

# **Reverse Engineering 101 using Radare2**

**Autor:** Ialle Teixeira

**Blog:** <https://www.linkedin.com/in/isDebuggerPresent/>

## Um pouco da história sobre o Radare2

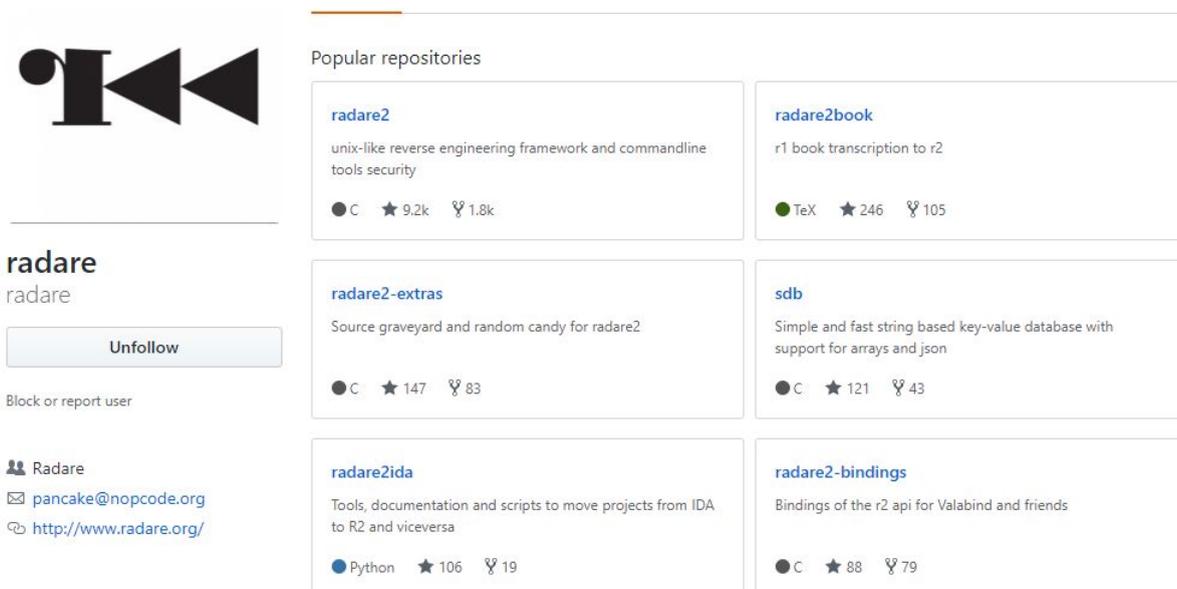
Por volta de 2006, Sergi Álvarez (também conhecido como pancake) estava trabalhando como Analista Forense Computacional. Como ele não podia usar software privado para suas eventuais necessidades pessoais, ele decidiu escrever uma pequena ferramenta, um editor hexadecimal com características muito básicas:

- ser extremamente portátil (amigável, linha de comando, escrito em C, pequeno)
- abrir imagens de disco usando offsets de 64 bits
- procurar por uma string ou hexpair
- revisar e dumpar os resultados para o disco

O editor foi originalmente projetado para recuperar um arquivo excluído de uma partição HFS +. Depois disso, a pancake decidiu estender a ferramenta para se atachar à um processo e algumas funcionalidades de debugging, suporte para múltiplas arquiteturas e análise de código.

Desde então, o projeto evoluiu para fornecer uma estrutura completa para analisar binários, ao mesmo tempo em que utiliza conceitos básicos do UNIX. Esses conceitos incluem os famosos paradigmas "everything is a file", pequenos programas que interagem usando stdin / stdout" e mantendo esses mesmos paradigmas simples.

Em 2009 o radare2 (r2) nasceu como um fork do radare1. O fonte foi totalmente refatorado e inseridos novos recursos dinâmicos. Isso permitiu uma integração muito melhor, abrindo caminho para o uso do r2 a partir de diferentes linguagens de programação. Mais tarde, a API r2pipe permitia acesso ao radare2 a partir de qualquer linguagem, especialmente em python.



**radare**  
radare

Unfollow

Block or report user

Radare  
pancake@nopcode.org  
<http://www.radare.org/>

Popular repositories

- radare2**  
unix-like reverse engineering framework and commandline tools security  
C 9.2k 1.8k
- radare2book**  
r1 book transcription to r2  
TeX 246 105
- radare2-extras**  
Source graveyard and random candy for radare2  
C 147 83
- sdb**  
Simple and fast string based key-value database with support for arrays and json  
C 121 43
- radare2ida**  
Tools, documentation and scripts to move projects from IDA to R2 and viceversa  
Python 106 19
- radare2-bindings**  
Bindings of the r2 api for Valabind and friends  
C 88 79

Executar um processo de engenharia reversa é basicamente ter a capacidade de disassemblar um programa para ver como ele funciona. A engenharia reversa le é usada em coisas como o Malware Analysis para entender o que um malware está fazendo e para criar uma assinatura que ele infecte seu computador novamente. Ele também pode ser usado para corrigir bugs em jogos antigos que você costumava curtir, ou para encontrar um exploit em algum software.

Para instalar o Radare2 você vai precisar clonar o projeto oficial do Github e executar o script sys/install.sh, ele é multiplataforma, você pode usar o Radare2 no Windows, Linux e MacOS:

```
$ git clone https://github.com/radare/radare2
$ cd radare2
$ sudo sys/install.sh
```

Considere o Radare2 como um framework, e dentro dele você terá uma suite de ferramentas integradas da melhor forma possível para agilizar seu processo de engenharia reversa; abaixo algumas informações que devem ser levadas em considerações sobre o framework:

**radare2:** A principal ferramenta de todo o framework. Ele usa o núcleo do editor hexadecimal e debugger.

**rabin2:** Ferramenta utilizada para extrair informações de executáveis como: ELF, PE, Java CLASS, Mach-O, e qualquer outro formato suportado pelos plugins do r2.

**rasm2:** Basicamente um assembler e desassembler em command line para múltiplas arquiteturas, incluindo Intel x86 e x86/64, MIPS, ARM, PowerPC, Java e afins...)

**rahash2:** Uma implementação de uma ferramenta block-based hash. Desde pequenos arquivos de textos, até imagens gigantes, o rahash2 suporta múltiplos algoritmos, incluindo MD4, MD5, CRC16, CRC32, SHA1, SHA256 e outros, você pode usar principalmente para verificar integridade ou acompanhar mudanças do seu binário, dump, disco e etc.

**radiff2:** O radiff2 realiza diff em binários que implementam múltiplos algoritmos, suportando desde byte-level ou delta diffing para binários e análise de código para identificar mudanças em uma parte específica.

**rafind2:** Uma simples ferramenta de byte patterns em arquivos, um bom uso seria usar para identificar padrões entre malwares que você encontrou funções reaproveitadas.

**ragg2:** Simplesmente um frontend para r\_egg(muito usada para software exploitation). Suporte para x86, x86-64, e ARM.

**rarun2:** Um launcher para executar programas em diferentes ambientes, com diferentes argumentos, permissões, diretórios e file descritor padrão substituídos. rarun2 é útil para: resolver crackmes, fuzzing.

**rax2:** Uma ferramenta para executar expressões matemáticas bem minimalista em command line, que é útil para fazer conversões de base entre valores de ponto flutuante, representação hexadecimal, conversão para ASCII, e muito mais.

Agora que sabemos um pouco mais sobre o Radare2 e seus utilitários, vamos ao que interessa. Uma das principais ações que você deve fazer iniciar um processo de eng reversa é saber para qual plataforma o binário foi desenvolvido; como podemos saber isso usando o radare2?! Vamos começar usando a opção -I do rabin2, isso nos dará informações importantes sobre o binário:

```
~rabin2 -I ialle.exe
```

```
ialle@ubuntu:~$ rabin2 -I ialle.exe
arch      x86
binsz    854072
bintype  pe
bits     64
canary   false
class    PE32+
cmp.csum 0x000d315d
compiled Wed Dec 31 16:00:00 1969
crypto   false
endian   little
havecode true
hdr.csum 0x000d315d
linenum  false
lsyms    false
machine  AMD 64
maxopsh 16
minopsh  1
nx       true
os       windows
overlay  true
pcalign  0
pic      true
relocs   false
signed   true
static   false
stripped true
subsys   Windows GUI
va       true
```

Isso nos diz que este programa será executado no Windows, também podemos ver o bintype sendo "PE". Isso é chamado de "magic number" e pode nos ajudar a descobrir que tipo de arquivo é esse (no exemplo, um executável do Windows). Isso é tudo que realmente precisamos saber por agora, o próximo comando que eu gosto de usar é rabin2 -z. Isto irá listar todas as strings da seção de dados do binário. Ao executar o comando, podemos ver algumas strings que podem indicar parâmetros ou funcionalidades do executável:

```
ialle@ubuntu:~$ rabin2 -z ialle.exe | more
000 0x00000660 0x140002060 4 5 (.rdata) ascii `A\|v@
001 0x00000690 0x140002090 5 6 (.rdata) ascii PuTTY
002 0x000006a4 0x1400020a4 24 25 (.rdata) ascii Connecting to %s port %d
003 0x000006bd 0x1400020bd 16 17 (.rdata) ascii Connecting to %s
004 0x000006ce 0x1400020ce 27 28 (.rdata) ascii Failed to connect to %s: %s
005 0x000006ea 0x1400020ea 4 5 (.rdata) ascii %s\r\n
006 0x000006f0 0x1400020f0 12 26 (.rdata) utf16le ../be_misc.c
007 0x0000070a 0x14000210a 8 18 (.rdata) utf16le len >= 2
008 0x0000071c 0x14000211c 9 10 (.rdata) ascii proxy: %s
009 0x00000726 0x140002126 5 6 (.rdata) ascii -load
010 0x0000072c 0x14000212c 4 5 (.rdata) ascii -ssh
011 0x00000731 0x140002131 7 8 (.rdata) ascii -telnet
012 0x00000739 0x140002139 7 8 (.rdata) ascii -rlogin
013 0x00000741 0x140002141 4 5 (.rdata) ascii -raw
014 0x00000746 0x140002146 7 8 (.rdata) ascii -serial
015 0x00000754 0x140002154 8 9 (.rdata) ascii -loghost
016 0x0000075d 0x14000215d 8 9 (.rdata) ascii -hostkey
017 0x00000766 0x140002166 62 63 (.rdata) ascii '%s' is not a valid format for a manual host key specification
018 0x000007af 0x1400021af 47 48 (.rdata) ascii '%c expects at least two colons in its argument
019 0x000007df 0x1400021df 6 7 (.rdata) ascii %c%.*s
020 0x000007f0 0x1400021f0 40 41 (.rdata) ascii -nc expects argument of form 'host:port'
```

Nosso próximo caso é o rasm2, embora seja usado internamente, também é um binário autônomo que você pode usar para executar funções como assembly e desassembler, além de funcionar para diversas arquiteturas como MIPS, no exemplo abaixo:

```
~rasm2 'mov eax, ebx; pop ebp; nop'
```

```
ialle@ubuntu:~$ rasm2 'mov eax, ebx; pop ebp; nop'
89d85d90
ialle@ubuntu:~$ rasm2 -d 89d85d90
mov eax, ebx
pop ebp
nop
ialle@ubuntu:~$ rasm2 -a mips 'addiu a1, a2, 8'
0800c524
ialle@ubuntu:~$ rasm2 -a mips -d 0800c524
addiu a1, a2, 8
ialle@ubuntu:~$ rasm2 -w sqrtpd
compute square roots of packed double-fp values
```

O rahash2, como foi dito no começo, é uma implementação de uma ferramenta de hash block-based. Desde pequenos texto à discos grandes, o rahash2 suporta vários algoritmos, incluindo MD4, MD5, CRC16, CRC32, SHA1, SHA256 e afins. O rahash2 pode ser usado para verificar a integridade ou rastrear alterações de arquivos, e até mesmo dump de memória:

```
~rahash2 ialle.exe -a rc2
```

```
ialle@ubuntu:~$ rahash2 ialle.exe -a rc2
ialle@ubuntu:~$ rahash2 ialle.exe -a md5
ialle.exe: 0x00000000-0x000d0837 md5: 54cb91395cdaad9d47882533c21fc0e9
ialle@ubuntu:~$ rahash2 ialle.exe -a md4
ialle.exe: 0x00000000-0x000d0837 md4: 1d4c6480fece327a1efcfc9a48415515
ialle@ubuntu:~$ rahash2 ialle.exe -a sha1
ialle.exe: 0x00000000-0x000d0837 sha1: 3b1333f826e5fe36395042fe0f1b895f4a373f1b
ialle@ubuntu:~$ rahash2 ialle.exe -a crc64
ialle.exe: 0x00000000-0x000d0837 crc64: d759398c937a94a4
ialle@ubuntu:~$ rahash2 ialle.exe -a xor
ialle.exe: 0x00000000-0x000d0837 xor: 7b
ialle@ubuntu:~$ rahash2 ialle.exe -a xorpair
ialle.exe: 0x00000000-0x000d0837 xorpair: 4b30
ialle@ubuntu:~$ rahash2 ialle.exe -a crc64we
ialle.exe: 0x00000000-0x000d0837 crc64we: 0b0aeb3290622bce
```

O radiff2 sem nenhum parâmetro mostrará por padrão quais bytes foram alterados e os seus correspondentes endereços de deslocamentos(offsets):

```
~radiff2 bin1 bin2
```

```
root@ubuntu:~# radiff2 ialle ialle2 | head
0x000000d0 a8 => b0 0x000000d0
0x000000d8 a8 => b0 0x000000d8
0x00000198 14 => 18 0x00000198
0x000001a0 14 => 18 0x000001a0
0x000001a8 14 => 18 0x000001a8
0x00000284 e6df5ac59705693e00da6697f5643b062450847 => f53eacce9608685de8357ae19165d63902800b49 0x00000284
0x00000905 48656c6c => 45752073 0x00000905
0x0000090a 2c => 75 0x0000090a
0x0000090c 576f726c6421000011b033b480000008 => 756d206d616c776172650a00011b033b4c 0x0000090c
0x00000920 8cfdffff94000000cfdffffbc00000dcfdffff6400000e6feffffd40000004fffff4000004dfffff140100006cfffff34010000dcfffff7c0100001400
000c8dfffffc000000d8fdfffff8000000e2ferffffd80000000fffff800000049fffff1801000068fffff38010000d8fffff8001 0x00000920
```

Se você não tiver certeza sobre o fato de estar lidando com binários aparentemente semelhantes, verifique algumas funções específicas com a opção -C. As colunas são:

- "Primeiro offset do binário",
- "Porcentagem de igualdade"
- "Segundo offset do binário"

```
root@ubuntu:~# radiff2 ialle ialle2 | head
0x000000d0 a8 => b0 0x000000d0
0x000000d8 a8 => b0 0x000000d8
0x00000198 14 => 18 0x00000198
0x000001a0 14 => 18 0x000001a0
0x000001a8 14 => 18 0x000001a8
0x00000284 e6df5ac59705693e00da6697f5643b062450847 => f53eacce9608685de8357ae19165d63902800b49 0x00000284
0x00000905 48656c6c => 45752073 0x00000905
0x0000090a 2c => 75 0x0000090a
0x0000090c 576f726c6421000011b033b4800000008 => 756d206d616c776172650a00011b033b4c 0x0000090c
0x00000920 8cfdffff94000000cfdffffbc000000dcfdffff64000000e6feffffd40000004ffffff40000004dffffff140100006cffffff34010000dcffffff7c0100001400
000c8fdffffc000000d8fdffff68000000e2feffffd800000000ffffff800000049ffffff1801000068ffffff38010000d8ffffff8001 0x00000920
```

Em um exemplo mais realista, você pode usar o comando `radiff2 -g main /bin/bin1 /bin/bin2 | xdot -`, dessa forma mostrará as diferenças entre a função main dos binários, o que facilitaria muito sua vida num processo de binary exploitation. Observe a ordem dos argumentos abaixo:

`~radiff2 -g main netcat netcat-edited | xdot -`

The screenshot shows a terminal window with the command `radiff2 -g main netcat netcat-edited | xdot -` and a window titled "Dot Viewer" displaying assembly code. The code is organized into blocks connected by arrows, showing the flow of execution and the differences between the two binaries. The assembly code includes instructions like `push r15`, `mov r15d, 0xffffffff`, `mov rbp, rsi`, `mov rbp, 0xd`, `mov esi, 1`, `lea rbx, qword [0x00006b50]`, `sub rsp, 0x4148`, `lea r13, qword [rsp + 0x28]`, `mov rax, qword fs:[0x28]`, `mov qword [rsp + 0x4138], rax`, `xor eax, eax`, `call 0x1e00`, `mov qword [rsp + 8], 0`, `mov dword [rsp + 0x14], 5`, `nop dword [rax]`, `lea rdx, qword str.46bCdDfHl:i:kIM:m:NnO:P:p:qr5s:TtUuV:vWw:Xx:Zz`, `mov rsi, rbp`, `mov edi, r12d`, `call 0x1ec0`, `cmp eax, 0xff`, `je 0x2590`, `sub eax, 0x34`, `cmp eax, 0x46`, `ja 0x2580`, `movsxd rax, dword [rbx + rax*4]`, `add rax, rbx`, and `jmp rax`. Red arrows point from the terminal command to the assembly code blocks in the Dot Viewer window.

O Rafind2 é mais um utilitário de linha de comando da r\_search library, basicamente ele permite que você procure por seqüências de caracteres, seqüências de bytes com binary masks, etc. O que torna ele muito eficiente, primeiro veja uma busca por "lib" dentro de /bin/ls:

```
~rafind2 -s lib /bin/ls
~rafind2 -s lib /bin/ls | wc -l
~export F=/bin/ls
~for a in `rafind2 -s lib $F` ; do \
    > r2 -ns $a -qc'x 32' $F ; done
~rafind2 -i /bin/ls
```

```
ialle@ubuntu:~$ rafind2 -s lib /bin/ls
0x239
0x1181
0x1202
0x167f
0x1a6a2
0x1af00
0x1b16e
ialle@ubuntu:~$ rafind2 -s lib /bin/ls | wc -l
7
ialle@ubuntu:~$ export F=/bin/ls
ialle@ubuntu:~$ for a in `rafind2 -s lib $F` ; do \
> r2 -ns $a -qc'x 32' $F ; done
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00000239 6c69 6236 342f 6c64 2d6c 696e 7578 2d78 lib64/ld-linux-x
0x00000249 3836 2d36 342e 736f 2e32 0004 0000 0010 86-64.so.2.....
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00001181 6c69 6273 656c 696e 7578 2e73 6f2e 3100 libselinux.so.1.
0x00001191 5f49 544d 5f64 6572 6567 6973 7465 7254 _ITM_deregisterT
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00001202 6c69 6263 2e73 6f2e 3600 6666 6c75 7368 libc.so.6.fflush
0x00001212 0073 7472 6370 7900 676d 7469 6d65 5f72 .strcpy.gmtime_r
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x0000167f 6c69 6263 5f73 7461 7274 5f6d 6169 6e00 libc_start_main.
0x0000168f 6469 7266 6400 6673 6565 6b6f 0073 7472 dirfd.fseeko.str
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x0001a6a2 6c69 6273 2f00 6c74 2d00 e280 9800 e280 libs/.lt-.....
0x0001a6b2 9900 a107 6500 a1af 0022 0060 0073 6865 ....e....".`.she
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x0001af00 6c69 622f 7873 7472 746f 6c2e 6300 0000 lib/xstrtoul.c...
0x0001af10 3020 3c3d 2073 7472 746f 6c5f 6261 7365 0 <= strtoul_base
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x0001b16e 6c69 622f 7838 365f 3634 2d6c 696e 7578 lib/x86_64-linux
0x0001b17e 2d67 6e75 0043 4841 5253 4554 414c 4941 -gnu.CHARSETALIA
ialle@ubuntu:~$ rafind2 -i /bin/ls
0x00000000 0x00000000 1 ELF 64-bit LSB shared object, x86-64, version 1
0x00000000 1 ELF 64-bit LSB shared object, x86-64, version 1
```

Com uma simples função, conseguimos identificar previamente algumas strings específicas, os offsets onde estão alocados as strings e usar o database de "magic number" do radare2. Se necessitar identificar strings de conexão de rede, use a opção "-z" semelhante ao que você faz com o rabin2 -z, mas sem se importar em analisar cabeçalhos e obedecer a sections do binário.

E por último, a principal ferramenta do framework; o radare2(r2), podendo ser chamado via command line das duas formas. Vamos fazer uma análise um pouco mais minuciosa, começando com a listagem das sections em módulo interativo:

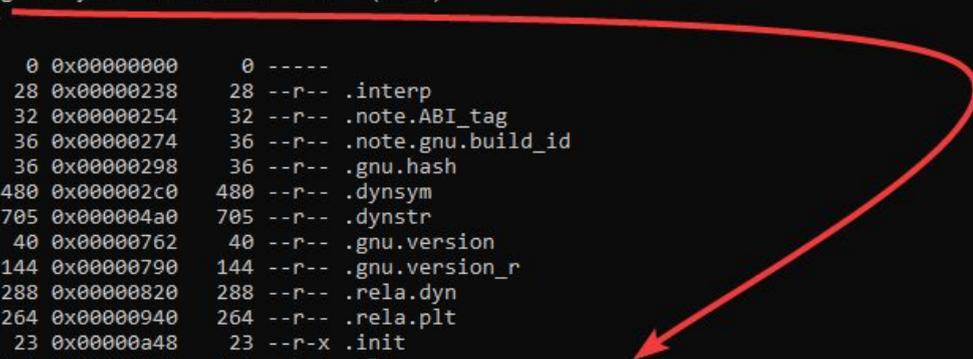
Como você deve saber, os binários têm sections e maps. As seções definem o conteúdo de uma parte do arquivo que pode ser mapeada em memória(ou não). O que é mapeado é definido pelos segmentos.

~r2 malware

[0x00000b30]>aaaa

[0x00000b30]>iS

```
[0x00000b30]> aaaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze len bytes of instructions for references (aar)
[x] Analyze function calls (aac)
[x] Emulate code to find computed references (aae)
[x] Analyze consecutive function (aat)
[x] Constructing a function name for fcn.* and sym.func.* functions (aan)
[x] Type matching analysis for all functions (afta)
[0x00000b30]> iS
[Sections]
00 0x00000000      0 0x00000000      0 -----
01 0x00000238     28 0x00000238     28 --r-- .interp
02 0x00000254     32 0x00000254     32 --r-- .note.ABI_tag
03 0x00000274     36 0x00000274     36 --r-- .note.gnu.build_id
04 0x00000298     36 0x00000298     36 --r-- .gnu.hash
05 0x000002c0    480 0x000002c0    480 --r-- .dynsym
06 0x000004a0    705 0x000004a0    705 --r-- .dynstr
07 0x00000762     40 0x00000762     40 --r-- .gnu.version
08 0x00000790    144 0x00000790    144 --r-- .gnu.version_r
09 0x00000820    288 0x00000820    288 --r-- .rela.dyn
10 0x00000940    264 0x00000940    264 --r-- .rela.plt
11 0x00000a48     23 0x00000a48     23 --r-x .init
12 0x00000a60    192 0x00000a60    192 --r-x .plt
13 0x00000b20      8 0x00000b20      8 --r-x .plt.got
14 0x00000b30    994 0x00000b30    994 --r-x .text
15 0x00000f14      9 0x00000f14      9 --r-x .fini
16 0x00000f20     81 0x00000f20     81 --r-- .rodata
17 0x00000f74     84 0x00000f74     84 --r-- .eh_frame_hdr
18 0x00000fc8    408 0x00000fc8    408 --r-- .eh_frame
19 0x00001160     19 0x00001160     19 --r-- .gcc_except_table
20 0x00001d38     16 0x00201d38     16 --rw- .init_array
21 0x00001d48      8 0x00201d48      8 --rw- .fini_array
22 0x00001d50    528 0x00201d50    528 --rw- .dynamic
23 0x00001f60    160 0x00201f60    160 --rw- .got
24 0x00002000     24 0x00202000     24 --rw- .data
25 0x00002018      0 0x00202020    280 --rw- .bss
26 0x00002018     42 0x00000000     42 ----- .comment
27 0x00002048   1992 0x00000000   1992 ----- .symtab
28 0x00002810   1390 0x00000000   1390 ----- .strtab
29 0x00002d7e    272 0x00000000    272 ----- .shstrtab
30 0x00000040    504 0x00000040    504 m-r-- PHDR
31 0x00000238     28 0x00000238     28 m-r-- INTERP
32 0x00000000   4467 0x00000000   4467 m-r-x LOAD0
33 0x00001d38    736 0x00201d38   1024 m-rw- LOAD1
```



Durante a análise de malware, é importante verificar a entropia das sections do PE. Se houver alta entropia, isso pode indicar a presença de dado criptografado(packer e afins), veja abaixo como usar o r2 para calcular a entropia:

```
~radare2 {malware.bin}
~iS
~iS entropy
~p==e @section..rsrc{nome da section que deseja analisar}
```

```
28 0x00002810 1390 0x00000000 1390 ---- entropy=05000000 .strtab
29 0x00002d7e 272 0x00000000 272 ---- entropy=04000000 .shstrtab
30 0x00000040 504 0x00000040 504 m-r-- entropy=01000000 PHDR
31 0x00000238 28 0x00000238 28 m-r-- entropy=03000000 INTERP
32 0x00000000 4467 0x00000000 4467 m-r-x entropy=04000000 LOAD0
33 0x00001d38 736 0x00201d38 1024 m-rw- entropy=01000000 LOAD1
34 0x00001d50 528 0x00201d50 528 m-rw- entropy=01000000 DYNAMIC
35 0x00000254 68 0x00000254 68 m-r-- entropy=03000000 NOTE
36 0x00000f74 84 0x00000f74 84 m-r-- entropy=03000000 GNU_EH_FRAME
37 0x00000000 0 0x00000000 0 m-rw- entropy=00000000 GNU_STACK
38 0x00001d38 712 0x00201d38 712 m-r-- entropy=01000000 GNU_RELRO
39 0x00000000 64 0x00000000 64 m-rw- entropy=02000000 ehdr

[0x00000b30]> p==e @section..strtab
[0x00000b30]> p==e @section..rodata
[0x00000b30]> _
```

Para listagens de “imports” em modo interativo, use o comando ii:

[0x00000b30]> ii

```
[0x00000b30]> ii
[Imports]
 1 0x00000000 WEAK FUNC __cxa_finalize
 2 0x00000a70 GLOBAL FUNC sym.imp.std::_cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string()
 3 0x00000a80 GLOBAL FUNC __cxa_atexit
 4 0x00000a90 GLOBAL FUNC sym.imp.std::basic_ostream<char,std::char_traits<char>>::operator<<<char,std::char_traits<char>,std::allocator<char>>>(std::basic_ost
ream<char,std::char_traits<char>>&,std::_cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::const&)
 5 0x00000aa0 GLOBAL FUNC sym.imp.std::allocator<char>::~allocator()
 6 0x00000ab0 GLOBAL FUNC __stack_chk_fail
 7 0x00000ac0 GLOBAL FUNC sym.imp.std::_cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(charconst*,std::allocator<char>const&
)
 8 0x00000ad0 GLOBAL FUNC sym.imp.std::_cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length()const
 9 0x00000ae0 GLOBAL FUNC sym.imp.std::ios_base::Init::~Init()
10 0x00000000 GLOBAL FUNC __gxx_personality_v0
11 0x00000000 WEAK NOTYPE ITM_registerTMCloneTable
12 0x00000af0 GLOBAL FUNC __Unwind_Resume
13 0x00000b00 GLOBAL FUNC sym.imp.std::allocator<char>::~allocator()
14 0x00000000 GLOBAL FUNC __libc_start_main
15 0x00000000 WEAK NOTYPE __gmon_start__
16 0x00000000 WEAK NOTYPE ITM_registerTMCloneTable
17 0x00000b10 GLOBAL FUNC sym.imp.std::_cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](unsignedlong)
18 0x00000000 GLOBAL FUNC sym.imp.std::ios_base::Init::~Init()
 1 0x00000000 WEAK FUNC __cxa_finalize
 2 0x00000a70 GLOBAL FUNC sym.imp.std::_cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string()
 3 0x00000a80 GLOBAL FUNC __cxa_atexit
 4 0x00000a90 GLOBAL FUNC sym.imp.std::basic_ostream<char,std::char_traits<char>>::operator<<<char,std::char_traits<char>,std::allocator<char>>>(std::basic_ost
ream<char,std::char_traits<char>>&,std::_cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::const&)
 5 0x00000aa0 GLOBAL FUNC sym.imp.std::allocator<char>::~allocator()
 6 0x00000ab0 GLOBAL FUNC __stack_chk_fail
 7 0x00000ac0 GLOBAL FUNC sym.imp.std::_cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(charconst*,std::allocator<char>const&
)
 8 0x00000ad0 GLOBAL FUNC sym.imp.std::_cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length()const
 9 0x00000ae0 GLOBAL FUNC sym.imp.std::ios_base::Init::~Init()
10 0x00000000 GLOBAL FUNC __gxx_personality_v0
11 0x00000000 WEAK NOTYPE ITM_registerTMCloneTable
12 0x00000af0 GLOBAL FUNC __Unwind_Resume
13 0x00000b00 GLOBAL FUNC sym.imp.std::allocator<char>::~allocator()
14 0x00000000 GLOBAL FUNC __libc_start_main
15 0x00000000 WEAK NOTYPE __gmon_start__
16 0x00000000 WEAK NOTYPE ITM_registerTMCloneTable
17 0x00000b10 GLOBAL FUNC sym.imp.std::_cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](unsignedlong)
18 0x00000000 GLOBAL FUNC sym.imp.std::ios_base::Init::~Init()
```

Para listar as funções que serão executadas, use o comando iz:

[0x00000b30]> iz

```
[0x00000b30]> iz
000 0x00000f30 0x00000f30 64 65 (.rodata) ascii Hey, eu sou um malware e vou droppar http://malware/malware.exe!
```

Observe que, ao contrário de muitas ferramentas, o radare2 não depende da API do Windows para analisar arquivos PDB, portanto eles podem ser carregados em qualquer outra plataforma suportada - por exemplo, Linux ou OS X. Você pode carregar os símbolos usando o comando is:

[0x00000b30]> is

```
[0x00000b30]> is
[Symbols]
028 0x00000b60 0x00000b60 LOCAL FUNC 0 deregister_tm_clones
029 0x00000ba0 0x00000ba0 LOCAL FUNC 0 register_tm_clones
030 0x00000bf0 0x00000bf0 LOCAL FUNC 0 __do_global_dtors_aux
031 0x00002130 0x00202130 LOCAL OBJECT 1 completed.7696
032 0x00001d48 0x00201d48 LOCAL OBJECT 0 __do_global_dtors_aux_fini_array_entry
033 0x00000c30 0x00000c30 LOCAL FUNC 0 frame_dummy
034 0x00001d38 0x00201d38 LOCAL OBJECT 0 __frame_dummy_init_array_entry
036 0x00000f28 0x00000f28 LOCAL OBJECT 1 std::piecewise_construct
037 0x00002131 0x00202131 LOCAL OBJECT 1 std::_ioinit
038 0x00000e3e 0x00000e3e LOCAL FUNC 73 __static_initialization_and_destruction_0(int,int)
039 0x00000e87 0x00000e87 LOCAL FUNC 21 _GLOBAL__sub_I_Z12inversaoCaseRNS7_cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
041 0x0000115c 0x0000115c LOCAL OBJECT 0 _FRAME_END_
043 0x00000f74 0x00000f74 LOCAL NOTYPE 0 __GNU_EH_FRAME_HDR
044 0x00001d50 0x00201d50 LOCAL OBJECT 0 _DYNAMIC

0x0000074 55 push rbp
0x0000075 489e5 mov rbp, rsp
0x0000078 53 push rbx
0x0000079 483ec48 sub rsp, 0x48 ; 'H'
0x000007d 6448b042528. mov rax, qword fs:[0x28] ; [0x28:8]=0x2e0 ; '('
0x0000086 48945e8 mov qword [local_18h], rax
0x000008a 31c0 xor eax, eax
0x000008c 48045bf lea rax, qword [local_41h]
0x0000098 489c7 mov rdi, rax
0x0000093 e86fdffff call sym.std::allocator_char::_allocator
0x0000098 48d55bf lea rdx, qword [local_41h]
0x000009c 48d45c0 lea rax, qword [local_40h]
0x000009a 48d3589100. lea rsi, qword str.Hey_eu_sou_um_malware_e_vou_droppar_http://malware/malware.exe ; 0xf30 ; "Hey, eu sou um malware e vou dr
/malware/malware.exe!"
```

Para finalizar, uma das formas mais inteligentes de ter um overview mais rápido do binário que está sendo analisado, é simplesmente usar o comando “V!”, e se ainda tiver dúvida ou estiver com dificuldade de usar algum comando, use o argumento “?” para listar informações e opções mais detalhadas:

The screenshot shows the Radare2 interface with the disassembly pane on the left and the symbols/stack/registers pane on the right. The disassembly pane shows instructions from address 0x1400a9380 to 0x1400a93fb. The symbols pane lists various Windows API functions like GDI32.dll\_CreateBitmap. The stack pane shows memory addresses from 0x00000000 to 0x0000000f. The registers pane shows the current state of registers like rax, rdx, r8, r11, r14, and rbp.

```
[0x1400a9384] > ?
Usage: [.] [times] [cmd] [~grep] [@[iter] addr|size] [ |>pipe ] ; ...
Append '?' to any char command to get detailed help
Prefix with number to repeat command N times (f.ex: 3x)
%var =valueAlias for 'env' command
* [?] off=[0x]value      Pointer read/write data/values (see ?v, wx, wv)
(macro arg0 arg1)       Manage scripting macros
.[?] [-](m)|f|sh|cmd]   Define macro or load r2, cparse or rlang file
=[?] [cmd]              Send/Listen for Remote Commands (rap://, http://, <fd>)
/[?]                   Search for bytes, regexps, patterns, ..
![?] [cmd]              Run given command as in system(3)
#[?] !lang [...]       Hashbang to run an rlang script
a [?]                  Analysis commands
b [?]                  Display or change the block size
c [?] [arg]            Compare block with given data
C [?]                  Code metadata (comments, format, hints, ..)
d [?]                  Debugger commands
e [?] [a=[b]]          List/get/set config evaluable vars
f [?] [name] [fmt] [ft]
```

Para finalizar a primeira parte do “Reverse Engineering 101 using Radare2” vou deixar abaixo alguns endereços importantes para usar como referência de estudos e desenvolvimento de suas habilidades em engenharia reversa, usando essa ferramenta que é altamente didática, até a próxima!

Book: <https://legacy.gitbook.com/book/radare/radare2book>

Github: <https://github.com/radare/radare2>

Site: <https://www.radare.org/r/>

Extra plugins: <https://github.com/radare/radare2-extras>

Tips: <http://radare.today/posts/>