Reverse Engineering 101 using Radare2

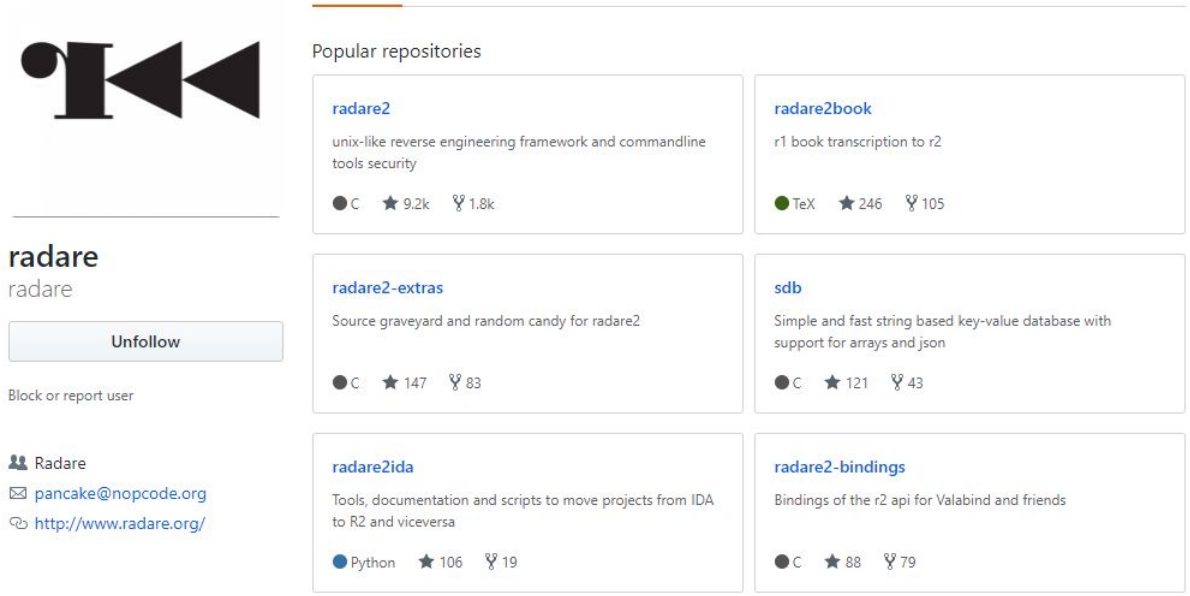
Autor: Ialle Teixeira
Blog: https://www.linkedin.com/in/isDebuggerPresent/

Um pouco da história sobre o Radare2

Por volta de 2006, Sergi Àlvarez (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 fonto 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.



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 multimplataforma, 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.

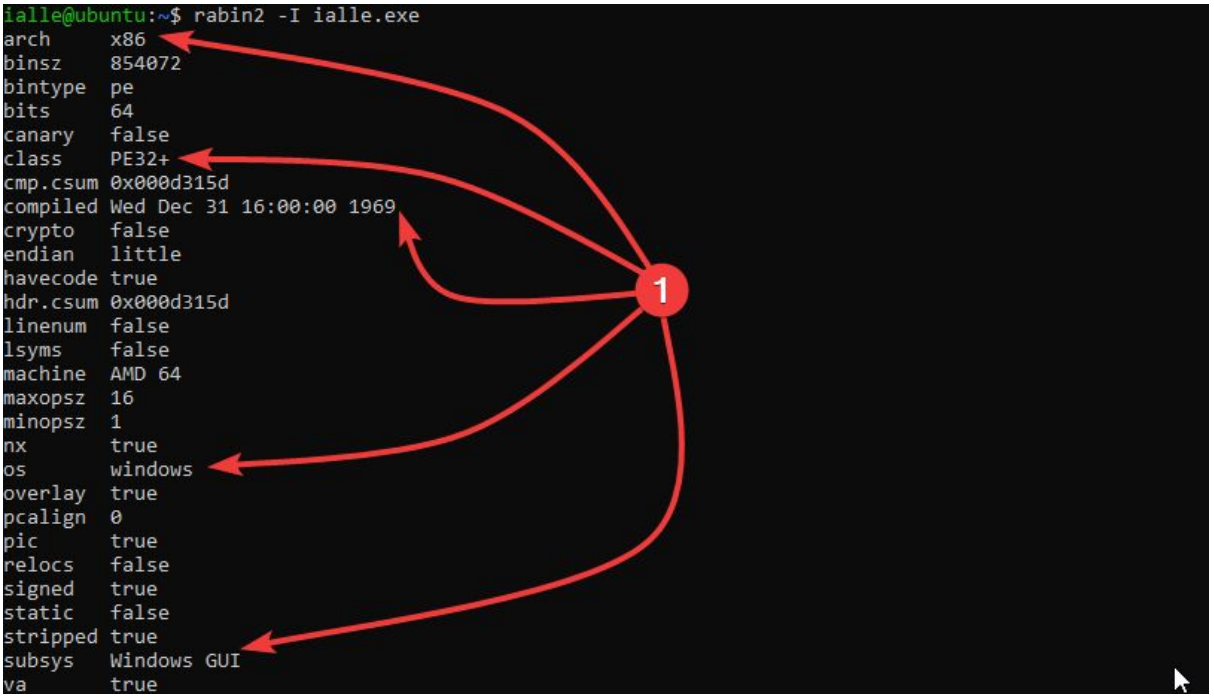
ragg2: Simplesmente um frontend para r_egg(muito usada para software exploitation). Suporte para x86, x86-64, e ARM.

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.

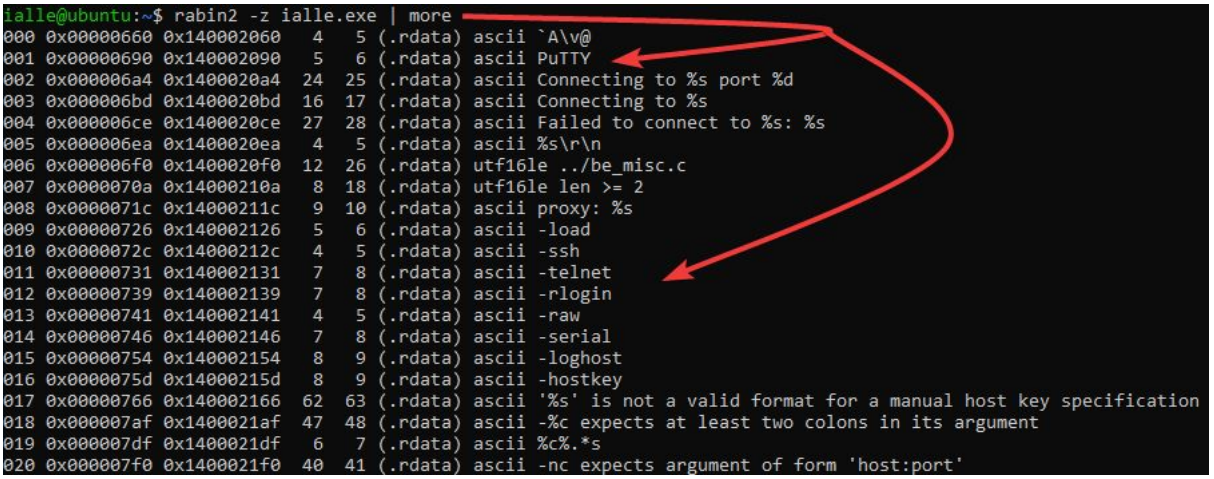
rax2: Uma ferramenta para executar expressões matemáticas bem minimalista em command line, que é útil para fazer conversões de base entre valores de ponto flutuante, representação hexadecimal, conversão para ASCII, e muito mais.

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



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:



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 e6df5ac59705693e00da6697f5643b6062450847 => f53eecce9608685de8357ae19165d63902800b49 0x00000284
0x00000905 48656c6c => 45752073 0x00000905
0x0000090a 2c => 75 0x0000090a
0x0000090c 576f726c64210000011b033b4800000008 => 756d206d616c776172650a00011b033b4c 0x0000090c
0x0000092d 8cfdffff94000000ccfdffffbc000000dcfdffff64000000e6feffffd40000000400000004ffffffff40000004dffffff140100006cffffff34010000dcffffff7c0100001400
000c8fdffffc0000000d8fdffff68000000e2feffffd80000000ffffffff800000049ffffff18010000068ffffff38010000d8ffffff8001 0x00000920
```

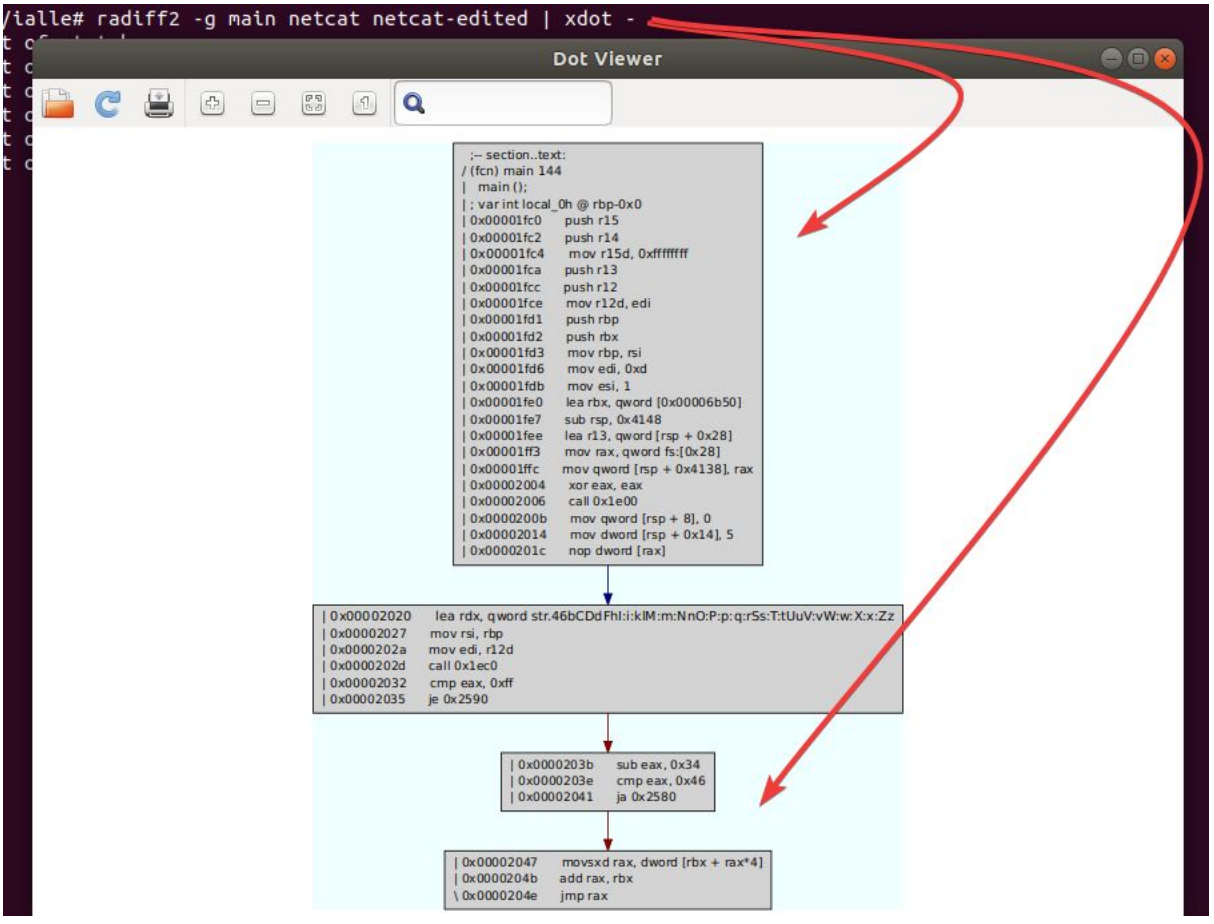
Se você não tiver certeza sobre o fato de estar lidando com binários aparentemente semelhantes, verifique algumas funções específicas com a opção -C. As colunas são:

- "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 e6df5ac59705693e00da6697f5643b6062450847 => f53eecce9608685de8357ae19165d63902800b49 0x00000284
0x00000905 48656c6c => 45752073 0x00000905
0x0000090a 2c => 75 0x0000090a
0x0000090c 576f726c64210000011b033b4800000008 => 756d206d616c776172650a00011b033b4c 0x0000090c
0x00000920 8cfdffff94000000ccfdffffbc000000dcfdffff64000000e6feffffd40000004ffffff40000004dffffff140100006cffffff34010000dcffffff7c0100001400
000c8fdffffc0000000d8fdffff68000000e2feffffd80000000ffffffff800000049ffffff18010000068ffffff38010000d8ffffff8001 0x00000920
```

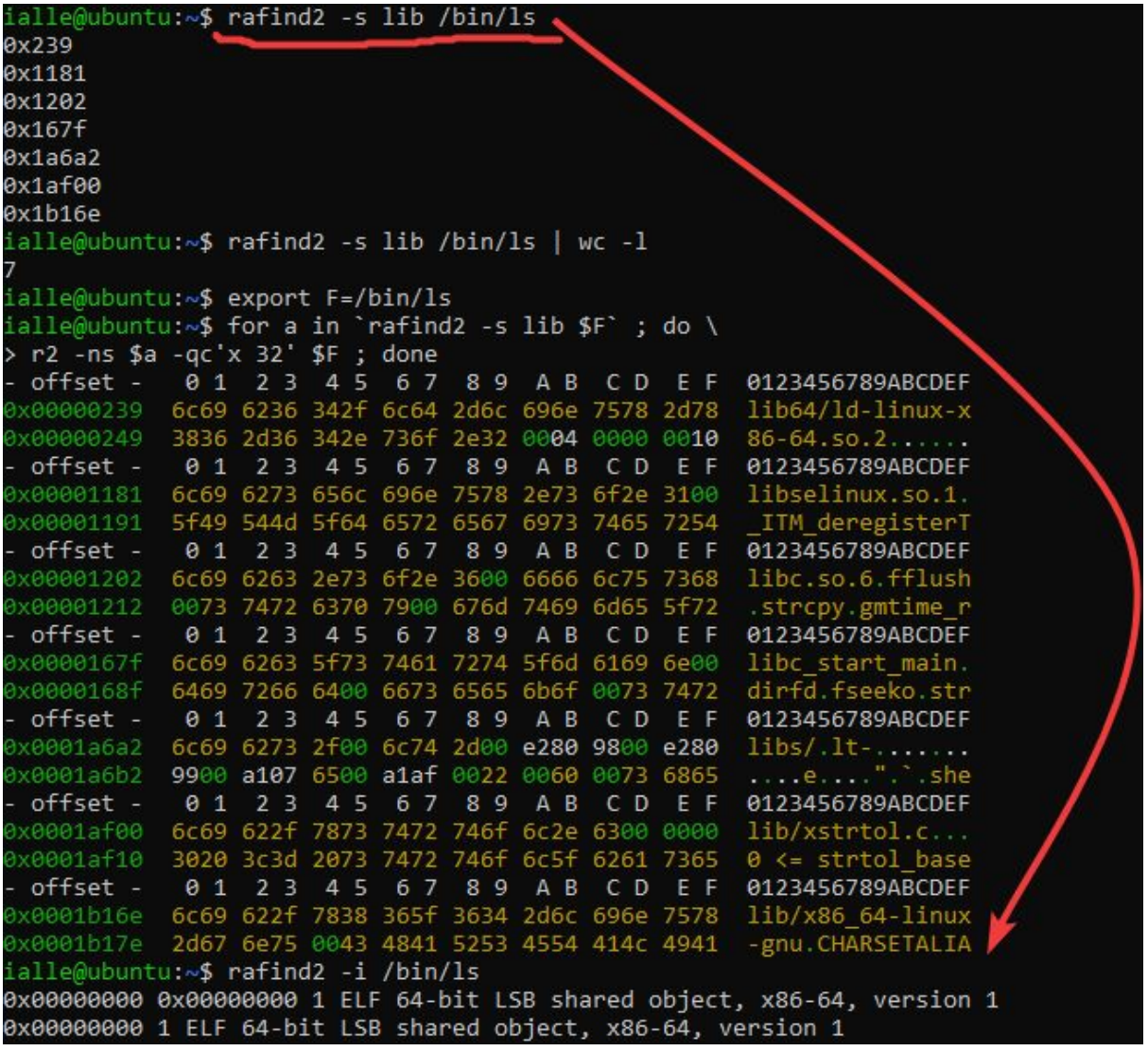
Em um exemplo mais realista, você pode usar o comando *radiff2 -g main /bin/bin1 /bin/bin2 | xdot -*, dessa forma mostrará as diferenças entre a função main dos binários, o que facilitaria muito sua vida num processo de binary exploitation. Observe a ordem dos argumentos abaixo:

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



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

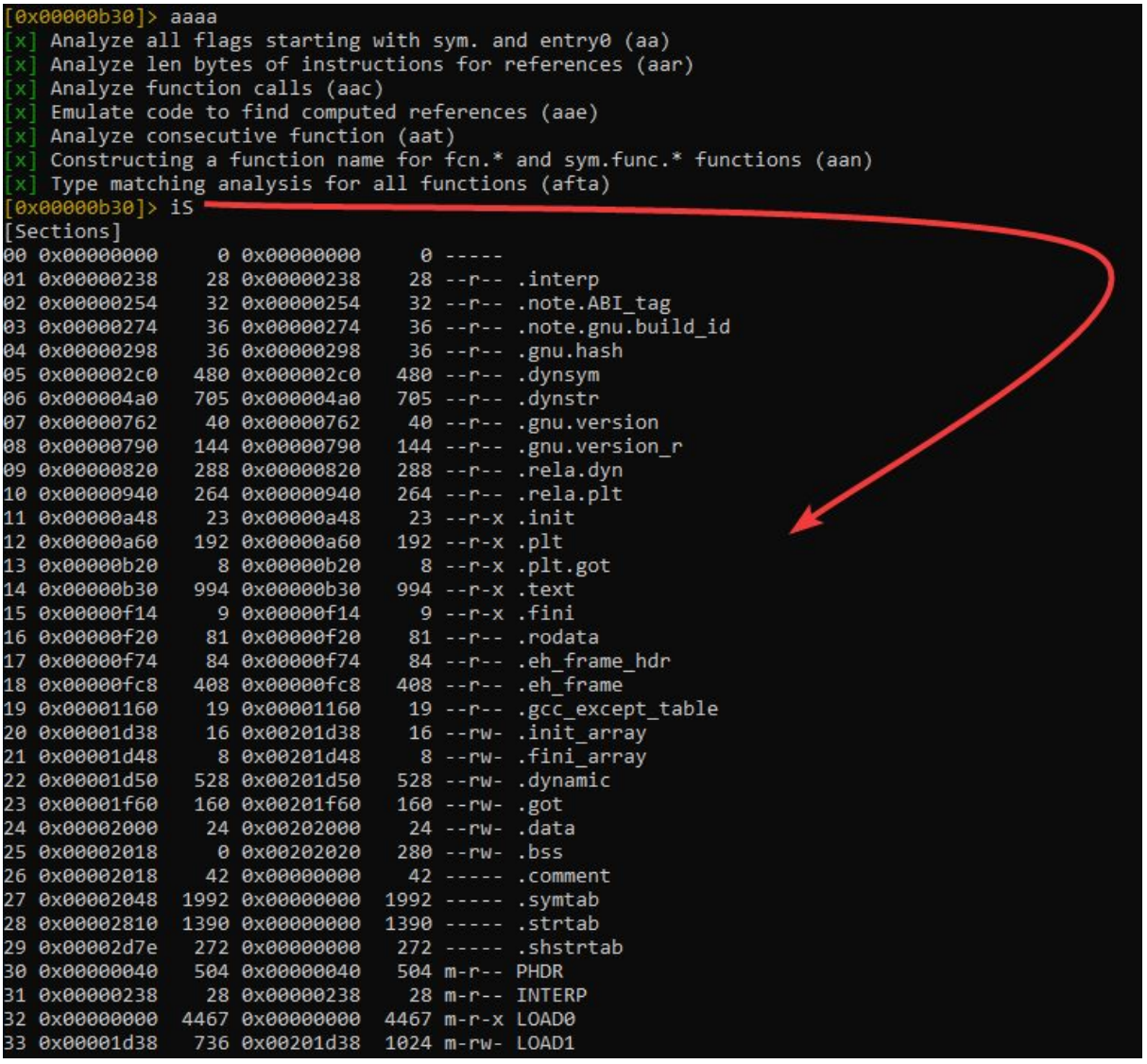


Com uma simples função, conseguimos identificar previamente algumas strings específicas, os offsets onde estão alocados as strings e usar o database de “magic number” do radare2. Se necessitar identificar strings de conexão de rede, use a opção “-z” semelhante ao que você faz com o rabin2 -z, mas sem 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:

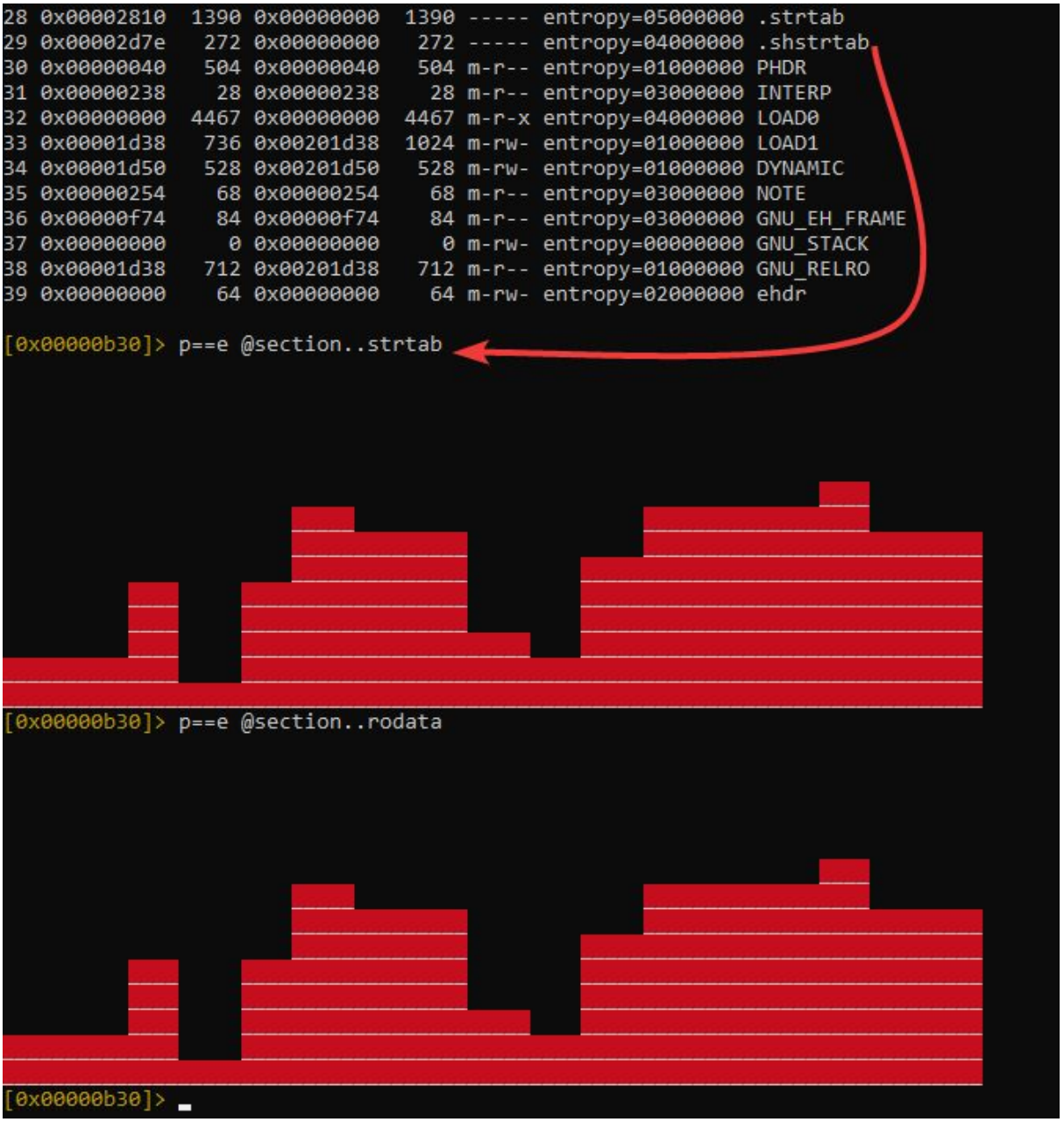
Como você deve saber, os binários têm sections e maps. As seções definem o conteúdo de uma parte do arquivo que pode ser mapeada em memória(ou não). O que é mapeado é definido pelos segmentos.

~r2 malware
[0x00000b30]>aaaa
[0x00000b30]>iS



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}



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>>&std::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_deregisterTMCloneTable
  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
  10 0x00000000  GLOBAL    FUNC __gxx_personality_v0
  11 0x00000000    WEAK  NOTYPE _ITM_deregisterTMCloneTable
  14 0x00000000  GLOBAL    FUNC __libc_start_main
  15 0x00000000    WEAK  NOTYPE __gmon_start__
  16 0x00000000    WEAK  NOTYPE _ITM_registerTMCloneTable
  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!
```

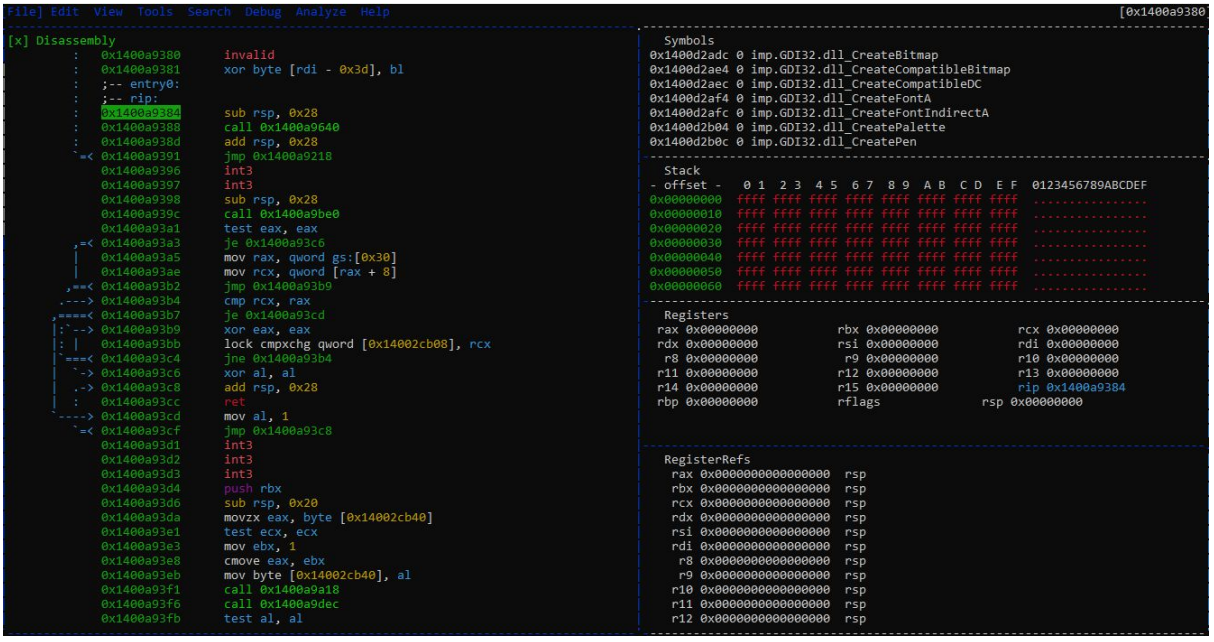
Observe que, ao contrário de muitas ferramentas, o radare2 não depende da API do Windows para analisar arquivos PDB, portanto eles podem ser carregados em qualquer outra plataforma suportada - por exemplo, Linux ou OS X. Você pode carregar os símbolos usando o comando is:

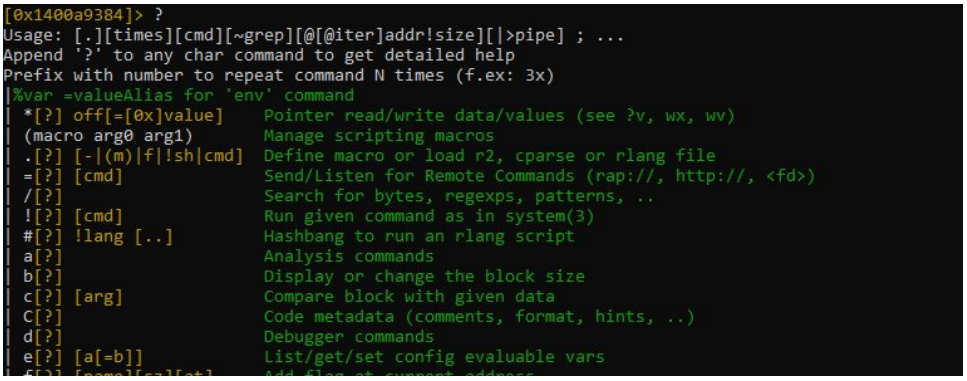
[0x00000b30]> is

```
[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__Z12inversaoCaseRNSt7__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
```

```
0x00000d74      55              push rbp
0x00000d75      4889e5          mov rbp, rsp
0x00000d78      53              push rbx
0x00000d79      4883ec48        sub rsp, 0x48               ; 'H'
0x00000d7d      64488b042528.   mov rax, qword fs:[0x28]    ; [0x28:8]=0x2e90 ; '('
0x00000d86      488945e8        mov qword [local_18h], rax
0x00000d8a      31c0            xor eax, eax
0x00000d8c      488d45bf        lea rax, qword [local_41h]
0x00000d90      4889c7          mov rdi, rax
0x00000d93      e868fdffff      call sym.std::allocator_char_::allocator
0x00000d98      488d55bf        lea rdx, qword [local_41h]
0x00000d9c      488d45c0        lea rax, qword [local_40h]
0x00000da0      488d3589010.    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:





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/