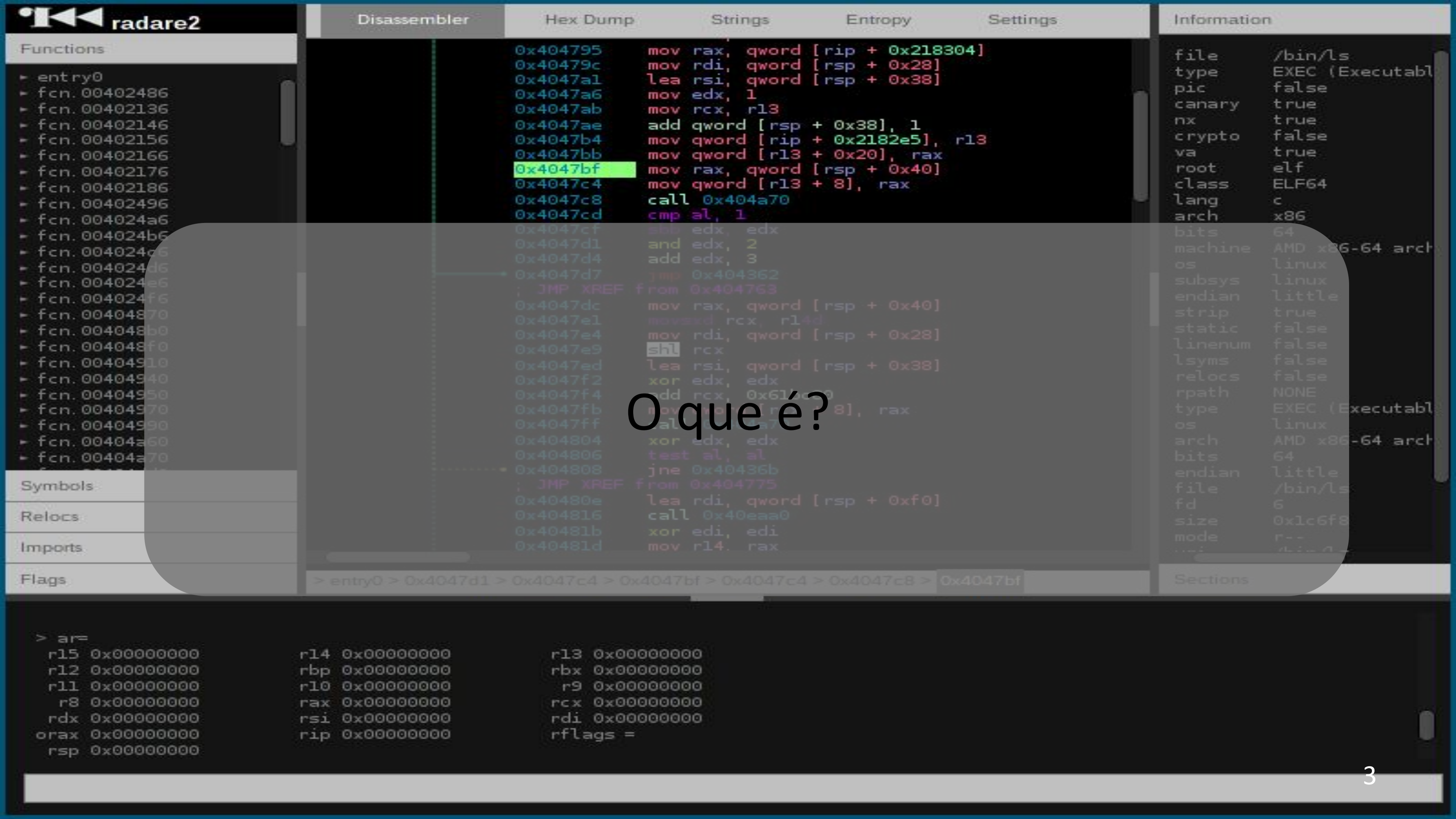
Pwning

python -c ‘print “A”*100’ | ./ganesh

Pwning

[1] 1782 segmentation fault (core dumped)



```
entry0
fcn.00402486
fcn.00402136
fcn.00402146
fcn.00402156
fcn.00402166
fcn.00402176
fcn.00402186
fcn.00402496
fcn.004024a6
fcn.004024b6
fcn.004024c6
fcn.004024d6
fcn.004024e6
fcn.004024f6
fcn.00404870
fcn.004048b0
fcn.004048f0
fcn.00404910
fcn.00404940
fcn.00404950
fcn.00404970
fcn.00404990
fcn.00404a60
fcn.00404a70
```

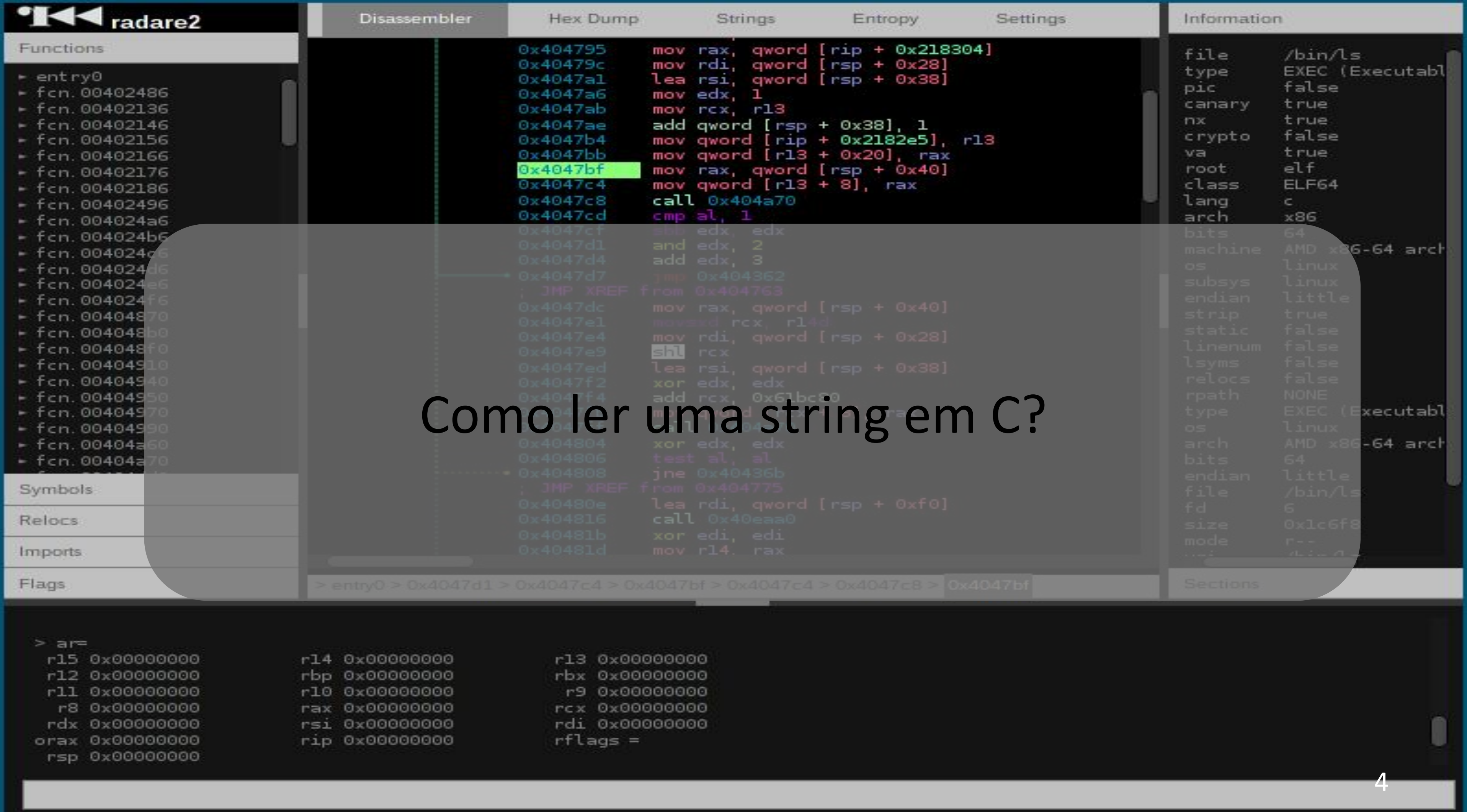
```
0x404795   mov rax, qword [rip + 0x218304]
0x40479c   mov rdi, qword [rsp + 0x28]
0x4047a1   lea rsi, qword [rsp + 0x38]
0x4047a6   mov edx, 1
0x4047ab   mov rcx, r13
0x4047ae   add qword [rsp + 0x38], 1
0x4047b4   mov qword [rip + 0x2182e5], r13
0x4047bb   mov qword [r13 + 0x20], rax
0x4047bf   mov rax, qword [rsp + 0x40]
0x4047c4   mov qword [r13 + 8], rax
0x4047c8   call 0x404a70
0x4047cd   cmp al, 1
0x4047cf   sbb edx, edx
0x4047d1   and edx, 2
0x4047d4   add edx, 3
0x4047d7   jmp 0x404362
; JMP XREF from 0x404763
0x4047dc   mov rax, qword [rsp + 0x40]
0x4047e1   movsxd rcx, r14d
0x4047e4   mov rdi, qword [rsp + 0x28]
0x4047e9   shl rcx
0x4047ed   lea rsi, qword [rsp + 0x38]
0x4047f2   xor edx, edx
0x4047f4   add rcx, 0x61[illegible]
0x4047fb   mov [illegible] 8], rax
0x4047ff   call [illegible]
0x404804   xor edx, edx
0x404806   test al, al
0x404808   jne 0x40436b
; JMP XREF from 0x404775
0x40480e   lea rdi, qword [rsp + 0xf0]
0x404816   call 0x40eaa0
0x40481b   xor edi, edi
0x40481d   mov r14, rax
```

```
> entry0 > 0x4047d1 > 0x4047c4 > 0x4047bf > 0x4047c4 > 0x4047c8 > 0x4047bf
```

```
file      /bin/ls
type      EXEC (Executabl
pic       false
canary    true
nx        true
crypto    false
va        true
root      elf
class     ELF64
lang      c
arch      x86
bits      64
machine   AMD x86-64 arch
os        linux
subsys    linux
endian    little
strip     true
static    false
linenum   false
lsyms     false
relocs    false
rpath     NONE
type      EXEC (Executabl
os        linux
arch      AMD x86-64 arch
bits      64
endian    little
file      /bin/ls
fd        6
size      0x1c6f8
mode      r--
```

```
> ar=
 r15 0x00000000        r14 0x00000000        r13 0x00000000
 r12 0x00000000        rbp 0x00000000        rbx 0x00000000
 r11 0x00000000        r10 0x00000000         r9 0x00000000
  r8 0x00000000        rax 0x00000000        rcx 0x00000000
 rdx 0x00000000        rsi 0x00000000        rdi 0x00000000
orax 0x00000000        rip 0x00000000        rflags =
 rsp 0x00000000
```

O que é?



```
entry0
fcn.00402486
fcn.00402136
fcn.00402146
fcn.00402156
fcn.00402166
fcn.00402176
fcn.00402186
fcn.00402496
fcn.004024a6
fcn.004024b6
fcn.004024c6
fcn.004024d6
fcn.004024e6
fcn.004024f6
fcn.00404870
fcn.004048b0
fcn.004048f0
fcn.00404910
fcn.00404940
fcn.00404950
fcn.00404970
fcn.00404990
fcn.00404a60
fcn.00404a70
```

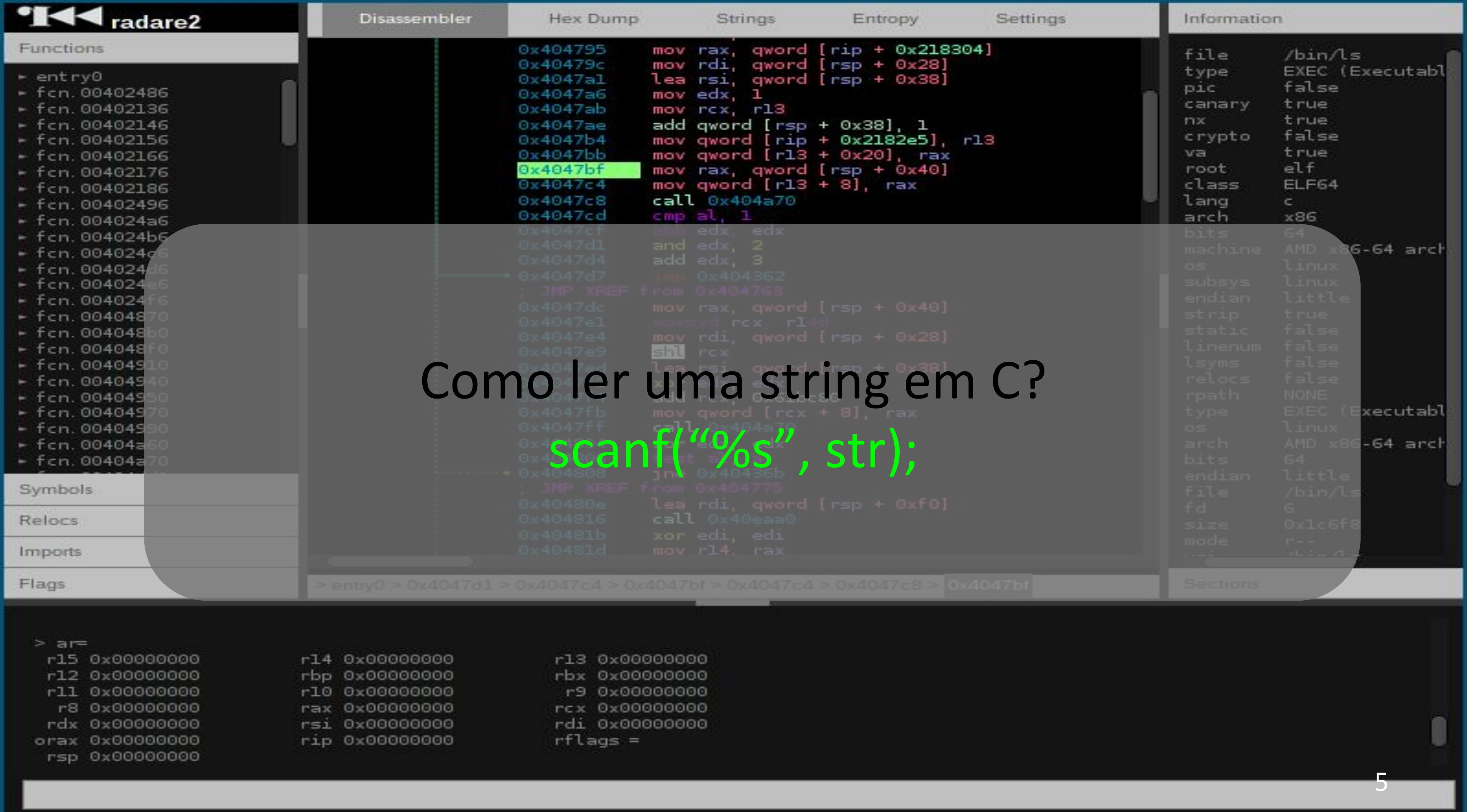
```
0x404795    mov rax, qword [rip + 0x218304]
0x40479c    mov rdi, qword [rsp + 0x28]
0x4047a1    lea rsi, qword [rsp + 0x38]
0x4047a6    mov edx, 1
0x4047ab    mov rcx, r13
0x4047ae    add qword [rsp + 0x38], 1
0x4047b4    mov qword [rip + 0x2182e5], r13
0x4047bb    mov qword [r13 + 0x20], rax
0x4047bf    mov rax, qword [rsp + 0x40]
0x4047c4    mov qword [r13 + 8], rax
0x4047c8    call 0x404a70
0x4047cd    cmp al, 1
0x4047cf    sbb edx, edx
0x4047d1    and edx, 2
0x4047d4    add edx, 3
0x4047d7    jmp 0x404362
; JMP XREF from 0x404763
0x4047dc    mov rax, qword [rsp + 0x40]
0x4047e1    movsxd rcx, r14d
0x4047e4    mov rdi, qword [rsp + 0x28]
0x4047e9    shl rcx
0x4047ed    lea rsi, qword [rsp + 0x38]
0x4047f2    xor edx, edx
0x4047f4    add rcx, 0x61bc80
[illegible]
[illegible]
0x404804    xor edx, edx
0x404806    test al, al
0x404808    jne 0x40436b
; JMP XREF from 0x404775
0x40480e    lea rdi, qword [rsp + 0xf0]
0x404816    call 0x40eaa0
0x40481b    xor edi, edi
0x40481d    mov r14, rax
```

```
> entry0 > 0x4047d1 > 0x4047c4 > 0x4047bf > 0x4047c4 > 0x4047c8 > 0x4047bf
```

```
file      /bin/ls
type      EXEC (Executabl
pic       false
canary    true
nx        true
crypto    false
va        true
root      elf
class     ELF64
lang      c
arch      x86
bits      64
machine   AMD x86-64 arch
os        linux
subsys    linux
endian    little
strip     true
static    false
linenum   false
lsyms     false
relocs    false
rpath     NONE
type      EXEC (Executabl
os        linux
arch      AMD x86-64 arch
bits      64
endian    little
file      /bin/ls
fd        6
size      0x1c6f8
mode      r--
```

```
> ar=
 r15 0x00000000      r14 0x00000000      r13 0x00000000
 r12 0x00000000      rbp 0x00000000      rbx 0x00000000
 r11 0x00000000      r10 0x00000000       r9 0x00000000
  r8 0x00000000      rax 0x00000000      rcx 0x00000000
 rdx 0x00000000      rsi 0x00000000      rdi 0x00000000
orax 0x00000000      rip 0x00000000      rflags =
 rsp 0x00000000
```

Como ler uma string em C?



```
entry0
fcn.00402486
fcn.00402136
fcn.00402146
fcn.00402156
fcn.00402166
fcn.00402176
fcn.00402186
fcn.00402496
fcn.004024a6
```

```
0x404795    mov rax, qword [rip + 0x218304]
0x40479c    mov rdi, qword [rsp + 0x28]
0x4047a1    lea rsi, qword [rsp + 0x38]
0x4047a6    mov edx, 1
0x4047ab    mov rcx, r13
0x4047ae    add qword [rsp + 0x38], 1
0x4047b4    mov qword [rip + 0x2182e5], r13
0x4047bb    mov qword [r13 + 0x20], rax
0x4047bf    mov rax, qword [rsp + 0x40]
0x4047c4    mov qword [r13 + 8], rax
0x4047c8    call 0x404a70
0x4047cd    cmp al, 1
```

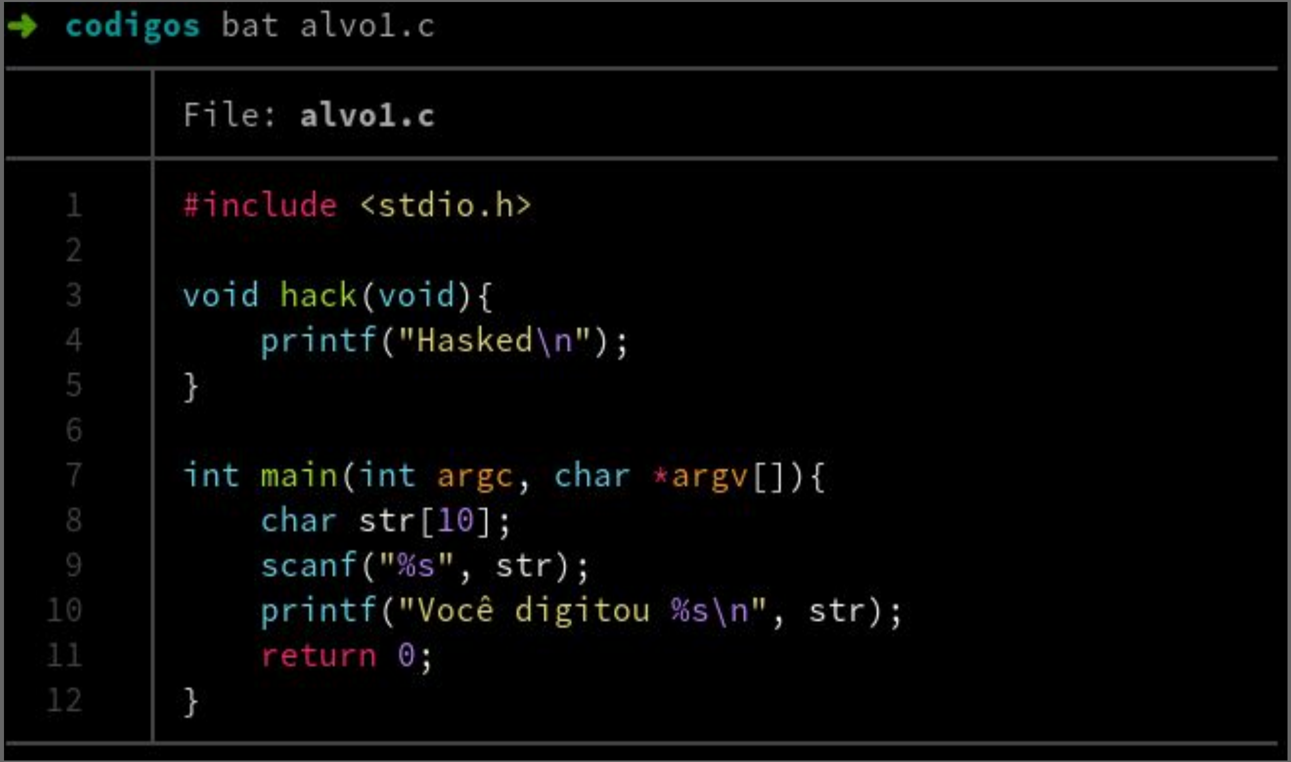
```
file      /bin/ls
type      EXEC (Executabl
pic       false
canary    true
nx        true
crypto    false
va        true
root      elf
class     ELF64
lang      c
arch      x86
```

```
> ar=
 r15 0x00000000      r14 0x00000000      r13 0x00000000
 r12 0x00000000      rbp 0x00000000      rbx 0x00000000
 r11 0x00000000      r10 0x00000000       r9 0x00000000
  r8 0x00000000      rax 0x00000000      rcx 0x00000000
 rdx 0x00000000      rsi 0x00000000      rdi 0x00000000
orax 0x00000000      rip 0x00000000      rflags =
 rsp 0x00000000
```

Como ler uma string em C?

`scanf("%s", str);`

Qual o problema?

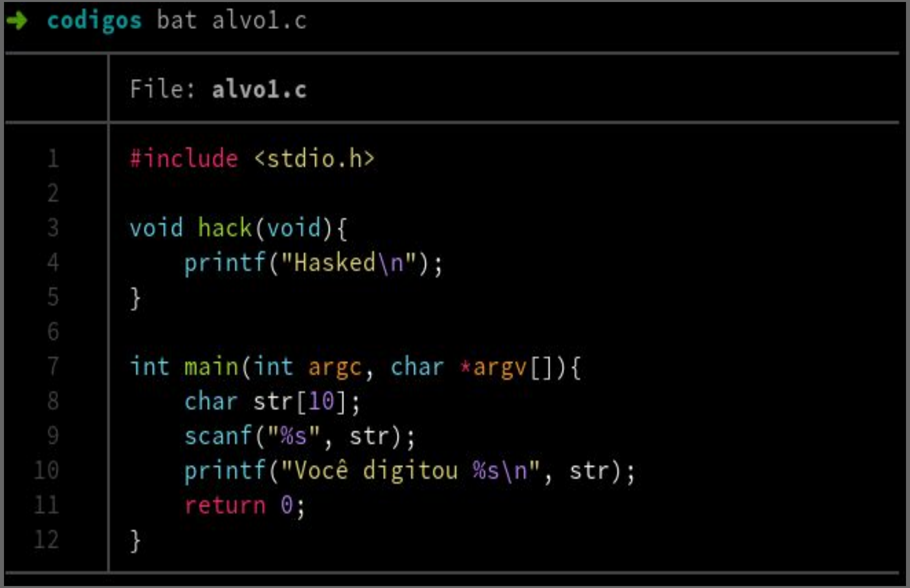


```
➜ codigos bat alvo1.c
       File: alvo1.c

#include <stdio.h>

void hack(void){
    printf("Hasked\n");
}

int main(int argc, char *argv[]){
    char str[10];
    scanf("%s", str);
    printf("Você digitou %s\n", str);
    return 0;
}
```

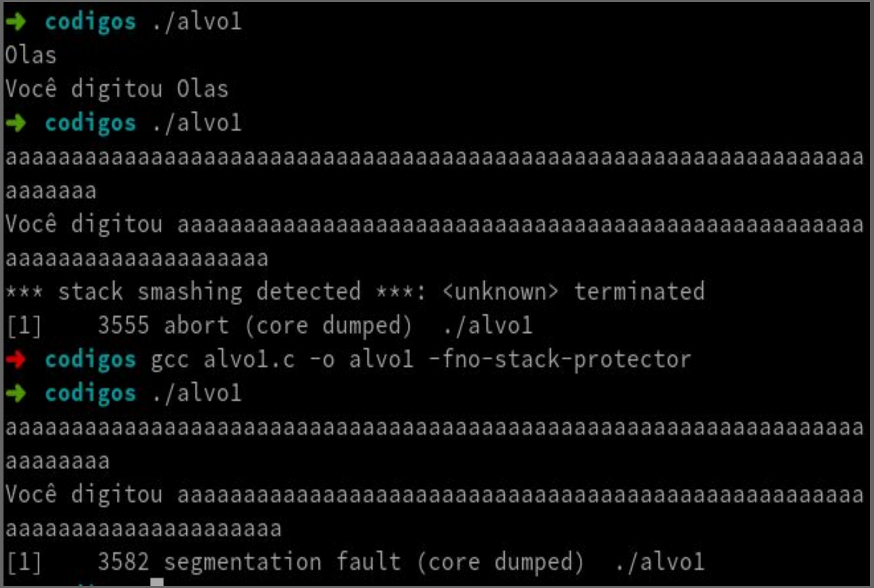


```
➜ codigos bat alvo1.c
       File: alvo1.c

#include <stdio.h>

void hack(void){
    printf("Hasked\n");
}

int main(int argc, char *argv[]){
    char str[10];
    scanf("%s", str);
    printf("Você digitou %s\n", str);
    return 0;
}
```

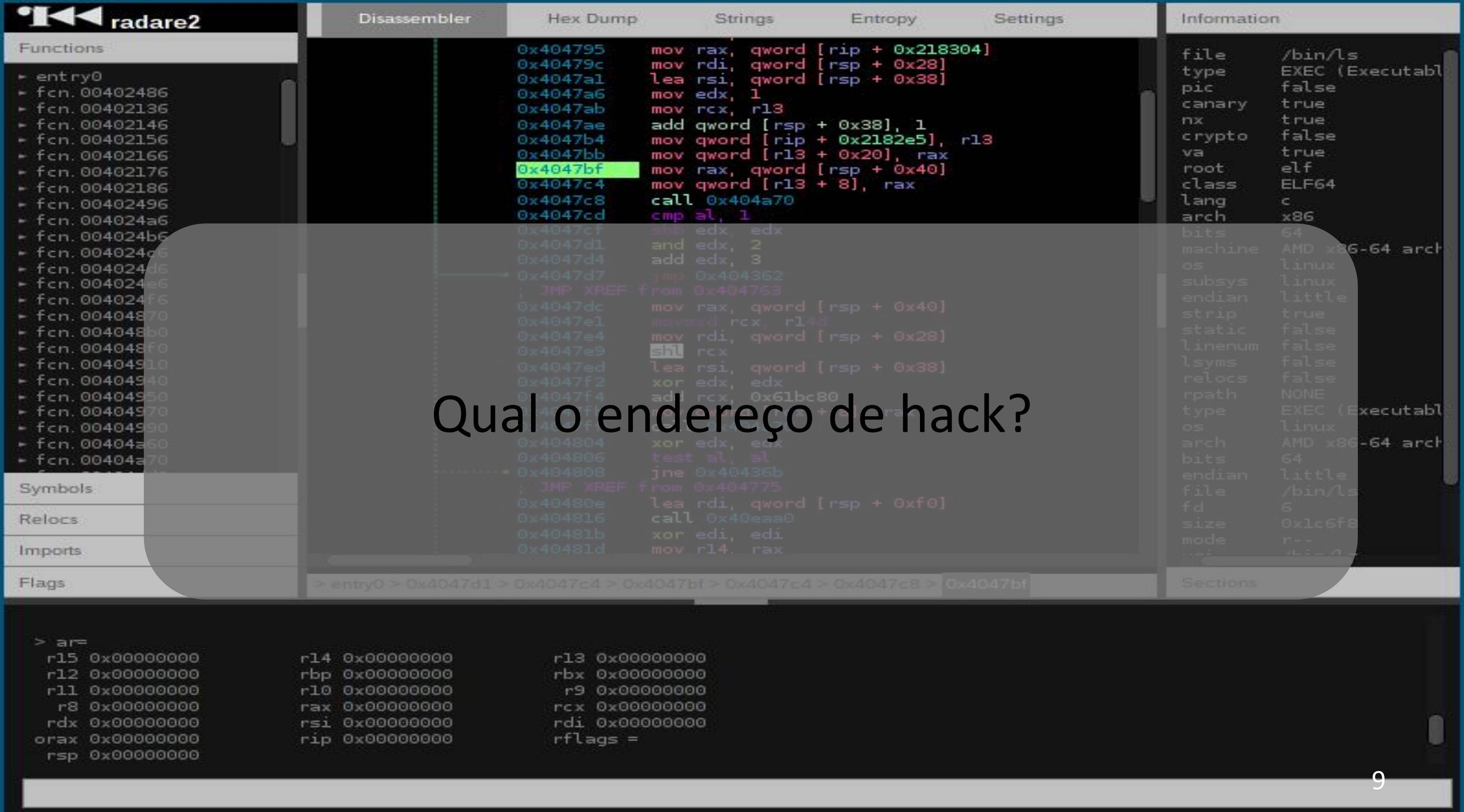


```
➜ codigos ./alvo1
Olas
Você digitou Olas
➜ codigos ./alvo1
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaa
Você digitou aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaa
*** stack smashing detected ***: <unknown> terminated
[1]    3555 abort (core dumped)  ./alvo1
➜ codigos gcc alvo1.c -o alvo1 -fno-stack-protector
➜ codigos ./alvo1
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaa
Você digitou aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaa
[1]    3582 segmentation fault (core dumped)  ./alvo1
```

Funcionamento da pilha

xxxxxA8	_libc_start_main	end. de retorno 8 bytes
xxxxxA0	?	rbp anterior 8 bytes
xxxxx00	<lixo>	str 10 bytes

- Queremos sobrescrever o endereço de retorno e colocar o endereço da função hack



```
► entry0
► fcn.00402486
► fcn.00402136
► fcn.00402146
► fcn.00402156
► fcn.00402166
► fcn.00402176
► fcn.00402186
► fcn.00402496
► fcn.004024a6
► fcn.004024b6
► fcn.004024c6
► fcn.004024d6
► fcn.004024e6
► fcn.004024f6
► fcn.00404870
► fcn.004048b0
► fcn.004048f0
► fcn.00404910
► fcn.00404940
► fcn.00404950
► fcn.00404970
► fcn.00404990
► fcn.00404a60
► fcn.00404a70
```

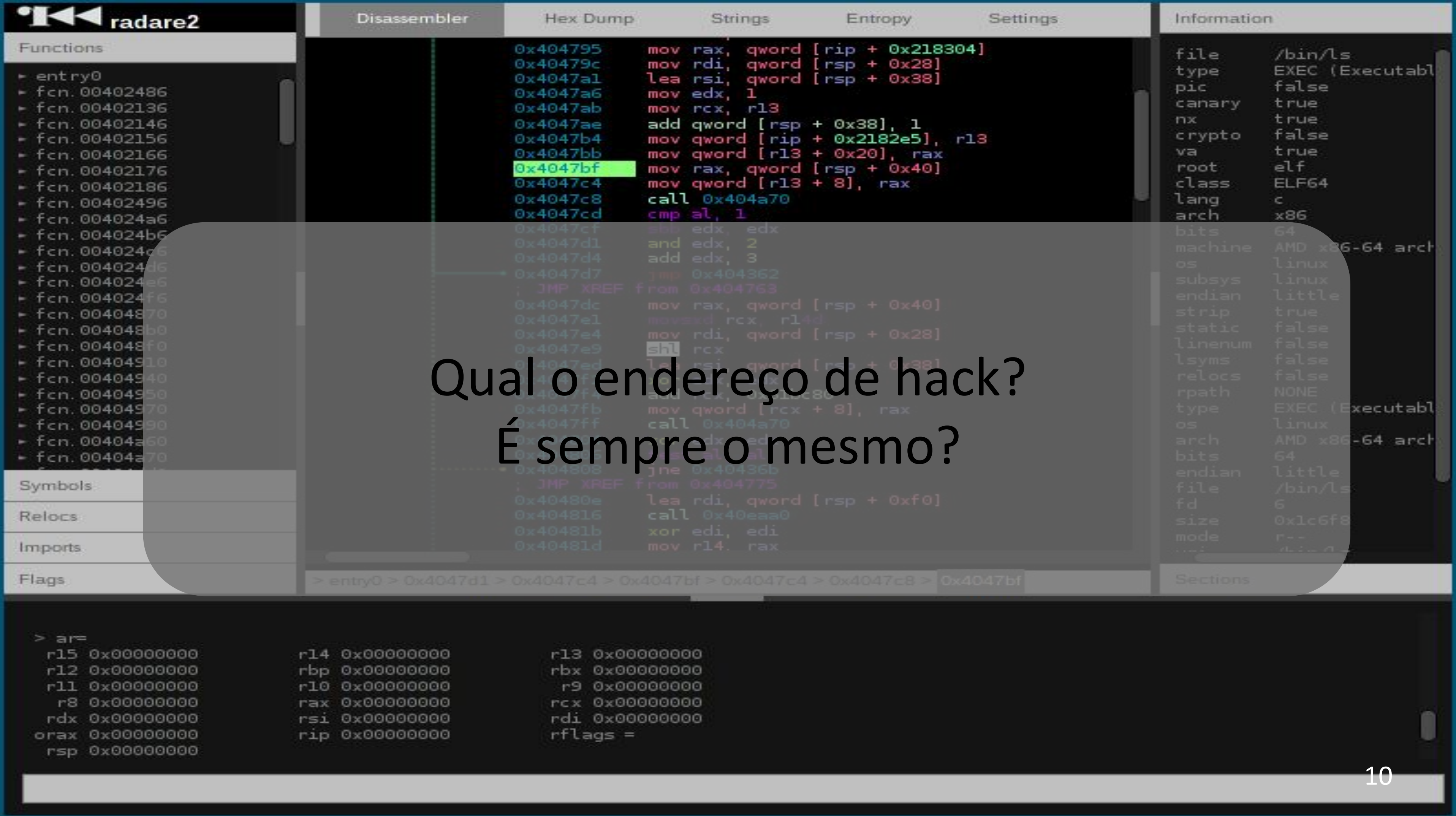
```
0x404795   mov rax, qword [rip + 0x218304]
0x40479c   mov rdi, qword [rsp + 0x28]
0x4047a1   lea rsi, qword [rsp + 0x38]
0x4047a6   mov edx, 1
0x4047ab   mov rcx, r13
0x4047ae   add qword [rsp + 0x38], 1
0x4047b4   mov qword [rip + 0x2182e5], r13
0x4047bb   mov qword [r13 + 0x20], rax
0x4047bf   mov rax, qword [rsp + 0x40]
0x4047c4   mov qword [r13 + 8], rax
0x4047c8   call 0x404a70
0x4047cd   cmp al, 1
0x4047cf   sbb edx, edx
0x4047d1   and edx, 2
0x4047d4   add edx, 3
0x4047d7   jmp 0x404362
; JMP XREF from 0x404763
0x4047dc   mov rax, qword [rsp + 0x40]
0x4047e1   movsxd rcx, r14d
0x4047e4   mov rdi, qword [rsp + 0x28]
0x4047e9   shl rcx
0x4047ed   lea rsi, qword [rsp + 0x38]
0x4047f2   xor edx, edx
0x4047f4   add rcx, 0x61bc80
[illegible]
[illegible]
0x404804   xor edx, edx
0x404806   test al, al
0x404808   jne 0x40436b
; JMP XREF from 0x404775
0x40480e   lea rdi, qword [rsp + 0xf0]
0x404816   call 0x40eaa0
0x40481b   xor edi, edi
0x40481d   mov r14, rax
```

```
> entry0 > 0x4047d1 > 0x4047c4 > 0x4047bf > 0x4047c4 > 0x4047c8 > 0x4047bf
```

```
file      /bin/ls
type      EXEC (Executabl
pic       false
canary    true
nx        true
crypto    false
va        true
root      elf
class     ELF64
lang      c
arch      x86
bits      64
machine   AMD x86-64 arch
os        linux
subsys    linux
endian    little
strip     true
static    false
linenum   false
lsyms     false
relocs    false
rpath     NONE
type      EXEC (Executabl
os        linux
arch      AMD x86-64 arch
bits      64
endian    little
file      /bin/ls
fd        6
size      0x1c6f8
mode      r--
```

```
> ar=
 r15 0x00000000      r14 0x00000000      r13 0x00000000
 r12 0x00000000      rbp 0x00000000      rbx 0x00000000
 r11 0x00000000      r10 0x00000000       r9 0x00000000
  r8 0x00000000      rax 0x00000000      rcx 0x00000000
 rdx 0x00000000      rsi 0x00000000      rdi 0x00000000
orax 0x00000000      rip 0x00000000      rflags =
 rsp 0x00000000
```

Qual o endereço de hack?



```
▸ entry0
▸ fcn.00402486
▸ fcn.00402136
▸ fcn.00402146
▸ fcn.00402156
▸ fcn.00402166
▸ fcn.00402176
▸ fcn.00402186
▸ fcn.00402496
▸ fcn.004024a6
▸ fcn.004024b6
▸ fcn.004024c6
▸ fcn.004024d6
▸ fcn.004024e6
▸ fcn.004024f6
▸ fcn.00404870
▸ fcn.004048b0
▸ fcn.004048f0
▸ fcn.00404910
▸ fcn.00404940
▸ fcn.00404950
▸ fcn.00404970
▸ fcn.00404990
▸ fcn.00404a60
▸ fcn.00404a70
```

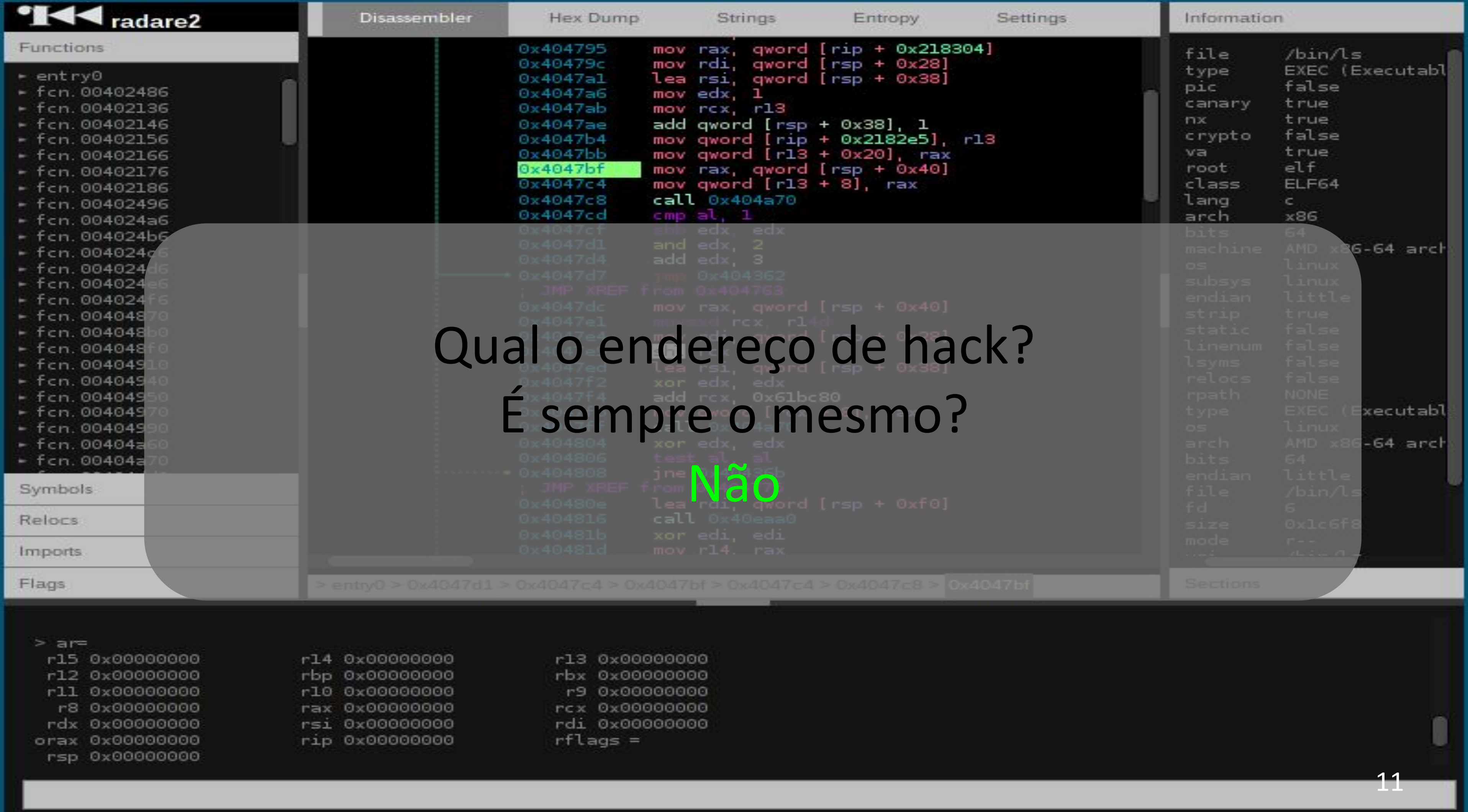
```
0x404795     mov rax, qword [rip + 0x218304]
0x40479c     mov rdi, qword [rsp + 0x28]
0x4047a1     lea rsi, qword [rsp + 0x38]
0x4047a6     mov edx, 1
0x4047ab     mov rcx, r13
0x4047ae     add qword [rsp + 0x38], 1
0x4047b4     mov qword [rip + 0x2182e5], r13
0x4047bb     mov qword [r13 + 0x20], rax
0x4047bf     mov rax, qword [rsp + 0x40]
0x4047c4     mov qword [r13 + 8], rax
0x4047c8     call 0x404a70
0x4047cd     cmp al, 1
0x4047cf     sbb edx, edx
0x4047d1     and edx, 2
0x4047d4     add edx, 3
0x4047d7     jmp 0x404362
; JMP XREF from 0x404763
0x4047dc     mov rax, qword [rsp + 0x40]
0x4047e1     movsxd rcx, r14d
0x4047e4     mov rdi, qword [rsp + 0x28]
0x4047e9     shl rcx
0x4047ed     lea rsi, qword [rsp + 0x38]
[illegible]
0x4047f4     add rcx, 0x61bc80
0x4047fb     mov qword [rcx + 8], rax
0x4047ff     call 0x404a70
[illegible]
0x404808     jne 0x40436b
; JMP XREF from 0x404775
0x40480e     lea rdi, qword [rsp + 0xf0]
0x404816     call 0x40eaa0
0x40481b     xor edi, edi
0x40481d     mov r14, rax
```

```
> entry0 > 0x4047d1 > 0x4047c4 > 0x4047bf > 0x4047c4 > 0x4047c8 > 0x4047bf
```

```
file      /bin/ls
type      EXEC (Executabl
pic       false
canary    true
nx        true
crypto    false
va        true
root      elf
class     ELF64
lang      c
arch      x86
bits      64
machine   AMD x86-64 arch
os        linux
subsys    linux
endian    little
strip     true
static    false
linenum   false
lsyms     false
relocs    false
rpath     NONE
type      EXEC (Executabl
os        linux
arch      AMD x86-64 arch
bits      64
endian    little
file      /bin/ls
fd        6
size      0x1c6f8
mode      r--
```

```
> ar=
 r15 0x00000000      r14 0x00000000      r13 0x00000000
 r12 0x00000000      rbp 0x00000000      rbx 0x00000000
 r11 0x00000000      r10 0x00000000       r9 0x00000000
  r8 0x00000000      rax 0x00000000      rcx 0x00000000
 rdx 0x00000000      rsi 0x00000000      rdi 0x00000000
orax 0x00000000      rip 0x00000000      rflags =
 rsp 0x00000000
```

Qual o endereço de hack?
É sempre o mesmo?



```
► entry0
► fcn.00402486
► fcn.00402136
► fcn.00402146
► fcn.00402156
► fcn.00402166
► fcn.00402176
► fcn.00402186
► fcn.00402496
► fcn.004024a6
► fcn.004024b6
► fcn.004024c6
► fcn.004024d6
► fcn.004024e6
► fcn.004024f6
► fcn.00404870
► fcn.004048b0
► fcn.004048f0
► fcn.00404910
► fcn.00404940
► fcn.00404950
► fcn.00404970
► fcn.00404990
► fcn.00404a60
► fcn.00404a70
```

```
0x404795    mov rax, qword [rip + 0x218304]
0x40479c    mov rdi, qword [rsp + 0x28]
0x4047a1    lea rsi, qword [rsp + 0x38]
0x4047a6    mov edx, 1
0x4047ab    mov rcx, r13
0x4047ae    add qword [rsp + 0x38], 1
0x4047b4    mov qword [rip + 0x2182e5], r13
0x4047bb    mov qword [r13 + 0x20], rax
0x4047bf    mov rax, qword [rsp + 0x40]
0x4047c4    mov qword [r13 + 8], rax
0x4047c8    call 0x404a70
0x4047cd    cmp al, 1
0x4047cf    sbb edx, edx
0x4047d1    and edx, 2
0x4047d4    add edx, 3
0x4047d7    jmp 0x404362
; JMP XREF from 0x404763
0x4047dc    mov rax, qword [rsp + 0x40]
0x4047e1    movsxd rcx, r14d
0x4047e4    [illegible]
0x4047e[illegible] [illegible]
0x4047ed    lea rsi, qword [rsp + 0x38]
0x4047f2    xor edx, edx
0x4047f4    add rcx, 0x61bc80
[illegible]
[illegible]
0x404804    xor edx, edx
0x404806    test al, al
0x404808    jne [illegible]
; JMP XREF from [illegible]
0x40480e    lea rdi, qword [rsp + 0xf0]
0x404816    call 0x40eaa0
0x40481b    xor edi, edi
0x40481d    mov r14, rax
```

```
> entry0 > 0x4047d1 > 0x4047c4 > 0x4047bf > 0x4047c4 > 0x4047c8 > 0x4047bf
```

```
file      /bin/ls
type      EXEC (Executabl
pic       false
canary    true
nx        true
crypto    false
va        true
root      elf
class     ELF64
lang      c
arch      x86
bits      64
machine   AMD x86-64 arch
os        linux
subsys    linux
endian    little
strip     true
static    false
linenum   false
lsyms     false
relocs    false
rpath     NONE
type      EXEC (Executabl
os        linux
arch      AMD x86-64 arch
bits      64
endian    little
file      /bin/ls
fd        6
size      0x1c6f8
mode      r--
```

```
> ar=
 r15 0x00000000        r14 0x00000000        r13 0x00000000
 r12 0x00000000        rbp 0x00000000        rbx 0x00000000
 r11 0x00000000        r10 0x00000000         r9 0x00000000
  r8 0x00000000        rax 0x00000000        rcx 0x00000000
 rdx 0x00000000        rsi 0x00000000        rdi 0x00000000
orax 0x00000000        rip 0x00000000        rflags =
 rsp 0x00000000
```

Qual o endereço de hack?

É sempre o mesmo?

Não

PIE

PIE - Position Independent Code

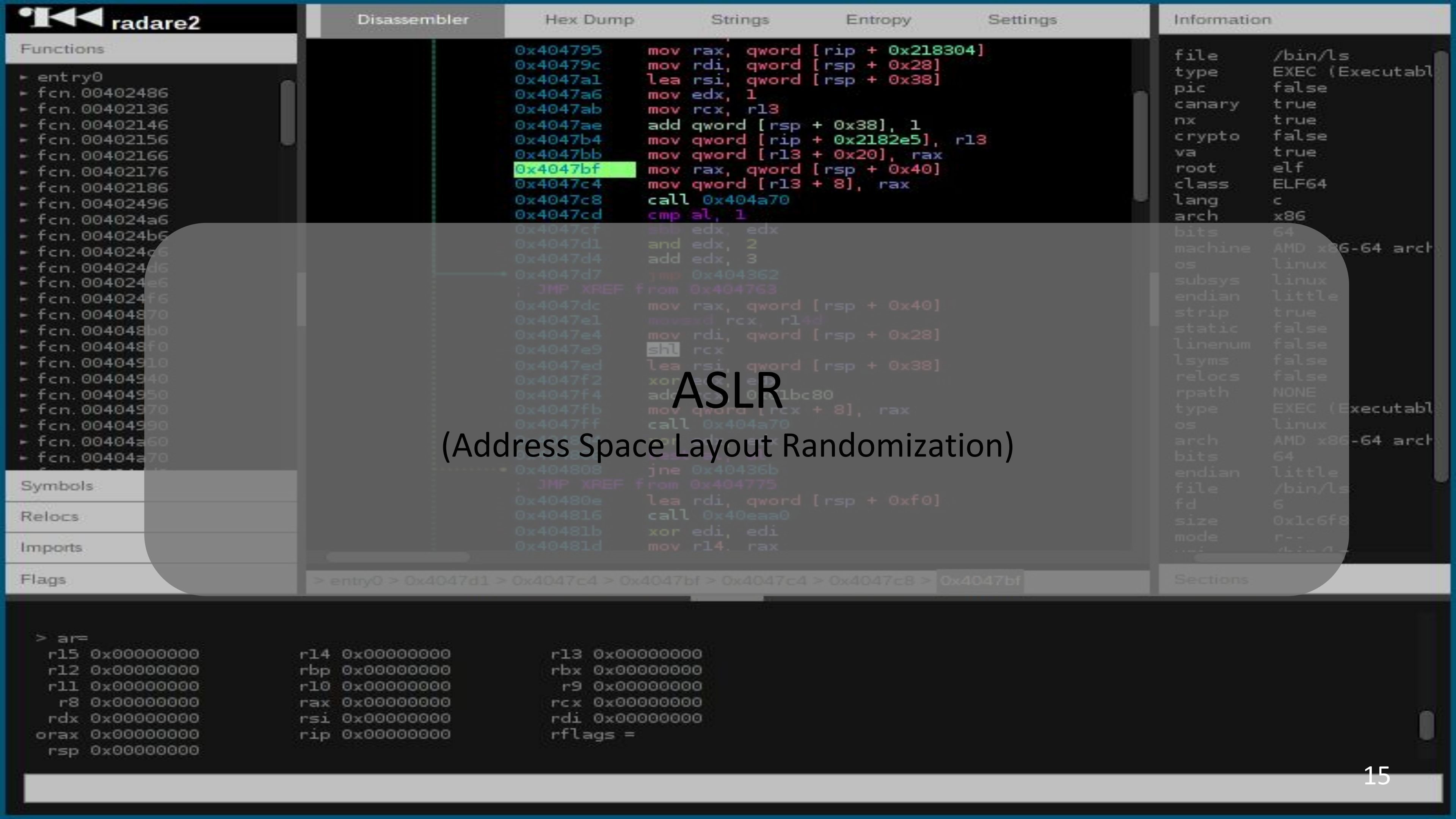
E se a função hack não existisse?

- Shellcode
  - Queremos aproveitar a variável str e injetar código nela
  - Podemos obter o endereço de str?
    - Se fosse global...

```
bits 64
global _start
section .text

_start:
     push 0x646e7770; ‘dnwp’
     mov rax, 1 ; write
     mov rdi, 1
     mov rsi, rsp
     mov rdx, 4
     syscall

     mov rax, 60 ; exit
     mov rdi, 0
     syscall
```



```
► entry0
► fcn.00402486
► fcn.00402136
► fcn.00402146
► fcn.00402156
► fcn.00402166
► fcn.00402176
► fcn.00402186
► fcn.00402496
► fcn.004024a6
```

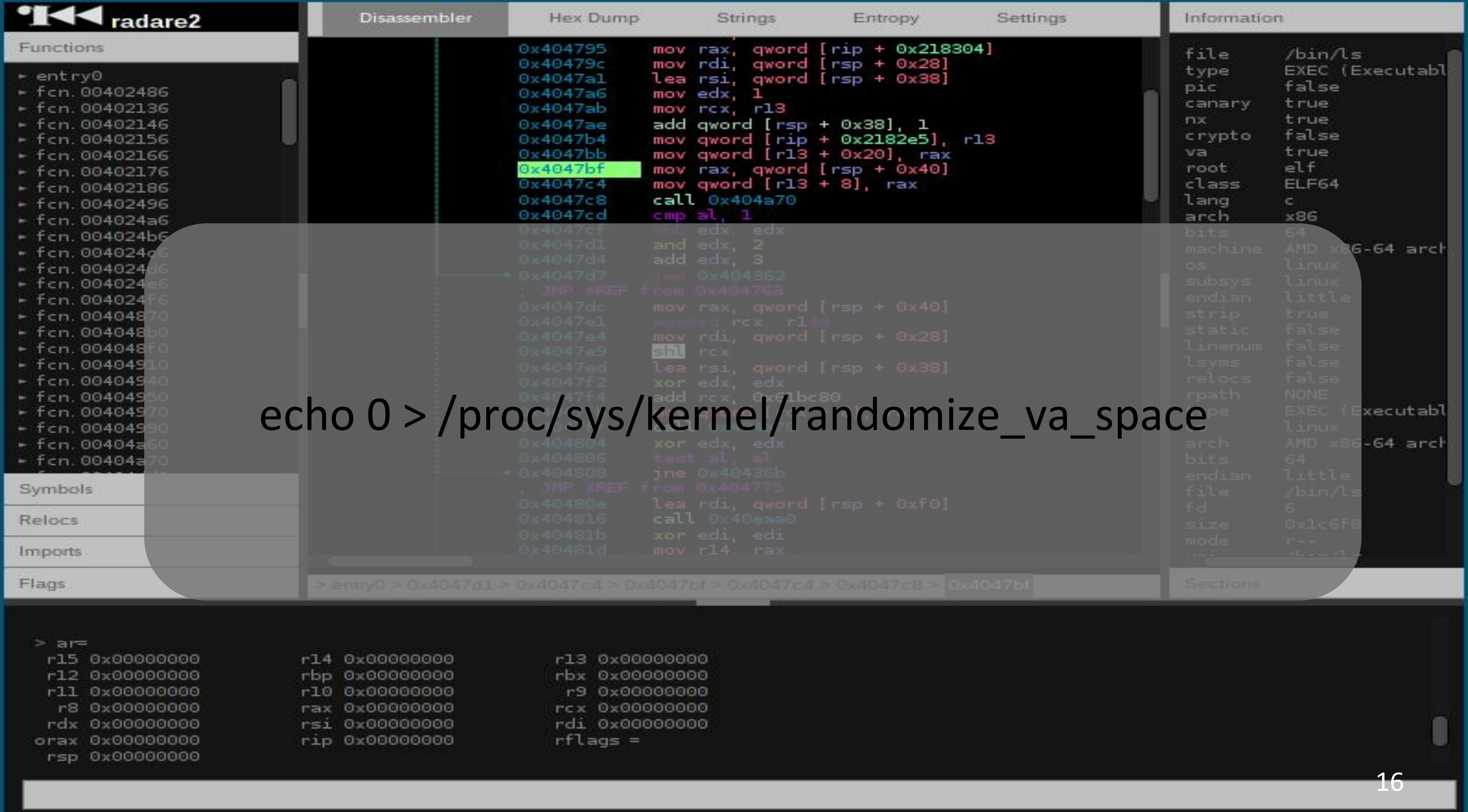
```
0x404795      mov rax, qword [rip + 0x218304]
0x40479c      mov rdi, qword [rsp + 0x28]
0x4047a1      lea rsi, qword [rsp + 0x38]
0x4047a6      mov edx, 1
0x4047ab      mov rcx, r13
0x4047ae      add qword [rsp + 0x38], 1
0x4047b4      mov qword [rip + 0x2182e5], r13
0x4047bb      mov qword [r13 + 0x20], rax
0x4047bf      mov rax, qword [rsp + 0x40]
0x4047c4      mov qword [r13 + 8], rax
0x4047c8      call 0x404a70
0x4047cd      cmp al, 1
```

```
file      /bin/ls
type      EXEC (Executabl
pic       false
canary    true
nx        true
crypto    false
va        true
root      elf
class     ELF64
lang      c
arch      x86
```

```
> ar=
 r15 0x00000000      r14 0x00000000      r13 0x00000000
 r12 0x00000000      rbp 0x00000000      rbx 0x00000000
 r11 0x00000000      r10 0x00000000       r9 0x00000000
  r8 0x00000000      rax 0x00000000      rcx 0x00000000
 rdx 0x00000000      rsi 0x00000000      rdi 0x00000000
orax 0x00000000      rip 0x00000000      rflags =
 rsp 0x00000000
```

ASLR

(Address Space Layout Randomization)



```
entry0
fcn.00402486
fcn.00402136
fcn.00402146
fcn.00402156
fcn.00402166
fcn.00402176
fcn.00402186
fcn.00402496
fcn.004024a6
fcn.004024b6
fcn.004024c6
fcn.004024d6
fcn.004024e6
fcn.004024f6
fcn.00404870
fcn.004048b0
fcn.004048f0
fcn.00404910
fcn.00404940
fcn.00404950
fcn.00404970
fcn.00404990
fcn.00404a60
fcn.00404a70
```

```
0x404795    mov rax, qword [rip + 0x218304]
0x40479c    mov rdi, qword [rsp + 0x28]
0x4047a1    lea rsi, qword [rsp + 0x38]
0x4047a6    mov edx, 1
0x4047ab    mov rcx, r13
0x4047ae    add qword [rsp + 0x38], 1
0x4047b4    mov qword [rip + 0x2182e5], r13
0x4047bb    mov qword [r13 + 0x20], rax
0x4047bf    mov rax, qword [rsp + 0x40]
0x4047c4    mov qword [r13 + 8], rax
0x4047c8    call 0x404a70
0x4047cd    cmp al, 1
0x4047cf    sbb edx, edx
0x4047d1    and edx, 2
0x4047d4    add edx, 3
0x4047d7    jmp 0x404362
; JMP XREF from 0x404763
0x4047dc    mov rax, qword [rsp + 0x40]
0x4047e1    movsxd rcx, r14d
0x4047e4    mov rdi, qword [rsp + 0x28]
0x4047e9    shl rcx
0x4047ed    lea rsi, qword [rsp + 0x38]
0x4047f2    xor edx, edx
0x4047f4    add rcx, 0x61bc80
0x404804    xor edx, edx
0x404806    test al, al
0x404808    jne 0x40436b
; JMP XREF from 0x404775
0x40480e    lea rdi, qword [rsp + 0xf0]
0x404816    call 0x40eaa0
0x40481b    xor edi, edi
0x40481d    mov r14, rax
```

```
> entry0 > 0x4047d1 > 0x4047c4 > 0x4047bf > 0x4047c4 > 0x4047c8 > 0x4047bf
```

```
file      /bin/ls
type      EXEC (Executabl
pic       false
canary    true
nx        true
crypto    false
va        true
root      elf
class     ELF64
lang      c
arch      x86
bits      64
machine   AMD x86-64 arch
os        linux
subsys    linux
endian    little
strip     true
static    false
linenum   false
lsyms     false
relocs    false
rpath     NONE
type      EXEC (Executabl
          linux
arch      AMD x86-64 arch
bits      64
endian    little
file      /bin/ls
fd        6
size      0x1c6f8
mode      r--
```

```
> ar=
  r15 0x00000000        r14 0x00000000        r13 0x00000000
  r12 0x00000000        rbp 0x00000000        rbx 0x00000000
  r11 0x00000000        r10 0x00000000         r9 0x00000000
   r8 0x00000000        rax 0x00000000        rcx 0x00000000
  rdx 0x00000000        rsi 0x00000000        rdi 0x00000000
 orax 0x00000000        rip 0x00000000        rflags =
  rsp 0x00000000
```

```
echo 0 > /proc/sys/kernel/randomize_va_space
```

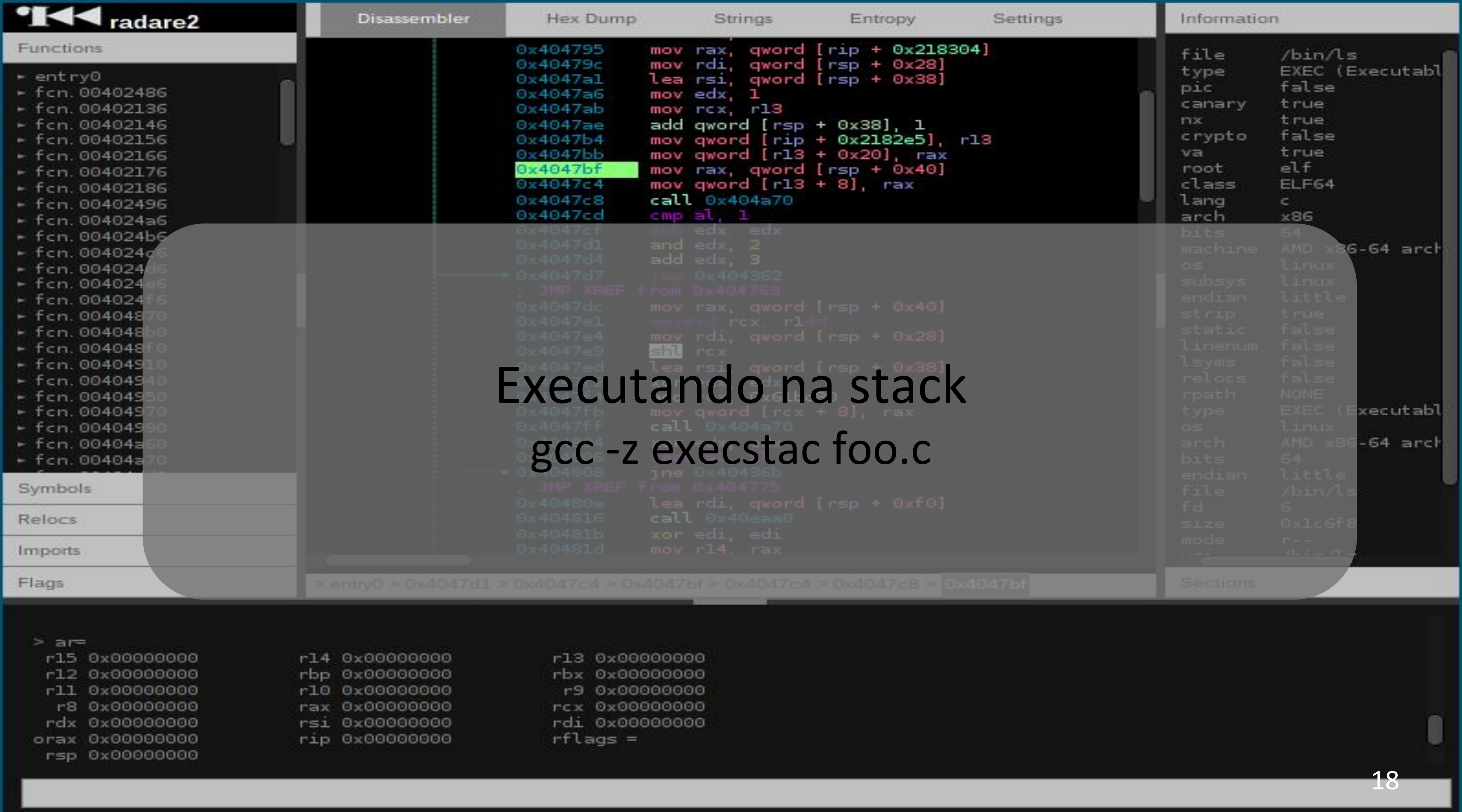
E se a função hack não existisse?

- Shellcode
  - Queremos aproveitar a variável str e injetar código nela
  - Podemos obter o endereço de str?
    - Se fosse global...

```
bits 64
global _start
section .text

_start:
    push 0x646e7770; ‘dnwp’
    mov rax, 1 ; write
    mov rdi, 1
    mov rsi, rsp
    mov rdx, 4
    syscall

    mov rax, 60 ; exit
    mov rdi, 0
    syscall
```



```
entry0
fcn.00402486
fcn.00402136
fcn.00402146
fcn.00402156
fcn.00402166
fcn.00402176
fcn.00402186
fcn.00402496
fcn.004024a6
fcn.004024b6
fcn.004024c6
fcn.004024d6
fcn.004024e6
fcn.004024f6
fcn.00404870
fcn.004048b0
fcn.004048f0
fcn.00404910
fcn.00404940
fcn.00404950
fcn.00404970
fcn.00404990
fcn.00404a60
fcn.00404a70
```

```
0x404795    mov rax, qword [rip + 0x218304]
0x40479c    mov rdi, qword [rsp + 0x28]
0x4047a1    lea rsi, qword [rsp + 0x38]
0x4047a6    mov edx, 1
0x4047ab    mov rcx, r13
0x4047ae    add qword [rsp + 0x38], 1
0x4047b4    mov qword [rip + 0x2182e5], r13
0x4047bb    mov qword [r13 + 0x20], rax
0x4047bf    mov rax, qword [rsp + 0x40]
0x4047c4    mov qword [r13 + 8], rax
0x4047c8    call 0x404a70
0x4047cd    cmp al, 1
0x4047cf    sbb edx, edx
0x4047d1    and edx, 2
0x4047d4    add edx, 3
0x4047d7    jmp 0x404362
; JMP XREF from 0x404763
0x4047dc    mov rax, qword [rsp + 0x40]
0x4047e1    movsxd rcx, r14d
0x4047e4    mov rdi, qword [rsp + 0x28]
0x4047e9    shl rcx
0x4047ed    lea rsi, qword [rsp + 0x38]
0x4047fb    mov qword [rcx + 8], rax
0x4047ff    call 0x404a70
0x404808    jne 0x40436b
; JMP XREF from 0x404775
0x40480e    lea rdi, qword [rsp + 0xf0]
0x404816    call 0x40eaa0
0x40481b    xor edi, edi
0x40481d    mov r14, rax
```

```
> entry0 > 0x4047d1 > 0x4047c4 > 0x4047bf > 0x4047c4 > 0x4047c8 > 0x4047bf
```

```
file      /bin/ls
type      EXEC (Executabl
pic       false
canary    true
nx        true
crypto    false
va        true
root      elf
class     ELF64
lang      c
arch      x86
bits      64
machine   AMD x86-64 arch
os        linux
subsys    linux
endian    little
strip     true
static    false
linenum   false
lsyms     false
relocs    false
rpath     NONE
type      EXEC (Executabl
os        linux
arch      AMD x86-64 arch
bits      64
endian    little
file      /bin/ls
fd        6
size      0x1c6f8
mode      r--
```

```
> ar=
 r15 0x00000000       r14 0x00000000       r13 0x00000000
 r12 0x00000000       rbp 0x00000000       rbx 0x00000000
 r11 0x00000000       r10 0x00000000        r9 0x00000000
  r8 0x00000000       rax 0x00000000       rcx 0x00000000
 rdx 0x00000000       rsi 0x00000000       rdi 0x00000000
orax 0x00000000       rip 0x00000000       rflags =
 rsp 0x00000000
```

Executando na stack

gcc -z execstac foo.c

Injetando shellcode

- Precisamos sempre fazer o código assembly?
    - Shell Storm - http://shell-storm.org/shellcode/
    - Pwntools - Shellcraft

41414141GANESH

2748 segmentation fault (core dumped)

ICMC / USP - São Carlos, SP

https://ganesh.icmc.usp.br/

ganesh@icmc.usp.br