Summary

This challenge has been stripped and has a function that is called, but doesn't execute the flag process.

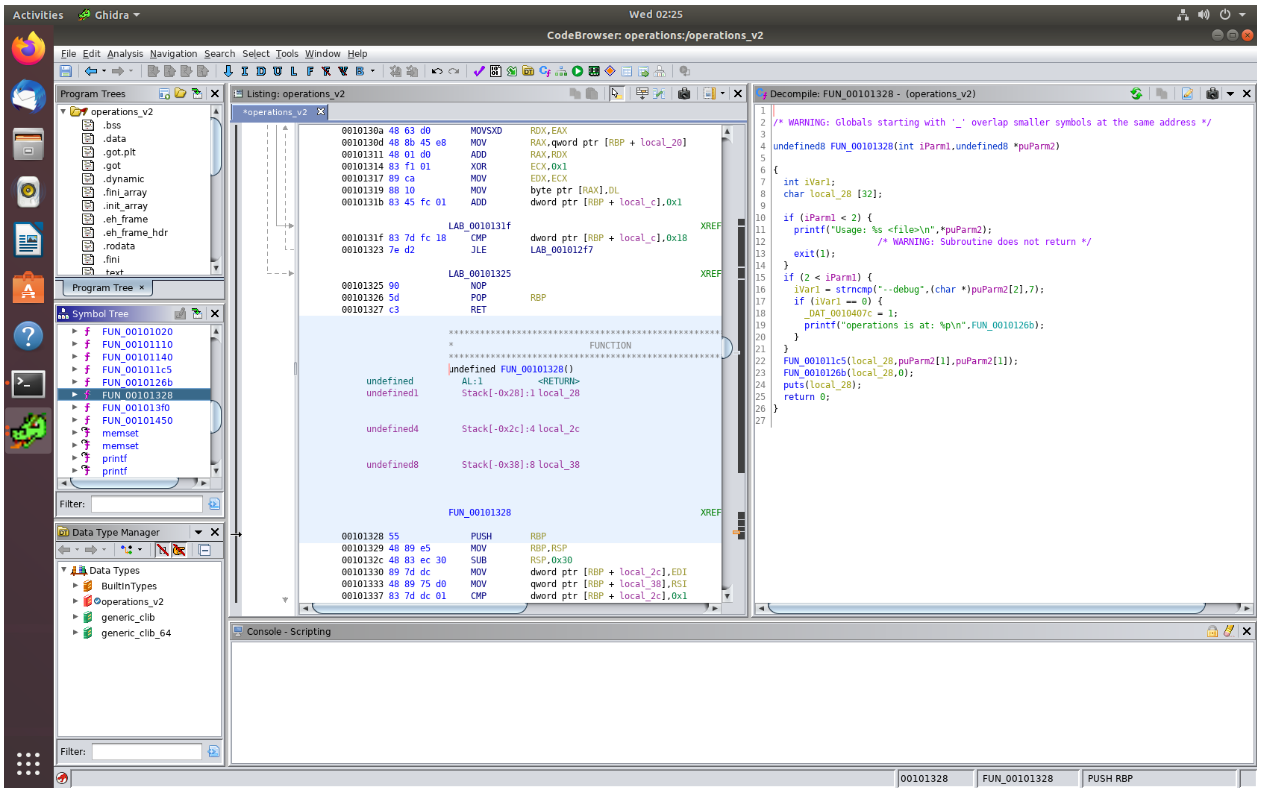
Information gathering

We get two files, `flag.txt` and `operations`. `flag.txt` contains a weird string: `eobdX{(mNp,|F)iDs5uhNor|` and `operations` is a stripped 64bit binary:

```
operations: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/l, for GNU/Linux 3.2.0,
BuildID[sha1]=94c8987eba9458d1ece60c5991a2c8c77adfbd69, stripped
```

Disassembling the binary

We can use `ghidra` to look at the predicted source code to understand how it works. There are two interesting functions; `FUN_00101328` and `FUN_0010126b`. Here's the disassembly:



CodeBrowser: operations:/operations_v2

```
0010130a 48 63 d0        MOVSXD    RDX,EAX
0010130d 48 8b 45 e8     MOV       RAX,qword ptr [RBP + local_20]
00101311 48 01 d0        ADD       RAX,RDX
00101314 83 f1 01        XOR       ECX,0x1
00101317 89 ca           MOV       EDX,ECX
00101319 88 10           MOV       byte ptr [RAX],DL
0010131b 83 45 fc 01     ADD       dword ptr [RBP + local_c],0x1

                     LAB_0010131f                                XREF
0010131f 83 7d fc 18     CMP       dword ptr [RBP + local_c],0x18
00101323 7e d2           JLE       LAB_001012f7

                     LAB_00101325                                XREF
00101325 90              NOP
00101326 5d              POP       RBP
00101327 c3              RET

                     ******************************************
                     *                 FUNCTION
                     ******************************************
                     undefined FUN_00101328()
       undefined        AL:1            <RETURN>
       undefined1       Stack[-0x28]:1 local_28
       undefined4       Stack[-0x2c]:4 local_2c
       undefined8       Stack[-0x38]:8 local_38
                     FUN_00101328                                XREF
00101328 55              PUSH      RBP
00101329 48 89 e5        MOV       RBP,RSP
0010132c 48 83 ec 30     SUB       RSP,0x30
00101330 89 7d dc        MOV       dword ptr [RBP + local_2c],EDI
00101333 48 89 75 d0     MOV       qword ptr [RBP + local_38],RSI
00101337 83 7d dc 01     CMP       dword ptr [RBP + local_2c],0x1
```

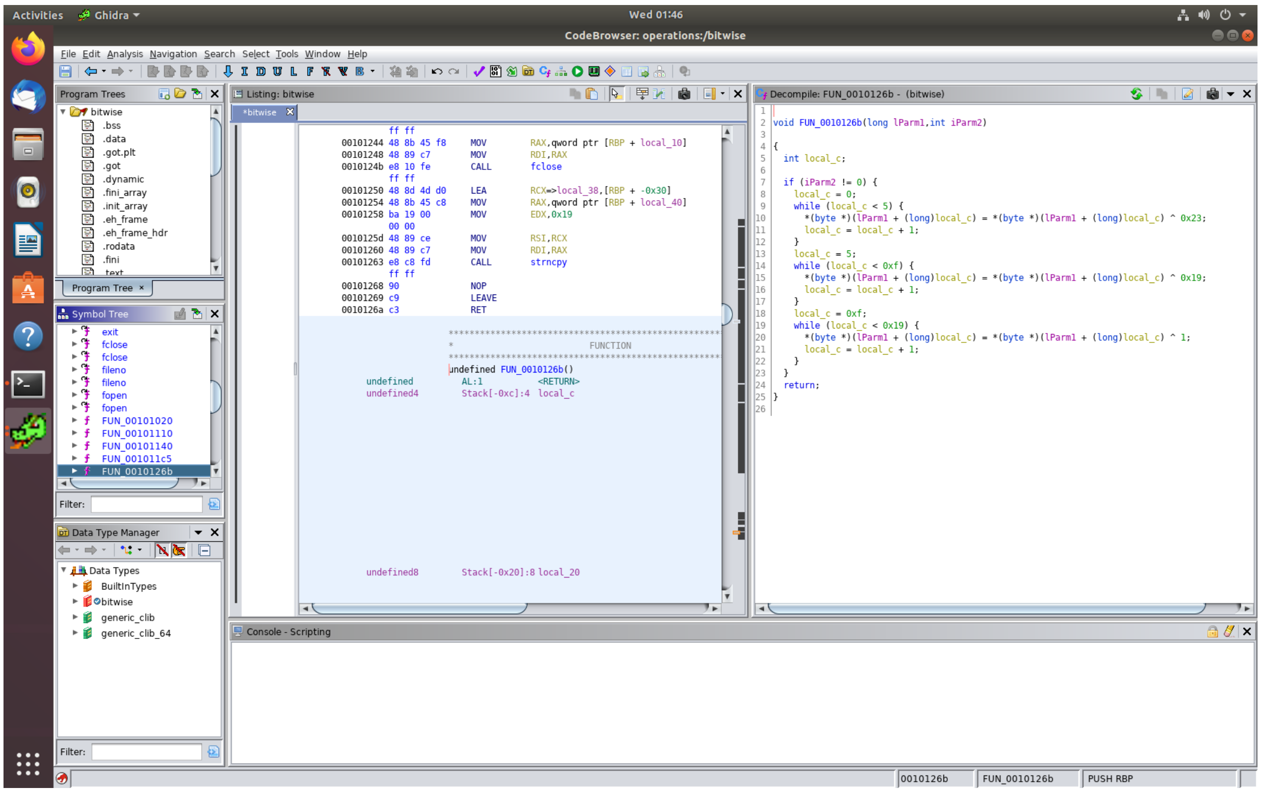
Decompile: FUN_00101328 - (operations_v2)

```c
/* WARNING: Globals starting with '_' overlap smaller symbols at the same address */

undefined8 FUN_00101328(int iParm1,undefined8 *puParm2)

{
  int iVar1;
  char local_28 [32];

  if (iParm1 < 2) {
    printf("Usage: %s <file>\n",*puParm2);
                    /* WARNING: Subroutine does not return */
    exit(1);
  }
  if (2 < iParm1) {
    iVar1 = strncmp("--debug",(char *)puParm2[2],7);
    if (iVar1 == 0) {
      _DAT_0010407c = 1;
      printf("operations is at: %p\n",FUN_0010126b);
    }
  }
  FUN_001011c5(local_28,puParm2[1],puParm2[1]);
  FUN_0010126b(local_28,0);
  puts(local_28);
  return 0;
}
```



CodeBrowser: operations:/bitwise

```
                 ff ff
00101244 48 8b 45 f8     MOV       RAX,qword ptr [RBP + local_10]
00101248 48 89 c7        MOV       RDI,RAX
0010124b e8 10 fe        CALL      fclose
         ff ff
00101250 48 8d 4d d0     LEA       RCX=>local_38,[RBP + -0x30]
00101254 48 8b 45 c8     MOV       RAX,qword ptr [RBP + local_40]
00101258 ba 19 00        MOV       EDX,0x19
         00 00
0010125d 48 89 ce        MOV       RSI,RCX
00101260 48 89 c7        MOV       RDI,RAX
00101263 e8 c8 fd        CALL      strncpy
         ff ff
00101268 90              NOP
00101269 c9              LEAVE
0010126a c3              RET

                     ******************************************
                     *                 FUNCTION
                     ******************************************
                     undefined FUN_0010126b()
       undefined        AL:1            <RETURN>
       undefined4       Stack[-0xc]:4  local_c

       undefined8       Stack[-0x20]:8 local_20
```

Decompile: FUN_0010126b - (bitwise)

```c
void FUN_0010126b(long lParm1,int iParm2)

{
  int local_c;

  if (iParm2 != 0) {
    local_c = 0;
    while (local_c < 5) {
      *(byte *)(lParm1 + (long)local_c) = *(byte *)(lParm1 + (long)local_c) ^ 0x23;
      local_c = local_c + 1;
    }
    local_c = 5;
    while (local_c < 0xf) {
      *(byte *)(lParm1 + (long)local_c) = *(byte *)(lParm1 + (long)local_c) ^ 0x19;
      local_c = local_c + 1;
    }
    local_c = 0xf;
    while (local_c < 0x19) {
      *(byte *)(lParm1 + (long)local_c) = *(byte *)(lParm1 + (long)local_c) ^ 1;
      local_c = local_c + 1;
    }
  }
  return;
}
```

`FUN_00101328` looks to be `main`. Looking at `FUN_0010126b` we can see some xor operations happening.

Xorring the flag

Let's try to xor the flag with the same numbers and locations as the binary:

```
fd = open('flag.txt', 'r')
input = fd.read()
flag = ''
for i in range(0, 5):
  flag += chr(ord(input[i]) ^ 0x23)
for i in range(5, 15):
  flag += chr(ord(input[i]) ^ 0x19)
for i in range(15, 25):
  flag += chr(ord(input[i]) ^ 1)

print('Flag: {}'.format(flag))
```

```
$ python rev.py
Flag: FLAG{b1tWi5e_0pEr4tiOns}
```

Got the flag!