ordinal=046 sz=10 len=9 section= rodata type=a string=L5_COLORS
vaddr=0x00013989 paddr=0x00013989 ordinal=047 sz=7 len=6 section= rodata type=a string=target
vaddr=0x00013992 paddr=0x00013992 ordinal=048 sz=6 len=5 section= rodata type=a string=found
vaddr=0x00013998 paddr=0x00013998 ordinal=049 sz=25 len=24 section= rodata type=a string=cannot open directory %s
vaddr=0x000139b1 paddr=0x000139b1 ordinal=050 sz=21 len=20 section= rodata type=a string=reading directory %s
vaddr=0x000139c6 paddr=0x000139c6 ordinal=051 sz=21 len=20 section= rodata type=a string=closing directory %s
vaddr=0x000139db paddr=0x000139db ordinal=052 sz=6 len=5 section= rodata type=a string=total
vaddr=0x000139e4 paddr=0x000139e4 ordinal=053 sz=18 len=9 section= rodata type=a string=//DIRENT//
vaddr=0x000139ee paddr=0x000139ee ordinal=054 sz=13 len=12 section= rodata type=a string=//SUBDIR//
vaddr=0x000139fb paddr=0x000139fb ordinal=055 sz=24 len=23 section= rodata type=a string=unrecognized prefix: %s
vaddr=0x00013a17 paddr=0x00013a17 ordinal=056 sz=6 len=5 section= rodata type=a string=force
vaddr=0x00013a1d paddr=0x00013a1d ordinal=057 sz=6 len=5 section= rodata type=a string=never
vaddr=0x00013a26 paddr=0x00013a26 ordinal=058 sz=5 len=4 section= rodata type=a string=none
vaddr=0x00013a2b paddr=0x00013a2b ordinal=059 sz=5 len=4 section= rodata type=a string=auto
vaddr=0x00013a38 paddr=0x00013a38 ordinal=060 sz=7 len=6 section= rodata type=a string=if-ty
vaddr=0x00013a37 paddr=0x00013a37 ordinal=061 sz=6 len=5 section= rodata type=a string=atune
vaddr=0x00013a3d paddr=0x00013a3d ordinal=062 sz=7 len=6 section= rodata type=a string=access
vaddr=0x00013a3d paddr=0x00013a3d ordinal=062 sz=7 len=6 section= rodata type=a string=access
```

Figure 5: iz

- Display strings from raw bins `rabin2 -zz file`
- Append j to get the result in json format!

`rabin2 -zj file`

```

rabin2 -z /bin/ls | more
('strings' [(('vaddr' 76856, 'paddr' 76856, 'length' 11, 'size' 12, 'type' 'ascii', 'string' 'dev_tno_pdp'), ('vaddr' 76936, 'paddr' 76936, 'length' 12, 'size' 13, 'type' 'ascii', 'string' 'format_inode'), ('vaddr' 76949, 'paddr' 76949, 'length' 10, 'size' 11, 'type' 'ascii', 'string' 'sort_files'), ('vaddr' 76968, 'paddr' 76968, 'length' 6, 'size' 7, 'type' 'ascii', 'string' 'posix.'), ('vaddr' 76967, 'paddr' 76967, 'length' 4, 'size' 5, 'type' 'ascii', 'string' 'nain'), ('vaddr' 79656, 'paddr' 79656, 'length' 10, 'size' 11, 'type' 'ascii', 'string' 'pcdb_lswd'), ('vaddr' 79676, 'paddr' 79676, 'length' 8, 'size' 9, 'type' 'ascii', 'string' 'src/lsc'), ('vaddr' 79679, 'paddr' 79679, 'length' 25, 'size' 26, 'type' 'ascii', 'string' 'sort_type = sort_version'), ('vaddr' 79765, 'paddr' 79765, 'length' 4, 'size' 5, 'type' 'ascii', 'string' 'xlu'), ('vaddr' 79710, 'paddr' 79710, 'length' 5, 'size' 6, 'type' 'ascii', 'string' 'xlu'), ('vaddr' 79721, 'paddr' 79721, 'length' 7, 'size' 8, 'type' 'ascii', 'string' 'xlu'), ('vaddr' 79732, 'paddr' 79732, 'length' 9, 'size' 10, 'type' 'ascii', 'string' 'xlu'), ('vaddr' 79742, 'paddr' 79742, 'length' 4, 'size' 5, 'type' 'ascii', 'string' 'xlu'), ('vaddr' 79747, 'paddr' 79747, 'length' 10, 'size' 11, 'type' 'ascii', 'string' 'cannot access %s'), ('vaddr' 79764, 'paddr' 79764, 'length' 20, 'size' 21, 'type' 'ascii', 'string' 'cannot read symbolic link %s'), ('vaddr' 79793, 'paddr' 79793, 'length' 9, 'size' 10, 'type' 'ascii', 'string' 'unlabeled'), ('vaddr' 79807, 'paddr' 79807, 'length' 22, 'size' 23, 'type' 'ascii', 'string' '/nReport %s bugs to %s/n'), ('vaddr' 79826, 'paddr' 79826, 'length' 21, 'size' 22, 'type' 'ascii', 'string' 'bug'), ('vaddr' 79848, 'paddr' 79848, 'length' 19, 'size' 20, 'type' 'ascii', 'string' 'Ns home page -%s/n'), ('vaddr' 79868, 'paddr' 79868, 'length' 13, 'size' 14, 'type' 'ascii', 'string' 'OU coreutils'), ('vaddr' 79886, 'paddr' 79886, 'length' 2, 'size' 3, 'type' 'ascii', 'string' 'vdr'), ('vaddr' 79891, 'paddr' 79891, 'length' 8, 'size' 9, 'type' 'ascii', 'string' 'full-iso'), ('vaddr' 79908, 'paddr' 79908, 'length' 17, 'size' 18, 'type' 'ascii', 'string' '/usr/share/locale'), ('vaddr' 79918, 'paddr' 79918, 'length' 13, 'size' 14, 'type' 'ascii', 'string' 'QUOTING_STYLE'), ('vaddr' 79932, 'paddr' 79932, 'length' 7, 'size' 8, 'type' 'ascii', 'string' 'COLL'), ('vaddr' 79948, 'paddr' 79948, 'length' 7, 'size' 8, 'type' 'ascii', 'string' 'TABSIZE'), ('vaddr' 79948, 'paddr' 79948, 'length' 22, 'size' 23, 'type' 'ascii', 'string' 'invalid line width %s'), ('vaddr' 79975, 'paddr' 79975, 'length' 20, 'size' 21, 'type' 'ascii', 'string' 'invalid tab size %s'), ('vaddr' 79990, 'paddr' 79990, 'length' 6, 'size' 7, 'type' 'ascii', 'string' 'sort'), ('vaddr' 80003, 'paddr' 80003, 'length' 6, 'size' 7, 'type' 'ascii', 'string' 'time'), ('vaddr' 80010, 'paddr' 80010, 'length' 8, 'size' 9, 'type' 'ascii', 'string' 'format'), ('vaddr' 80019, 'paddr' 80019, 'length' 7, 'size' 8, 'type' 'ascii', 'string' 'color'), ('vaddr' 80027, 'paddr' 80027, 'length' 17, 'size' 18, 'type' 'ascii', 'string' 'indicator-style'), ('vaddr' 80045, 'paddr' 80045, 'length' 15, 'size' 16, 'type' 'ascii', 'string' 'quota'), ('vaddr' 80061, 'paddr' 80061, 'length' 15, 'size' 16, 'type' 'ascii', 'string' 'David Mackenzie'), ('vaddr' 80077, 'paddr' 80077, 'length' 19, 'size' 20, 'type' 'ascii', 'string' 'Richard M. Stallman'), ('vaddr' 80097, 'paddr' 80097, 'length' 13, 'size' 14, 'type' 'ascii', 'string' 'LS_BLOCK_SIZE'), ('vaddr' 80111, 'paddr' 80111, 'length' 5, 'size' 6, 'type' 'ascii', 'string' 'wall'), ('vaddr' 80117, 'paddr' 80117, 'length' 10, 'size' 11, 'type' 'ascii', 'string' 'TIME_STYLE'), ('vaddr' 80128, 'paddr' 80128, 'length' 20, 'size' 21, 'type' 'ascii', 'string' 'invalid time style format %s'), ('vaddr' 80157, 'paddr' 80157, 'length' 10, 'size' 11, 'type' 'ascii', 'string' 'time'), ('vaddr' 80168, 'paddr' 80168, 'length' 21, 'size' 22, 'type' 'ascii', 'string' 'valid arguments are /n'), ('vaddr' 80198, 'paddr' 80198, 'length' 13, 'size' 14, 'type' 'ascii', 'string' 'posix %s/n'), ('vaddr' 80206, 'paddr' 80206, 'length' 23, 'size' 24, 'type' 'ascii', 'string' 'Ny-Nd NH-NM NS-MN-Nz'), ('vaddr' 80230, 'paddr' 80230, 'length' 14, 'size' 15, 'type' 'ascii', 'string' 'Ny-Nd NH-NM'), ('vaddr' 80245, 'paddr' 80245, 'length' 9, 'size' 10, 'type' 'ascii', 'string' 'Ny-Nd'), ('vaddr' 80255, 'paddr' 80255, 'length' 9, 'size' 10, 'type' 'ascii', 'string' 'LS_COLORS'), ('vaddr' 80265, 'paddr' 80265, 'length' 6, 'size' 7, 'type' 'ascii', 'string' 'target'), ('vaddr' 80274, 'paddr' 80274, 'length' 5, 'size' 6, 'type' 'ascii', 'string' 'found'), ('vaddr' 80280, 'paddr' 80280, 'length' 24, 'size' 25, 'type' 'ascii', 'string' 'cannot open directory %s'), ('vaddr' 80309, 'paddr' 80309, 'length' 20, 'size' 21, 'type' 'ascii', 'string' 'reading directory %s'), ('vaddr' 80320, 'paddr' 80320, 'length' 20, 'size' 21, 'type' 'ascii', 'string' 'closing directory %s'), ('vaddr' 80347, 'paddr' 80347, 'length' 5, 'size' 6, 'type' 'ascii', 'string' 'total'), ('vaddr' 80356, 'paddr' 80356, 'length' 9, 'size' 10, 'type' 'ascii', 'string' 'DIRED/'), ('vaddr' 80366, 'paddr' 80366, 'length' 12, 'size' 13, 'type' 'ascii', 'string' 'DIRED/'), ('vaddr' 80379, 'paddr' 80379, 'length' 23, 'size' 24, 'type' 'ascii', 'string' 'unrecognized prefix %s'), ('vaddr' 80407, 'paddr' 80407, 'length' 5, 'size' 6, 'type' 'ascii', 'string' 'force'), ('vaddr' 80413, 'paddr' 80413, 'length' 5, 'size' 6, 'type' 'ascii', 'string' 'never'), ('vaddr' 80422, 'paddr' 80422, 'length' 4, 'size' 5, 'type' 'ascii', 'string' 'auto'), ('vaddr' 80432, 'paddr' 80432, 'length' 6, 'size' 7, 'type' 'ascii', 'string' 'tty'), ('vaddr' 80439, 'paddr' 80439, 'length' 5, 'size' 6, 'type' 'ascii', 'string' 'atime'), ('vaddr' 80445, 'paddr' 80445, 'length' 6, 'size' 7, 'type' 'ascii', 'string' 'access'), ('vaddr' 80456, 'paddr' 80456, 'length' 5, 'size' 6, 'type' 'ascii', 'string' 'ctime'), ('vaddr' 80462, 'paddr' 80462, 'length' 6, 'size' 7, 'type' 'ascii', 'string' 'status'), ('vaddr' 80469, 'paddr' 80469, 'length' 9, 'size' 10, 'type' 'ascii', 'string' 'extension'), ('vaddr' 80479, 'paddr' 80479, 'length' 7, 'size' 8, 'type' 'ascii', 'string' 'verbose'), ('vaddr' 80487, 'paddr' 80487, 'length' 4, 'size' 5, 'type' 'ascii', 'string' 'long'), ('vaddr' 80492, 'paddr' 80492, 'length' 6, 'size' 7, 'type' 'ascii', 'string' 'connect'), ('vaddr' 80499, 'paddr' 80499, 'length' 10, 'size' 11, 'type' 'ascii', 'string' 'horizontal'), ('vaddr' 80510, 'paddr' 80510, 'length' 6, 'size' 7, 'type' 'ascii', 'string' 'across'), ('vaddr' 80517, 'paddr' 80517, 'length' 8, 'size' 9, 'type' 'ascii', 'string' 'vertical'), ('vaddr' 80526, 'paddr' 80526, 'length' 13, 'size' 14, 'type' 'ascii', 'string' 'single-column'), ('vaddr' 80540, 'paddr' 80540, 'length' 6, 'size' 7, 'type' 'ascii', 'string' 'escape'), ('vaddr' 80547, 'paddr' 80547, 'length' 9, 'size' 10, 'type' 'ascii', 'string' 'directory'), ('vaddr' 80557, 'paddr' 80557, 'length' 3, 'size' 4, 'type' 'ascii', 'string' 'dired'), ('vaddr' 80563, 'paddr' 80563, 'length' 9, 'size' 10, 'type' 'ascii', 'string' 'full-time'), ('vaddr' 80573, 'paddr' 80573, 'length' 23, 'size' 24, 'type' 'ascii', 'string' 'dired')])

```

Figure 6: izj

- Radare2
 - Display all strings in r2 [0x00404888]>izz

Rabin2 and Radare2 can display both ASCII and Unicode strings: See `type=a` or `type=u`.

Suffix:

```

~ or grep grep/cut interno
| pipe to program
> pipe to file
>> concat to a
@ temporal seek
@@ iterator
* output in commands
j output in json
? help

```

Get information about a binary (i?)

We've seen how to parse a binary or any file format to modify or retrieve information at a low-level. You can also retrieve information using info command `i?`:

- Get General information about the binary: `iI // rabin2 -I`
- Get Header information `ih // rabin2 -H`
- Get Imports: `ii // rabin2 -i`
- Get Entrypoints: `ie // rabin2 -e`
- Get Exports: `is // rabin2 -s`
- Get Relocs: `iR // rabin2 -R`
- Get Sections: `iS // rabin2 -S`

```
[0x00404f3e]> iI
file      /home/maijin/Documents/ch22.exe
type      EXEC (Executable file)
pic       true
canary    false
nx        true
crypto    false
has_va    true
root      pe
class     PE32
lang      msil
arch      x86
bits      32
machine   i386
os        windows
subsys    Windows GUI
endian    little
strip     true
static    false
linenum   false
lsyms     false
relocs    false
rpath     NONE
[0x00404f3e]> 
```

Figure 7: Get General information about the binary: `iI/Rabin2 -I`

Parse a File format

File Format definition:

A file format is a standard way that information is encoded for storage in a computer file. It specifies how bits are used to encode information in a digital storage medium.

Portable Executable definition:

The Portable Executable (PE) format is a file format for executable files, object code, DLLs, (...) used by Windows operating systems.

- [PE101 by Corkami](#)
- [Portable Executable header](#)

Pf: print formatted data

Usage: pf[.key[.field[=value]]][val]][times][[size] format] [arg0 arg1 ...]

Examples:

```
pf 10xiz pointer length string
pf {array_size}b @ array_base
pf [4]w[7]i      # like pf w..i... 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)
b - byte (unsigned)
B - resolve enum bitfield (see t?) `pf B (Bitfield_type)arg_name`
c - char (signed byte)
E - resolve enum name (see t?) `pf E (Enum_type)arg_name`
X - show n hexpairs (default n=1) 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)
T - show Ten first bytes of buffer
d - 0x%08x hexadecimal value (4 bytes)
D - disassemble one opcode
o - 0x%08o octal value (4 byte)
x - 0x%08x hexadecimal value and flag (fd @ addr)
```

```

X - show formatted hexpairs
z - \0 terminated string
Z - \0 terminated wide string
s - 32bit pointer to string (4 bytes)
S - 64bit pointer to string (8 bytes)
? - data structure `pf ? (struct_type)struct_name`
* - next char is pointer (honors asm.bits)
+ - toggle show flags for each offset
: - skip 4 bytes
. - skip 1 byte

```

1. Look at the structure defined in .h or any valuable documentation about a file format

```

typedef struct _ IMAGE_DOS_HEADER {          // DOS .EXE header
    WORD    e_magic;                          // Magic number
    WORD    e_cblp;                           // Bytes on last page of file
    WORD    e_cp;                             // Pages in file
    WORD    e_crlc;                           // Relocations
    WORD    e_cparhdr;                        // Size of header in paragraphs
    WORD    e_minalloc;                      // Minimum extra paragraphs needed
    WORD    e_maxalloc;                      // Maximum extra paragraphs needed
    WORD    e_ss;                            // Initial (relative) SS value
    WORD    e_sp;                             // Initial SP value
    WORD    e_csum;                          // Checksum
    WORD    e_ip;                             // Initial IP value
    WORD    e_cs;                             // Initial (relative) CS value
    WORD    e_lfarlc;                        // File address of relocation table
    WORD    e_ovno;                          // Overlay number
    WORD    e_res[4];                        // Reserved words
    WORD    e_oemid;                         // OEM identifier (for e_oeminfo)
    WORD    e_oeminfo;                      // OEM information; e_oemid specific
    WORD    e_res2[10];                     // Reserved words
    LONG    e_lfanew;                       // File address of new exe header
} IMAGE_DOS_HEADER, * PIMAGE_DOS_HEADER;

```

2. Convert each component type in pf symbol equivalent, for example first is **WORD e_magic**;: * **WORD** is w. * **e_ident** should contain the Magic number: A constant numerical or text value used to identify a file format. In PE, this magic number is a magic text ('MZ'), So we can also display/parse it like a string of size 2 [2]z.

```
w e_magic` or `[2]z e_magic
```

3. Set this new type in pf just using: pf.dos_header [2]z e_magic

To try that new type and parse a pe to retrieve the MZ magic:

1. Open an pe file: `r2 *.exe`
2. Do not forget to set the type: `pf.dos_header [2]z e_magic`
3. Run stored format at offset 0 of the elf file and profit: `pf.dos_header @ 0`
4. Retrieve a single value: `pf.dos_header.e_magic @ 0`

```
[0x00000000]> pf.dos_header [2]z e_magic
[0x00000000]> pf.dos_header @ 0
e_magic : 0x00000000 = MZ
[0x00000000]> █
```

Figure 8: Run stored format at offset 0 of the elf file and profit

The complete dos_header could be done like this:

```
pf.pe_dos_header [2]zwww[4]www[10]wx
e_magic e_cblp e_cp e_crlc e_cparhdr e_minalloc
e_maxalloc e_ss e_sp e_csum e_ip e_cs e_lfarlc
e_ovno e_res e_oemid e_oeminfo e_res2 e_lfanew
```

You can contribute on this part to implement PE, ELF and Mach-O in r2. This work has already started for elf and Mach-o, just open a elf file using `r2 -nn` which means only load the rbin structures and profit.

```
pf.pe_dos_header @ pe_dos_header
pf.pe_nt_image_headers32 @ pe_nt_image_headers32
```

or

```
pf.pe_nt_image_headers64 @ pe_nt_image_headers64
```

Packing

Packer definition:

Packers are wrappers put around pieces of software to compress and/or encrypt their contents. They can be used by legitimate software to minimise download times and storage space or to protect copyrighted coding, but are commonly used in malware to disguise the contents of malicious files from malware scanners. Runtime packers essentially unpack (i.e. decrypt or decompress) executable files as they run - the first stage is the unwrapping process, and the unpacked file is then loaded into memory and run. A file can be packed numerous times with slight changes to the packing method, or with small and insignificant changes to the file inside, thus producing a final file which appears different from another identical file packed differently.

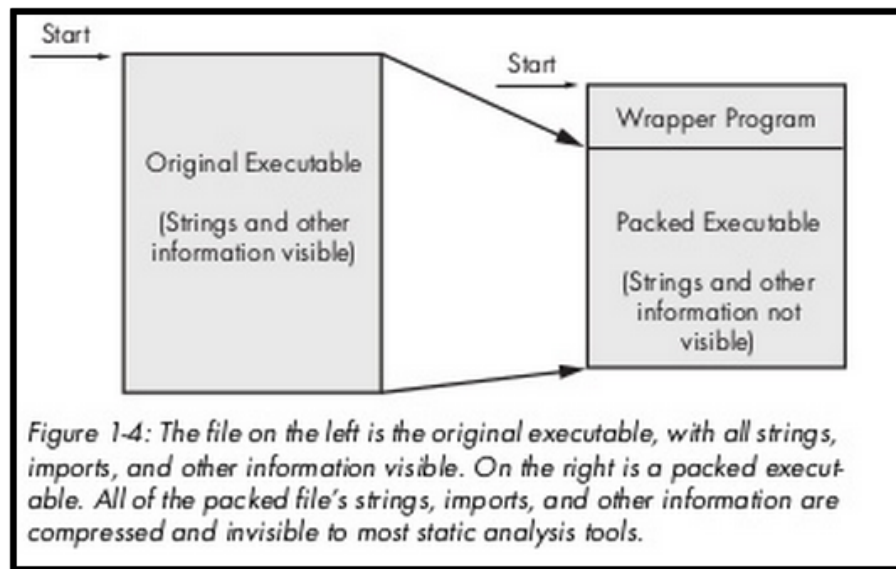


Figure 9: Packer

Detect Packing: Entropy

Entropy as it relates to digital information is the measurement of randomness in a given set of values (data).

<http://www.forensickb.com/2013/03/file-entropy-explained.html>

Entropy can be used in many different ways, but quite commonly to detect encryption and compression, since truly random data is not common in typical user data. This is especially true with executable files that have purposely been encrypted with a real-time decryption routine. This prevents an AV engine from seeing “inside” the executable as it sits on the disk in order to detect strings or patterns. It is also very helpful in identifying files that have a high-amount of randomness, which could indicate an encrypted container/volume that may go otherwise unnoticed.

- Entropy of this file using: `#entropy $s @ 0`
- Entropy block by block using: `p=`
- Entropy Section using rabin2: `rabin2 -K entropy -S /bin/ls`



Figure 10: Entropy block by block using: `p=`

Detect Packing: Yara and signatures

YARA is a tool aimed at (but not limited to) helping malware researchers to identify and classify malware samples. With YARA you can create descriptions of malware families (or whatever you want to describe) based on textual or binary patterns. Each description, a.k.a rule, consists of a set of strings and a Boolean expression which determine its logic. Let's see an example:

You can apply Yara rule inside r2

To use yara just type: `yara scan`. This command will apply the several rules shipped with radare2 (yara list to display the list of rules). You can use yours using `yara add`

Crypto Algorithm

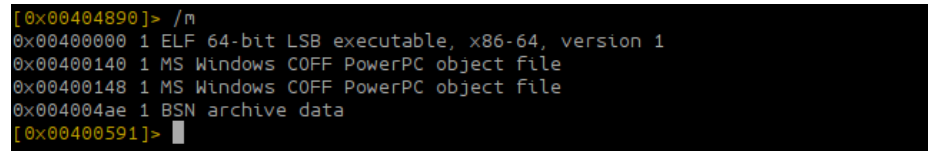
Detection of some cryptographic algorithm are implemented:

```
/Ca Search for AES keys  
/Cr Search for private RSA keys
```

Magic number

You can search for magic number (constant numerical or text value used to identify a file format or protocol; for files) using `/m`. You can restrict the search from a certain offset using eval variable.

```
e search.from=0 // To set beginning address  
e search.to=0x1000 // To set ending address
```



```
[0x00404890]> /m  
0x00400000 1 ELF 64-bit LSB executable, x86-64, version 1  
0x00400140 1 MS Windows COFF PowerPC object file  
0x00400148 1 MS Windows COFF PowerPC object file  
0x004004ae 1 BSN archive data  
[0x00400591]> █
```

Figure 11: Search for magic using `/m`

GO GO GO

First steps

To disassemble a program using r2, first open a binary using `r2 file.exe` command.

Radare2 can perform analysis on a binary in order to get function name and so on. You can launch this analysis using `aa` for analyse all or launch the analysis when opening the file directly: `r2 -A file.exe`

Each command is associated to a single letter. The rest are subcommands.

```
px  print hex
pd  print disassembly
pD  print disassembly (takes the number of bytes instead of the number of opcodes.)
pdf print disassembly of a function
pc  output in C
pcp output in Python
afl list functions
axf xref from
axt xref to
s   seek
?d  Describe opcode
wx 9090 write two intel nops
wo? write in block with operation (wox xor, woA and...)
...
```

Basic print commands

One of the key features of radare is displaying information in various formats. The goal is to offer a selection of displaying choices to best interpret binary data.

Binary data can be represented as integers, shorts, longs, floats, timestamps, hexpair strings, or more complex formats like C structures, disassembly, decompilations, external processors, ..

Here's a list of the available print modes listable using `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) 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)
```



```

[0x00400591]> px 4
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00400591 0000 0000
[0x00400591]> px 2
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00400591 0000
[0x00400591]> pd 1
0x00400591 0000 add [rax], al
[0x00400591]> pc 1
#define _BUFFER_SIZE 1
unsigned char buffer[1] = {
    0x00, };
[0x00400591]> pcp 1
import struct
buf = struct.pack ("1B",
0x00)
[0x00400591]> ?d add
adds src and dst, stores result on dst
[0x00400591]>

```

Figure 12: Print commands

Hexadecimal User-friendly way:

```

[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...

```

Show hexadecimal words dump (32bit)

```

[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 0xffff46690 0x662e0f1f .(@..?....f.f...

```

8bit hexpair list of bytes

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

Show hexadecimal quad-words dump (64bit)

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

Date formats The current supported timestamp print modes are:

```
[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
```

For example, you can ‘view’ the current buffer as timestamps in ntfs time:

```
[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
```

As you can see, the endianness affects the print formats. Once you have printed a timestamp you can grep the results by the year for example:

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

The default date format can be configured using the `cfg.datefmt` variable. The field definitions follow the well-known `strftime(3)` format.

Source (asm, C) Valid print code formats are:

```
pc      C
pcs     string
pcj     json
pcJ     javascript
pcp     python
pcw     words (4 byte)
pcd     dwords (8 byte)

[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\x8d
\x24\xc4\x29\xc2\x52\x48\x89\xd6\x49\x89
\xe5\x48\x83\xe4\xf0\x48\x8b\x3d\x06\x1a"
```

Strings Strings are probably one of the most important entry points when starting to reverse engineer a program because they are usually referencing information about the functions actions (asserts, debug or info messages, ...).

Therefore radare supports various string formats:

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

Most strings will be zero-terminated. Here's an example by using the debugger to continue the execution of the program until it executes the 'open' syscall. When we recover the control over the process, we get the arguments passed to the syscall, pointed by %ebx. In the case of the 'open' call, this parameter is a zero terminated string which we can inspect using **psz**.

```

[0x4A13B8C0]> dcs open
0x4a14fc24 syscall(5) open ( 0x4a151c91 0x00000000 0x00000000 ) = 0xffffffffda
[0x4A13B8C0]> dr
    eax  0xffffffffda    esi  0xffffffff    eip    0x4a14fc24
    ebx  0x4a151c91     edi  0x4a151be1    oeax   0x00000005
    ecx  0x00000000     esp  0xbfbdb1c    eflags 0x200246
    edx  0x00000000     ebp  0xbfbdbb0    cPaZstIdor0 (PZI)
[0x4A13B8C0]>
[0x4A13B8C0]> psz @ 0x4a151c91
/etc/ld.so.cache

```

Disassembly The `pd` command is used to disassemble code. It accepts a numeric value to specify how many opcodes should be disassembled. The `pd` command is similar but instead of a number of instructions it decompiles a given number of bytes.

```

d: disassembly N opcodes    count of opcodes
D: asm.arch disassembler    bsize bytes

```

```

[0x00404888]> pd 1
    ;-- entry0:
    0x00404888    31ed            xor ebp, ebp

```

Selecting the architecture The architecture flavour for the disassembly is defined by the `asm.arch` eval variable. You can use `e asm.arch = ?` to list all available architectures.

```

[0xB7F08810]> e asm.arch = ?

```

There are also multiple options that can be used to configure the output of the disassembler, all these options are described using `e? asm`. See also Eval Variable chapter.

The `syntax` variable is used to influence the flavour of assembly syntax the disassembler engine outputs.

```

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

```

You can also check `asm.pseudo` which is an experimental pseudocode view and `asm.esil` which outputs ESIL ('Evaluable Strings Intermediate Language'). It aims to output a human readable representation of every opcode. Those representations can be evaluated in order to emulate the code.

XREF in radare2

Cross references (XREF) can help us determine where certain functions were called from.

In radare2, xref are displayed in disassembly like this:

```
|          ; DATA XREF from 0x080484f0 (sub.printf_4ec)
|          ;-- str.Great:
|          0x08048662      .string "Great" ; len=5
```

You can quickly get the xref using `axt @ str.Great` (find data/code references to this address).

Block size, Values and Flags in radare2

Block Size

The block size is the default view size for radare. All commands will work with this constraint, but you can always temporally change the block size just giving a numeric argument to the print commands for example (px 20)

```
[0xB7F9D810]> b? Usage: b[f] [arg] b display current block size b+3 increase
blocksize by 3 b-16 decrement blocksize by 3 b 33 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
```

The `b` command is used to change the block size:

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

The `bf` command is used to change the block size to the one specified by a flag. For example in symbols, the block size of the flag represents the size of the function.

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

You can perform these two operations in a single one (pdf):

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

Values

Values are numbers expressed in various formats:

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

Flags

Flagspaces are groups of flags. Some of them are automatically created by rabin while identifying strings, symbols, sections, etc., and others are updated at runtime like by commands like ‘regs’ (registers) or ‘search’ (search results).

Flags are similar to bookmarks. They represent a certain offset in the file. Flags can be grouped in ‘flag spaces’. A flag space is something like a namespace for flags. They are used to group flags of similar characteristic or type. Some example of flagspaces could be sections, registers, symbols.

To create a flag just type:

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

You can remove a flag by prefixing its name with -. Most commands accept - as argument-prefix as a way to delete items.

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

To switch between or create new flagspaces use the **fs** command:

```
[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 flagospace
[0x4A13B8C0]> f          ; list only flags in symbols flagospace
[0x4A13B8C0]> fs *      ; select all flagspaces
```

You can rename flags with **fr**.

Variables

You can also use variables and `seeks` to build more complex expressions. Here are a few examples:

```

?@?      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)

```

? 1+2 // Do calculus and conversion hex/oct/bin...

You can also perform calculus with the `rax2` standalone tool

```
maljin@maljin-ThinkPad-T440s ~/radare2 master rax2 3+0x80
0x83
maljin@maljin-ThinkPad-T440s ~/radare2 master rax2 -S AB
4142
maljin@maljin-ThinkPad-T440s ~/radare2 master rax2 -k 90203010
+--[0x10302090]---+
|Eo...
|. . .
|   o
|. . .
|   S
|.....
+-----+
```

Figure 13: Rax2 commands

Basic Write commands

Radare can manipulate a loaded binary file in multiple ways. You can resize the file, move and copy/paste bytes, insert new bytes (shifting data to the end of the block or file) or simply overwrite bytes at an address, contents of a file, a widestring or even inline assembling an opcode.

To resize use the `r` command which accepts a numeric argument. A positive value sets the new size to the file. A negative one will strip N bytes from the current seek, down-sizing the file.

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

To write bytes use the `w` command. It accepts multiple input formats like inline assembly, endian-friendly dwords, files, hexpair files, wide strings:

```
[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)
```

Some examples:

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


Write over with operation The `wo` command (write operation) accepts multiple kinds of operations that can be applied on the current block. This is for example a 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 -= subtraction
|  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
```

This way it is possible to implement cipher-algorithms using radare core primitives.

A sample session doing a `xor(90) + addition(01 02)`:

```
[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
```

Basic search commands

A basic search for a plain string in a whole file would be something like:

```
$ 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 //` quiet mode (no prompt) and quit after `-i`

As you can see, radare generates a `hit` flag for each search result found. You can just use the `ps` command to visualise the strings at these offsets in this way:

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

We can also search wide-char strings (the ones containing zeros between each letter) using the `/w` in this way:

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

It is also possible to mix hexadecimal scape sequences in the search string:

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

But if you want to perform an hexadecimal search you will probably prefer an hexpair input with `/x`:

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

Once the search is done, the results are stored in the `search` flag space.

```
[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
```

To remove these flags, you can just use the **f@-hit*** command.

Sometimes while working long time in the same file you will need to launch the last search more than once and you will probably prefer to use the **//** command instead of typing all the string again.

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

Search in assembly If you want to search for a certain type of opcodes you can either use **/c** or **/a**:

```
/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
```

Graph



Figure 14: Graphviz ag \$\$ > a.dot

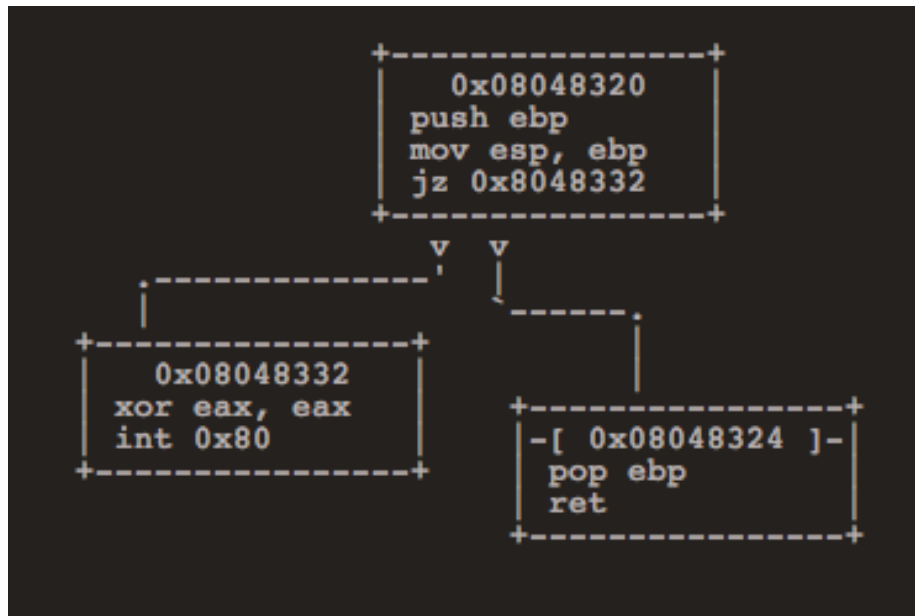


Figure 15: Ascii ART VVV

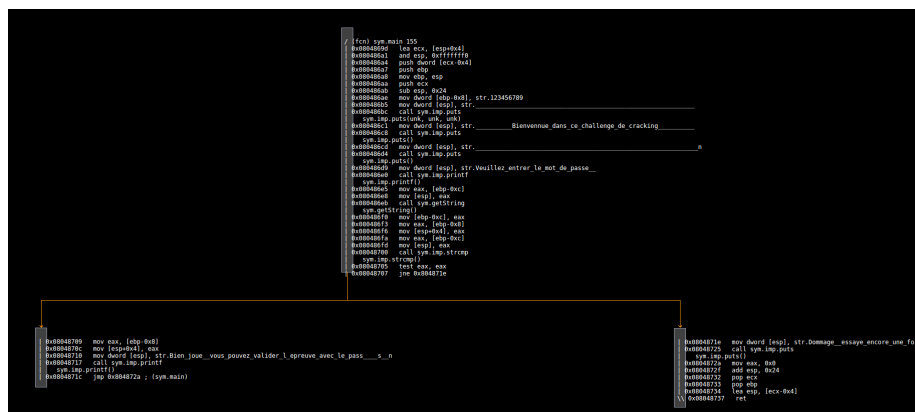


Figure 16: WebView =H , agv (display graph in web-ui)

Visual Mode

- V: Launch the visual mode
- ?: To get help in the visual mode
- d: Define (define code, undefined, rename function) equivalent of ida rightclick define

- ;: Add a comment
- p or P: Switch Visual view
- _: HUD
- u undo/Back to previous screen

```

r2 /bin/ls
[0x00404890 120 /bin/ls]> pd $r @ entry0
;-- entry0:
0x00404890 31ed      xor ebp, ebp
0x00404892 4989d1    mov r9, rdx
0x00404895 5e        pop rsi
0x00404896 4889e2    mov rdx, rsp
0x00404899 4883e4f0  and rsp, 0xfffffffffffffff0
0x0040489d 50        push rax
0x0040489e 54        push rsp
0x0040489f 49c7c0d01e4. mov r8, 0x411ed0 ; 0x00411ed0
0x004048a6 48c7c1601e4. mov rcx, 0x411e60 ; 0x00411e60
0x004048ad 48c7c7c0284. mov rdi, 0x4028c0 ; main
0x004048b4 e837dcffff call syn.imp.__libc_start_main ;[1]
0x004024f0(unk, unk, unk) ; syn.imp.__libc_start_main
0x004048b9 f4        hlt
0x004048ba 660f1f440000 o16 nop [rax+rax]
0x004048c0 b8ffa56100 mov eax, 0x61a5ff ; 0x0061a5ff
0x004048c5 55        push rbp
0x004048c6 482df8a56100 sub rax, 0x61a5f8
0x004048cc 4883f80e   cmp rax, 0xe
0x004048d0 4889e5     mov rbp, rsp
;=< 0x004048d3 7702     ja 0x4048d7 ;[2]
| 0x004048d5 5d        pop rbp
| 0x004048d6 c3        ret

```

Figure 17: Disassembly Visual Mode

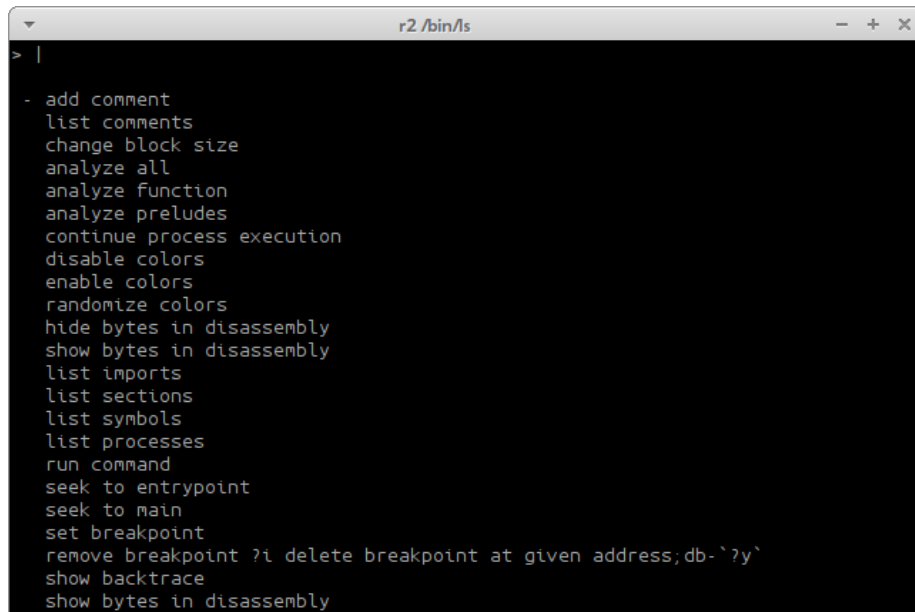


Figure 18: HUD in Visual Mode

Functions in Visual mode You can seek to a symbol or a function typing the number on next to it and get back using u, In this example you can type 3 to seek to `sym.imp.printf` symbol

```
|      ;--main:
|      0x08048330    55          push ebp
|      0x08048331    89e5        mov ebp, esp
|      0x08048333    83ec1c      sub esp, 0x1c
|      0x08048336    53          push ebx
|      0x08048337    c745fc00000. mov dword [ebp-0x4], 0x0
|      0x0804833e    c745f800000. mov dword [ebp-0x8], 0x0
|      0x08048345    686c850408   push str._n_tCrackme_1_by_syscalo_n ; str._n_tCrackme_
|      0x0804834a    e861ffffff   call sym.imp.printf ;[3]
```

You can also display a list a function and quickly navigate between them using v (Visual code analysis manipulation)


```

-[ functions ]-----
(a) add (x)xrefs (q)quit
(n) modify (c)calls (g)go
(d) delete (v)variables (?)help
0x000482c0 (entry0)
* 0x000482a0 (sym.imp.__libc_start_main)
0x000482a6 (fcn.000482a6)
0x00048280 (loc.00048280)
0x0004828c (fcn.0004828c)
0x00048296 (fcn.00048296)
0x000482b0 (sym.imp.printf)
0x000482b6 (fcn.000482b6)
0x000482e2 (fcn.000482e2)
0x00048317 (fcn.00048317)
0x00048327 (sub.printf_327)
0x000484f0 (loc.000484f0)
0x00048504 (fcn.00048504)
0x00048537 (fcn.00048537)
0x00048547 (fcn.00048547)
0x00048562 (section_end..fini)
0x000484ec (sub.printf_4ec)

Visual code analysis manipulation
| | | ; CALL XREF from 0x000482dc (entry0)
| | | / (fcn) sym.imp.__libc_start_main 6
| | | / 0x000482a0 jmp dword [0x00496b8] ; 0x000496b8
| | | / (fcn) fcn.000482a6 10
| | | | 0x000482a6 push 0x8
| | | | 0x000482ab jmp loc.00048280 ;[1]
| | | | ; XREFS: CALL 0x0004834a CALL 0x00048357 CALL 0x00048364
| | | | ; XREFS: CALL 0x00048371 CALL 0x000483b8 CALL 0x000483d6
| | | | ; XREFS: CALL 0x000484d3 CALL 0x000484f5
| | | / (fcn) sym.imp.printf 6
| | | | 0x000482b0 jmp dword [0x00496bc] ; 0x000496bc
| | | / (fcn) fcn.000482b6 10
| | | | 0x000482b6 push 0x10 ; 0x00000010
| | | | 0x000482bb jmp loc.00048280 ;[1]
| | | | ; [11] va=0x000482c0 pa=0x0000002c sz=648 vsz=648 rwx=r-x .text
| | | / (fcn) entry0 34
| | | ;-- section..text:
| | | 0x000482c0 xor ebp, ebp
| | | 0x000482c2 pop esi
| | | 0x000482c3 mov ecx, esp
| | | 0x000482c5 and esp, 0xffffffff
| | | 0x000482c8 push eax
| | | 0x000482c9 push esp
| | | 0x000482ca push edx
| | | 0x000482cb push section..fini ; section..fini
| | | 0x000482d0 push section..init ; section..init
| | | 0x000482d5 push ecx

```

Figure 19: Visual code analysis manipulation

XREF in Visual mode Radare2 implements many user-friendly features for the visual interface to walk thru the assembly code. One of them is the **x** key that popups a menu for selecting the xref (data or code) against the current seek and then jump there. For example when pressing **x** when looking at those 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)

```

After pressing **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)

All the calls and jumps are numbered (1, 2, 3...) these numbers are the keybindings for seeking there from the visual mode. All the seek history is stored, by pressing u key you will go back in the seek history time :)

```

0x08048562  0000      add [eax], al
; [13] va=0x08048564 pa=0x00000564 sz=301 vsz=301 rwx=-r-- .rodata
;-- section..rodata:
0x08048564  0300      add eax, [eax]
0x08048566  0000      add [eax], al
;-- sym..IO_stdin_used:
0x08048568  0100      add [eax], eax
0x0804856a  0200      add al, [eax]
; DATA XREF from 0x08048345 (sub.printf_327)
;-- str._n_tCrackme_1_by_syscalo_n:
0x0804856c  .string "\\n\\tCrackme 1 by syscalo\\n" ; len=24
; DATA XREF from 0x08048352 (sub.printf_327)
;-- str._n_tTROVARE_IL_SERIALE_CORRETTO_n:
0x08048584  .string "\\n\\tTROVARE IL SERIALE CORRETTO\\n" ; len=31
; DATA XREF from 0x0804835f (sub.printf_327)
;-- str._nQuesto_crackme_e_stato_scritto_per_dare_la_possibilita__ai_newbies_di_esercitarsi_c
0x080485a3  .string "\\nQuesto crackme e' stato scritto per dare la possibilita' ai newbies
; DATA XREF from 0x0804836c (sub.printf_327)
;-- str._C_per_terminare_n:
0x0804861a  .string "^C per terminare\\n" ; len=18
; DATA XREF from 0x080483b3 (sub.printf_327)
;-- str.Inserisci_il_seriale_:
0x0804862c  .string "Inserisci il seriale: " ; len=23
; DATA XREF from 0x080483c4 (sub.printf_327)
0x08048643  2578000a00 and eax, 0xa0078
; DATA XREF from 0x080484ce (sub.printf_327)
;-- str.Seriale_errato_riprova_n:
0x08048648  .string "Seriale errato, riprova!\\n" ; len=26
; DATA XREF from 0x080484f0 (sub.printf_4ec)
;-- str.Complimenti_hai_trovato_il_seriale_corretto_n:
0x08048662  .string "Complimenti, hai trovato il seriale corretto!\\n" ; len=47

```

Figure 20: XREF in Visual mode

Let's tweak this interface

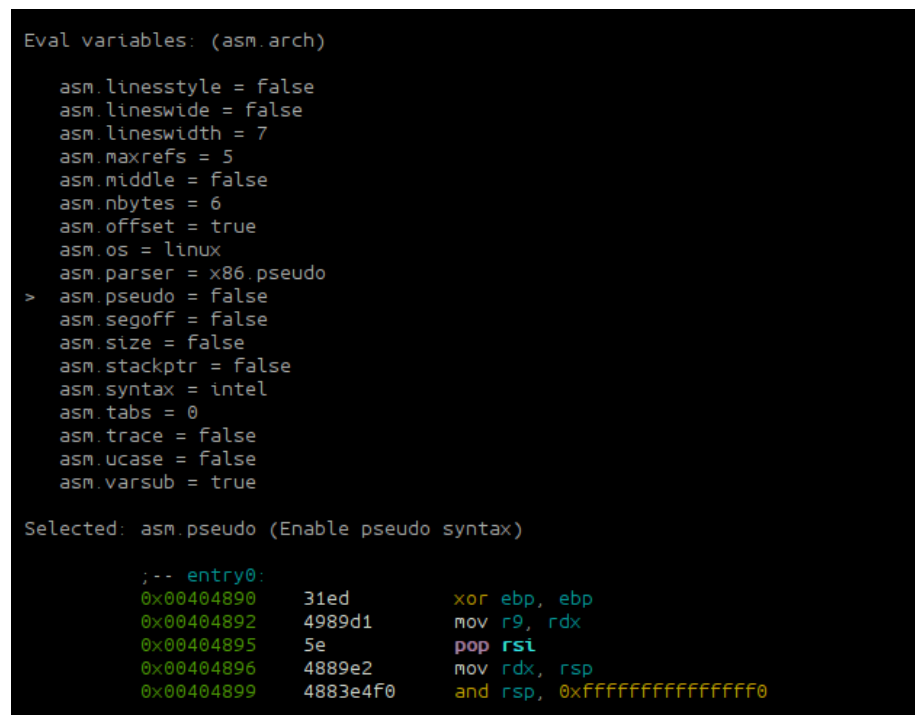
Eval Variable

All the configuration of radare2 is done with the eval command **e** which allows the user to change some variables from an internal hashtable containing string pairs.

These configurations can be also defined using the **-e** flag of radare2 while loading it, so you can setup different initial configurations from the command line.

```
radare2 -e scr.color=false file
```

You can also use the rc file: `~/.radare2rc` There are enhanced interfaces to help users to interactively configure this hashtable. One is **Ve** and provides a shell for walking through the tree and change variables. You can also get list of all variables with description **e??**



```
Eval variables: (asm.arch)

asm.linesstyle = false
asm.lineswide = false
asm.lineswidth = 7
asm.maxrefs = 5
asm.middle = false
asm.nbytes = 6
asm.offset = true
asm.os = linux
asm.parser = x86.pseudo
> asm.pseudo = false
asm.segoff = false
asm.size = false
asm.stackptr = false
asm.syntax = intel
asm.tabs = 0
asm.trace = false
asm.ucase = false
asm.varsub = true

Selected: asm.pseudo (Enable pseudo syntax)

;-- entry0:
0x00404890 31ed xor ebp, ebp
0x00404892 4989d1 mov r9, rdx
0x00404895 5e pop rsi
0x00404896 4889e2 mov rdx, rsp
0x00404899 4883e4f0 and rsp, 0xfffffffffffffff0
```

Figure 21: Ve command

- See the state of an eval variable: **e asm.pseudo**
- Set an eval variable: **e asm.pseudo = true**