

# ISCC2025 WriteUp 提交模板

## REVERSE+uglyCpp

### 解题思路

1. 用 ida 打开文件，在 strings 看到输入 flag，我们跳转对应地方 (main 函数)

Address	Length	Type	String
.rodata:...	00000006	C	wrong
.rodata:...	00000006	C	right
.rodata:...	00000012	C	Input your flag:
.rodata:...	00000031	C	cannot create std::vector larger than max_size()

```
1 int __fastcall main(int argc, const char **argv, const char **envp)
2 {
3     __int64 v3; // rax
4     __int64 v4; // rdx
5     __int64 v5; // rax
6     _BYTE v7[16]; // [rsp+10h] [rbp-100h] BYREF
7     _BYTE v8[32]; // [rsp+20h] [rbp-F0h] BYREF
8     _BYTE v9[32]; // [rsp+40h] [rbp-D0h] BYREF
9     _BYTE v10[32]; // [rsp+60h] [rbp-B0h] BYREF
10    _BYTE v11[32]; // [rsp+80h] [rbp-90h] BYREF
11    _BYTE v12[32]; // [rsp+A0h] [rbp-70h] BYREF
12    _DWORD v13[10]; // [rsp+C0h] [rbp-50h] BYREF
13    unsigned __int64 v14; // [rsp+E8h] [rbp-28h]
14
15    v14 = __readfsqword(0x28u);
16    std::string::basic_string(v12, argv, envp);
17    qmemcpy(v13, "key1key2key3key4", 16);
18    std::allocator<unsigned int>::allocator(v11);
19    std::vector<unsigned int>::vector(v8, v13, 4LL, v11);
20    std::allocator<unsigned int>::~~allocator(v11);
21    v3 = std::operator<<<std::char_traits<char>>((std::ostream *)&std::cout);
22    std::ostream::operator<<<(v3, std::endl, char, std::char_traits<char>);
23    std::operator>><char>((std::istream *)&std::cin);
24    ZNK17g3uSFzT86rFKFJog2MU1RKNSt7_cxx1112basic_stringIcSt11char_traitsIcESaIcEEEE_c1
25    std::string::basic_string(v13, &g3uSFzT86rFKFJog2, v4);
26    std::shared_ptr<strc>::shared_ptr(v11, v7);
27    std::function<void ()(std::shared_ptr<strc>, std::string &)>::operator()(&GxZuWxsXX1
28    std::shared_ptr<strc>::~~shared_ptr(v11);
29    ZNK17KDXgsB2q4YQad5xBZMU1RKNSt7_cxx1112basic_stringIcSt11char_traitsIcESaIcEEEE_c1
30    if ( std::string::size(v12) == 36 )
31    {
32        ZNK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUIRKSt6vectorIjSaIjEEE_cIES3_(v11, &mJ6Xq4ExTMs4qa
33        ZNK28gxoPJ4FNZcYkUgP7wE96Z9Pzuw8MU1RKNSt6vectorIjSaIjEES3_mE_cIES3_S3_m(
34        v10,
35        &gxoPJ4FNZcYkUgP7wE96Z9Pzuw8,
36        v9,
37        v11,
38        0x1B5E5A3C8E2F4D6ALL);
39        std::vector<unsigned int>::~~vector(v11);
40        ZNK12S4V3u5wVUXnyMU1RSt6vectorIjSaIjEEE_cIES2_(&S4V3u5wVUXny, v10);
41        std::vector<unsigned int>::~~vector(v10);
42    }
43    else
44    {
45        v5 = std::operator<<<std::char_traits<char>>((std::ostream *)&std::cout);
```

2. 能看到有一个密钥 key1key2key3key4，用户输入字符串存储到 v12，然后调用外部函数 GxZuWxsXX1sb 等处理，最后检查输入字符串长度是否为 36 字节。如果长度为 36 就执行相关函数。逐一查看。
3. ZNK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUIRKSt6vectorIjSaIjEEE\_cIES3\_ 是密钥生成逻辑，对密钥进行扩展，通过 20 轮迭代生成密钥片段将原始密钥扩展到 44。生成初始序列，每轮通过 0x61C88647 递减生成数值，调用

ZZnK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUIRKSt6vectorIjSaIjEEE\_cIES3\_ENKUIvE0\_cIEv 反转序列顺序。通过 ZZnK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUIRKSt6vectorIjSaIjEEE\_cIES3\_ENKUIRmS5\_RjS6\_RS1\_E1\_cIES5\_S5\_S6\_S6\_S7\_进行双向量混合，最终生成 44 字节扩展密钥。

```

1 __int64 __fastcall ZZnK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUIRKSt6vectorIjSaIjEEE_cIES3_(__int64
2 {
3     int v5; // [rsp+28h] [rbp-98h] BYREF
4     int v6; // [rsp+2Ch] [rbp-94h] BYREF
5     __int64 v7; // [rsp+30h] [rbp-90h] BYREF
6     __int64 v8; // [rsp+38h] [rbp-88h] BYREF
7     __int64 v9; // [rsp+40h] [rbp-80h] BYREF
8     __int64 v10; // [rsp+48h] [rbp-78h] BYREF
9     _QWORD v11[2]; // [rsp+50h] [rbp-70h] BYREF
10    _QWORD v12[2]; // [rsp+60h] [rbp-60h] BYREF
11    _QWORD v13[4]; // [rsp+70h] [rbp-50h] BYREF
12    _BYTE v14[24]; // [rsp+90h] [rbp-30h] BYREF
13    unsigned __int64 v15; // [rsp+A8h] [rbp-18h]
14
15    v15 = __readfsqword(0x28u);
16    v7 = std::vector<unsigned int>::size(a3);
17    v11[1] = 20LL;
18    v8 = 44LL;
19    std::allocator<unsigned int>::allocator(v14);
20    std::vector<unsigned int>::vector(a1, 44LL, v14);
21    std::allocator<unsigned int>::~~allocator(v14);
22    v12[0] = a1;
23    v12[1] = &v8;
24    v9 = a3;
25    v13[0] = &v8;
26    v13[1] = &v7;
27    v13[2] = a1;
28    ZZnK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUIRKSt6vectorIjSaIjEEE_cIES3_ENKUIvE0_cIEv(v12);
29    ZZnK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUIRKSt6vectorIjSaIjEEE_cIES3_ENKUIvE0_cIEv(v14, &v9);
30    v10 = 0LL;
31    v11[0] = 0LL;
32    v5 = 0;
33    v6 = 0;
34    ZZnK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUIRKSt6vectorIjSaIjEEE_cIES3_ENKUIRmS5_RjS6_RS1_E1_cI
35        v13,
36        &v10,
37        v11,
38        &v5,
39        &v6,
40        v14);
41    std::vector<unsigned int>::~~vector(v14);

```

- 在数据段中发现硬编码的异或密钥，对应 9 个 32 位无符号整数，查看内存数据段提取数值：xor=[0x3ED6325B, 0xD709BF17, 0xE3F27E18, 0xA0870791, 0x0146D6F9, 0x7C6140FF, 0x10B69406, 0x94DDE0F6, 0x40B2BB6C]。
- 最后一个函数是验证函数，处理结果要和 v12 完全一致才输入有效。所以整个流程是用户输入→数据处理→与密钥异或→比对结果是否等于硬编码的 v12。

```

9 | __int64 v9; // [rsp+20h] [rbp-80h] BYREF
10 | __int64 v10; // [rsp+28h] [rbp-78h]
11 | _BYTE v11[32]; // [rsp+30h] [rbp-70h] BYREF
12 | _DWORD v12[10]; // [rsp+50h] [rbp-50h] BYREF
13 | unsigned __int64 v13; // [rsp+78h] [rbp-28h]
14
15 | v13 = __readfsqword(0x28u);
16 | v12[0] = 1555645228;
17 | v12[1] = -1805464030;
18 | v12[2] = -776254107;
19 | v12[3] = -1830657342;
20 | v12[4] = 1912709250;
21 | v12[5] = 1226013110;
22 | v12[6] = 586150225;
23 | v12[7] = -944784715;
24 | v12[8] = 1927997992;
25 | std::allocator<unsigned int>::allocator(&v9);
26 | std::vector<unsigned int>::vector(v11, v12, 9LL, &v9);
27 | std::allocator<unsigned int>::~~allocator(&v9);
28 | v2 = std::vector<unsigned int>::size(a2);
29 | if ( v2 == std::vector<unsigned int>::size(v11) )

```

6. 写脚本运行，得到一个 32 位的字符串

```

v = [1555645228, -1805464030,
     -776254107, -1830657342, 1912709250,
     1226013110, 586150225, -944784715,
     1927997992]

v = [i & 0xffffffff for i in v] # Convert to unsigned 32-bit integers

xor = [0x3ED6325B, 0xD709BF17,
       0xE3F27E18, 0xA0870791, 0x0146D6F9,
       0x7C6140FF, 0x10B69406, 0x94DDE0F6,
       0x40B2BB6C]

for i in range(len(v)):
    v[i] ^= xor[i]

# 确保flag被正确定义
flag = "".join([i.to_bytes(length=4, byteorder="little").decode() for i in v])
print("解密结果:", flag)

```

test x

E:\reverse\python\venv\Scripts\python.exe E:\reverse\python\reverse\test.py  
解密结果: wqob5qkC}3I2See2{v6sI9r5WeY2CZrSDQX2

7. 不一样是因为密钥扩展前进行 ZNK17g3uSFZt86rfKFJog2MUIRKNSt7\_cxx1112basic\_stringlcSt11char\_traitsl cESalcEEEE\_cIES6\_处理了输入，进行了置换提取处理打乱的置换表，"5p6h7q8d9risbtjuevkwxlyfzm0c1n2g3o4"再进行映射

得到最终 flag: ISCC{WDbleGrYrXqq32e2vs95e2ZSQ2wo5k}

```
E:\reverse\python\venv\Scripts\python.exe E:\reverse\python\reverse\test.py
解密结果: wqob5qkC}3I2See2{vGsI9r5WeY2CZrSDQX2
ISCC{WDbIeGrYrXqq32e2vs95e2ZSQ2wo5k}
```

## Exp

```
v = [1555645228, -1805464030,
     -776254107, -1830657342, 1912709250,
     1226013110, 586150225, -944784715,
     1927997992]
```

```
v = [i & 0xffffffff for i in v]
```

```
xor = [0x3ED6325B, 0xD709BF17,
       0xE3F27E18, 0xA0870791, 0x0146D6F9,
       0x7C6140FF, 0x10B69406, 0x94DDE0F6,
       0x40B2BB6C]
```

```
for i in range(len(v)):
    v[i] ^= xor[i]
```

```
# 确保 flag 被正确定义
```

```
flag = "".join([i.to_bytes(length=4, byteorder="little").decode() for i in v])
```

```
print("解密结果:", flag)
```

```
#
```

```
c_table = "5p6h7q8d9risbtjuevkwaxlyfzm0c1n2g3o4"
```

```
table = "abcdefghijklmnopqrstuvwxyz0123456789"
```

```
res = ""
```

```
for i in range(len(table)):
    tmp = c_table.index(table[i])
    res += flag[tmp]
```

```
print(res)
```