MoeCTF 逆向工程官方题解

逆向工程入门指北

用 IDA Pro (以下简称 IDA) 打开附件, 在字符串表 (Shift+F12) 中搜索 flag 或 moectf 即得.

speed

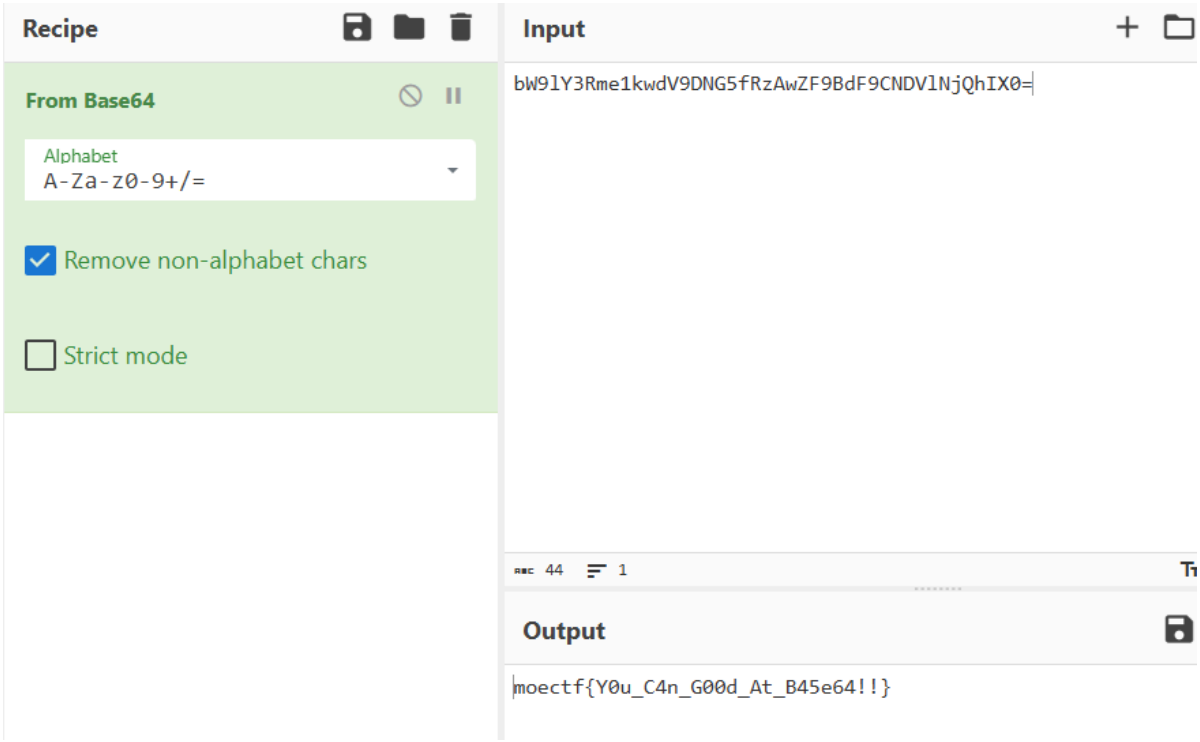
考察动态调试. 首先是最基本的 IDA 打开, **F5**反编译. 要注意这里的主函数在 WinMain. 里面是新建一个调用 WndProc 函数的窗口, 然后会调用 Sleep(1) 然后 DestroyWindow, 导致窗口只显示 1ms.

进入 WndProc 分析可以知道是把 flag 打印在窗口里, 打印之前有一个 RC4 解密, 但是其实并不需要知道怎么解, 只需要在 RC4 解密后**查看内存**, 或者下一个**断点**, 程序会自己把 flag 打印在屏幕上.

```
34  else if ( Msg == 15 )
35  {
36    hdc = BeginPaint(hWnd, &Paint);
37    strcpy(Destination, "Your flag is ");
38    v7 = 0;
39    v8 = 0LL;
40    v9 = 0LL;
41    v10 = 0LL;
42    v11 = 0LL;
43    v12 = 0LL;
44    v13 = 0LL;
45    v14 = 0LL;
46    v15 = 0LL;
47    v16 = 0LL;
48    v17 = 0LL;
49    v18 = 0LL;
50    v19 = 0LL;
51    v20 = 0LL;
52    v21 = 0LL;
53    *(_QWORD *)Source = 0x7F1B3E885EF9160LL;
54    v25 = 0x2CD336BCB0464A89LL;
55    v26[0] = 0xEF5FC91642917EE1uLL;
56    *(_QWORD *)((char *)v26 + 6) = 0x739D40A4E356EF5FLL;
57    strcpy(mylittlepony, "mylittlepony");
58    n12 = 12;
59    v27 = strlen(Source);
60    RC4Crypt((unsigned __int8 *)Source, v27, (const unsigned __int8 *)mylittlepony, 12);
61    strcat(Destination, Source);
62    GetClientRect(hWnd, &Rect);
63    DrawTextA(hdc, Destination, -1, &Rect, DT_CENTER | DT_VCENTER | DT_WORD_ELLIPSIS);
64    EndPaint(hWnd, &Paint);
65    return 0LL;
66  }
```

base

base64标准编码



catch

需要了解 C++ 异常处理机制.

容易分析发现 try 里面一定会抛出异常，而sub_114514是一个没有副作用的函数, 也就是说 try 里面相当于什么都没干. 我们可以把 try 和 cleanup 的部分全部 patch 成 nop 再重新反编译, 可以看到真正的逻辑.

```
.text:00000001400014D6 ; __unwind { // __gxx_personality_seh0
.text:00000001400014D6                 push    rbp
.text:00000001400014D7                 push    rsi
.text:00000001400014D8                 push    rbx
.text:00000001400014D9                 sub     rsp, 30h
.text:00000001400014DD                 lea     rbp, [rsp+30h]
.text:00000001400014E2                 lea     rax, aMyFlagIsHidden ; "my flag is hidden in this program. Can "...
.text:00000001400014E9                 mov     rcx, rax        ; char *
.text:00000001400014EC                 call    _Z6printfPKcz   ; printf(char const*,...)
.text:00000001400014F1 ;   try {
.text:00000001400014F1                 call    _Z10sub_114514v ; sub_114514(void)
.text:00000001400014F1 ;   } // starts at 1400014F1
.text:00000001400014F6                 mov     ecx, 10h        ; thrown_size
.text:00000001400014FB                 call    __cxa_allocate_exception
.text:0000000140001500                 mov     rbx, rax
.text:0000000140001503                 lea     rax, aNothingButErro ; "nothing but error"
.text:000000014000150A                 mov     rdx, rax        ; char *
.text:000000014000150D                 mov     rcx, rbx        ; this
.text:0000000140001510 ;   try {
.text:0000000140001510                 call    _ZNSt11logic_errorC1EPKc ; std::logic_error::logic_error(char const*)
.text:0000000140001510 ;   } // starts at 140001510
.text:0000000140001515                 mov     r8, cs:_refptr__ZNSt11logic_errorD1Ev ; void (*)(void *)
.text:000000014000151C                 lea     rax, _ZTISt11logic_error ; `typeinfo for'std::logic_error
.text:0000000140001523                 mov     rdx, rax        ; lptinfo
.text:0000000140001526                 mov     rcx, rbx        ; void *
.text:0000000140001529 ;   try {
.text:0000000140001529                 call    __cxa_throw
.text:0000000140001529 ;   } // starts at 140001529
.text:000000014000152E ; ---------------------------------------------------------------------------
.text:000000014000152E ;   cleanup() // owned by 140001510
.text:000000014000152E                 mov     rsi, rax
.text:0000000140001531                 mov     rcx, rbx        ; void *
.text:0000000140001534                 call    __cxa_free_exception
.text:0000000140001539                 mov     rax, rsi
.text:000000014000153C                 jmp     short $+2
.text:000000014000153E ; ---------------------------------------------------------------------------
.text:000000014000153E
.text:000000014000153E loc_14000153E:                          ; CODE XREF: solve(void)+66↑j
.text:000000014000153E ;   catch(...) // owned by 1400014F1    ; void *
.text:000000014000153E ;   catch(...) // owned by 140001529
```

```
void __noreturn solve(void)
{
  void *v0; // rax
  int i_1; // [rsp+24h] [rbp-Ch]
  int C; // [rsp+28h] [rbp-8h]
  int i; // [rsp+2Ch] [rbp-4h]

  v0 = (void *)printf("my flag is hidden in this program. Can you find it?\n");
  _cxa_begin_catch(v0);
  i_1 = strlen(solve(void)::hidesuwa);          // "zbrpgs{F4z3_Ge1px_jvgu_@sybjre_qrfhjn}"
  for ( i = 0; i < i_1; ++i )
  {
    C = (unsigned __int8)solve(void)::hidesuwa[i];// "zbrpgs{F4z3_Ge1px_jvgu_@sybjre_qrfhjn}"
    if ( islower(C) )
    {
      C = (C - 84) % 26 + 97;
    }
    else if ( isupper(C) )
    {
      C = (C - 52) % 26 + 65;
    }
    solve(void)::hidesuwa[i] = C;               // "zbrpgs{F4z3_Ge1px_jvgu_@sybjre_qrfhjn}"
  }
  printf("so you didn't catch me?\n");
  _cxa_end_catch();
}
```

发现是一个 rot13, 还原即可拿到 flag.

upx

`upx -d` 脱壳
程序中有一个需要注意的点是`fgets()`,很多人被恶心到了， 出题人给大家道歉(T人T)

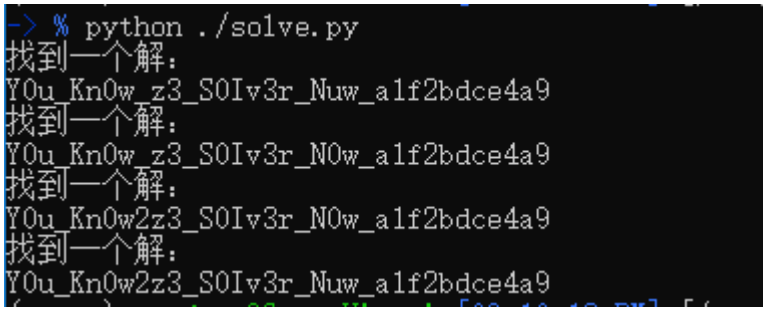
```c
#include <stdio.h>

int main() {
    unsigned char b[] = {
        0x23, 0x2b, 0x27, 0x36, 0x33, 0x3c, 0x03, 0x48,
        0x64, 0x0b, 0x1d, 0x76, 0x7b, 0x10, 0x0b, 0x3a,
        0x3f, 0x65, 0x76, 0x29, 0x15, 0x37, 0x1c, 0x0a,
        0x08, 0x21, 0x3e, 0x3c, 0x3d, 0x16, 0x0b, 0x24,
        0x29, 0x24, 0x56
    };

    unsigned char flag[36] = { 0 };

    // 最后一个字符是换行符 '\n'
    flag[35] = '\n';

    // 从后往前解密
    for (int i = 34; i >= 0; i--) {
        flag[i] = b[i] ^ 0x21 ^ flag[i + 1];
    }

    // 打印 flag（不包含换行）
    printf("Flag: %.35s\n", flag);

    return 0;
}
//moectf{YOu_c4n_unp4ck_It_vvith_upx}
```

ez3

需要一点耐心的题目. 考察 z3 的使用

去 check 函数里找检验逻辑, 发现是一个非线性变换, 不能直接倒着解.

学习并使用 z3-solver 可以求出 4 组解, 其中第二个是正确的 flag.



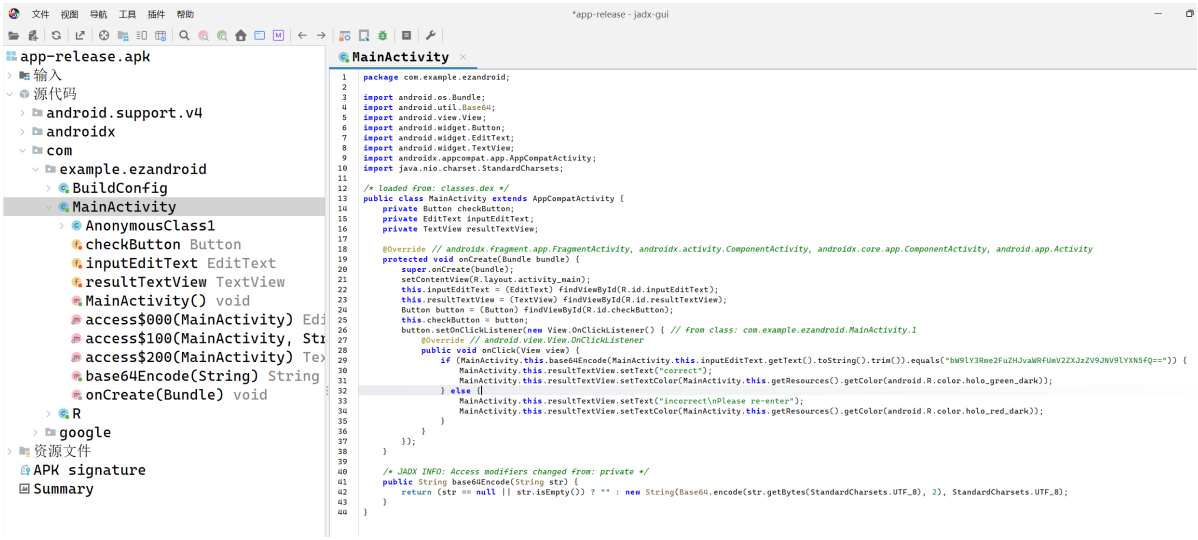
具体脚本如下, 其中 `solver.add(Or([s[i] != model1[s[i]] for i in range(N)]))` 的作用就是把每次求出来的解又作为新的限制, 从而可以求出其余的可行解.

```python
from z3 import *

N = 34

solver = Solver()

a=[45488,22136,32754,41778,41192,13900,11220,51454,19068,24,11236,16708,15270,48780,36734,13816,25002,11082,26664,45982,46402,13292,51160,17548,37648,34824,44500,15554,1942,51520,20018,20014,37450,23388]
s = [BitVec(f's_{i}', 32) for i in range(N)]

for i in range(N):
    solver.add(And(s[i] >= 32, s[i] <= 127))

b = [0] * N
for i in range(N):
    b[i] = (s[i] + i) * 0xbabe
    if (i > 0):
        b[i] ^= b[i - 1] ^ 0x114514
    b[i] %= 0xcafe
    solver.add(b[i] == a[i])

# 检查是否有解
if solver.check() != sat:
    print("无解")
else:
    # 获取第一个解
    while (solver.check() == sat):
        model1 = solver.model()
        print("找到一个解: ")
        sol1 = [model1[s[i]].as_long() for i in range(N)]
        print(''.join([chr(c) for c in sol1]))

        solver.add(Or([s[i] != model1[s[i]] for i in range(N)]))
```

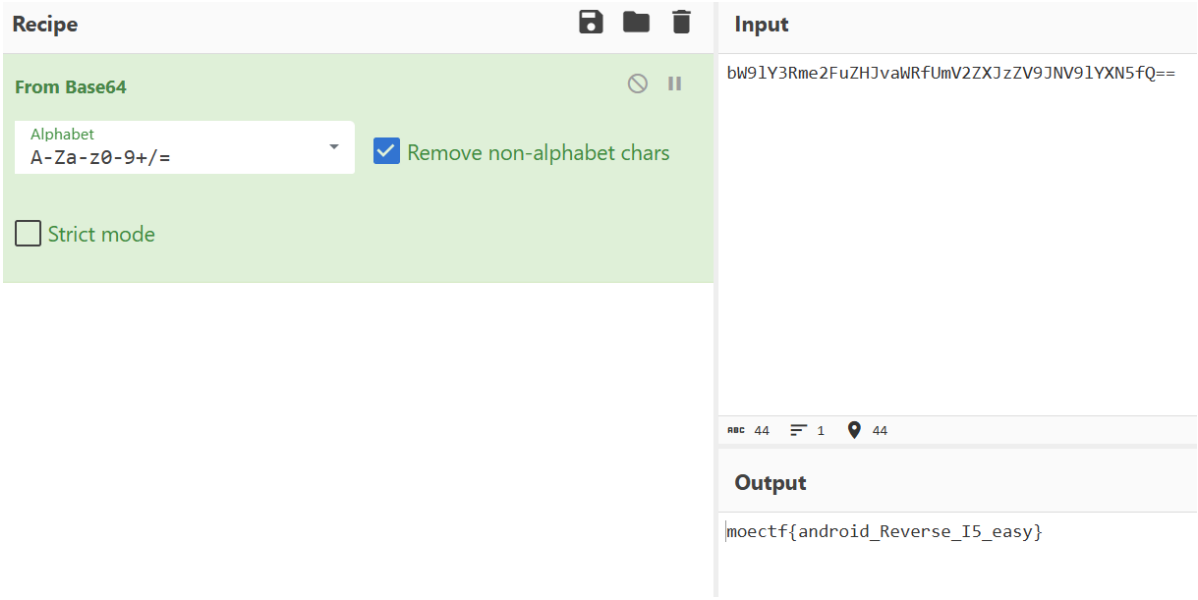
P.S. 比赛中有不少人锤我说怎么 flag 不对, 可能是因为不正确的 flag 错的太不明显了, 之后的出题人要引以为戒啊TwT

ezandroid

拖入jadx



里面是一个base64

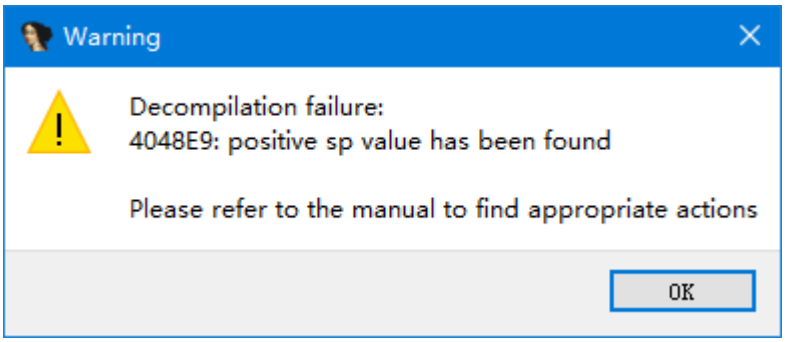


flower

考察花指令.

IDA打开会发现 main 函数有很长一段, check 判断了有没有反调试, 后面有判断是否zero==0, 如有抛出异常. 不过我们先看 solve 函数, 发现反编译失败了.

```
10  v9 = __readfsqword(0x28u);
11  check();
12  v3 = std::operator<<<std::char_traits<char>>(&std::cout, "Input your flag:\n> ");
13  std::ostream::operator<<(v3, std::flush<char,std::char_traits<char>>);
14  if ( !main::zero )
15  {
16    exception = (std::runtime_error *)__cxa_allocate_exception(0x10uLL);
17    std::runtime_error::runtime_error(exception, "Division by zero");
18    __cxa_throw(exception, (struct type_info *)&`typeinfo for'std::runtime_error, std::runtime_error::~runtime_error);
19  }
20  v5 = std::ostream::operator<<(&std::cout, (unsigned int)(10 / main::zero), (unsigned int)(10 % main::zero));
21  std::ostream::operator<<(v5, std::endl<char,std::char_traits<char>>);
22  std::string::basic_string(v7);
23  std::operator>><char>(&std::cin, v7);
24  std::string::basic_string(v8, v7);
25  solve((__int64)v8);
26  std::string::~string(v8);
27  std::string::~string(v7);
28  return 0;
29}
```



提示说检测到了正的 stack pointer 值.

回到汇编检查, 找到下面这个显眼的红色. 可以看到下面的两个跳转一定有一个会成立, 所以这个 call 是永远不会被执行的, 但是会导致IDA的调用栈识别错误. 直接按 Ctrl-N patch 即可.

```
.text:00000000004048E5 ; ---------------------------------------------------------------------------
.text:00000000004048E5
.text:00000000004048E5 loc_4048E5:                             ; CODE XREF: solve(std::string)+A6↑j
.text:00000000004048E5                 jz      short Label
.text:00000000004048E7                 jnz     short Label
.text:00000000004048E9                 call    near ptr Label+1
.text:00000000004048EE
.text:00000000004048EE Label:                                  ; CODE XREF: solve(std::string):loc_4048E5↑j
.text:00000000004048EE                                         ; solve(std::string)+DA↑j ...
.text:00000000004048EE                 lea     rax, [rbp+var_59]
.text:00000000004048EE _Z5solveNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE endp ; sp-analysis failed
.text:00000000004048EE
```

重新识别函数就能正确反编译了.

接着分析 solve 函数

```
53    *(_DWORD *)&v12[5] = std::string::length(a1);
54    if ( *(_DWORD *)&v12[5] == 32 )
55    {
56      *(_DWORD *)&v12[1] = 0;
57      while ( *(int *)&v12[1] < *(int *)&v12[5] )
58      {
59        v8 = (char *)std::string::operator[](a1, *(int *)&v12[1]);
60        if ( (unsigned int)encode(*v8) != enc[*(int *)&v12[1]] )
61        {
62          v9 = std::operator<<<std::char_traits<char>>(&std::cout, "Wrong!");
63          std::ostream::operator<<(v9, std::endl<char,std::char_traits<char>>);
64          return v16 - __readfsqword(0x28u);
65        }
66        ++*(_DWORD *)&v12[1];
67      }
68      if ( enc[*(int *)&v12[5]] )
69        v10 = std::operator<<<std::char_traits<char>>(&std::cout, "Wrong!");
70      else
71        v10 = std::operator<<<std::char_traits<char>>(&std::cout, "You Win!\n");
72      std::ostream::operator<<(v10, std::endl<char,std::char_traits<char>>);
73    }
74    else
75    {
76      v7 = std::operator<<<std::char_traits<char>>(&std::cout, "LENGTH ERROR");
77      std::ostream::operator<<(v7, std::endl<char,std::char_traits<char>>);
78    }
```

虽然有点丑陋, 不过容易看出这是一个长度为32的flag, 经过 encode 函数后, 再跟enc数组的每一位比较.

encode 函数如下, 每次返回与全局变量 key 做异或的结果, 然后 key 自增.

```
1  __int64 __fastcall encode(int a1)
2  {
3    int v1; // eax
4
5    v1 = key++;
6    return a1 ^ (unsigned int)v1;
7  }
```

那么只需要找到 key 的值就能得出 flag. 不难找到 enc 和 key 的初始值.

```
.data:00000000005D9140 ; _DWORD enc[100]
.data:00000000005D9140 enc             dd 4Fh, 1Ah, 59h, 1Fh, 5Bh, 1Dh, 5Dh, 6Fh, 7Bh, 47h, 7Eh
.data:00000000005D9140                                         ; DATA XREF: solve(std::string)+1F4↑o
.data:00000000005D9140                                         ; solve(std::string)+250↑o
.data:00000000005D916C                 dd 44h, 6Ah, 7, 59h, 67h, 0Eh, 52h, 8, 63h, 5Ch, 1Ah, 52h
.data:00000000005D919C                 dd 1Fh, 20h, 7Bh, 21h, 77h, 70h, 25h, 74h, 2Bh, 44h dup(0)
.data:00000000005D92D0                 public key
.data:00000000005D92D0 key             dd 23h                  ; DATA XREF: encode(int)+8↑r
.data:00000000005D92D0                                         ; encode(int)+14↑w ...
```

但是尝试解密会发现不对, 这是因为除零的异常处理里对 key 有修改

```
.text:0000000000404C6F loc_404C6F:                             ; CODE XREF: main+147↑j
.text:0000000000404C6F                 mov     rdi, rax        ; void *
.text:0000000000404C72                 call    __cxa_begin_catch
.text:0000000000404C77                 mov     eax, cs:key
.text:0000000000404C7D                 xor     eax, 0Ah
.text:0000000000404C80                 mov     cs:key, eax
.text:0000000000404C86                 call    __cxa_end_catch
.text:0000000000404C8B                 jmp     loc_404BE6
```

而因为 C++ 的全局变量默认是0, 所以 main 里的异常处理一定会触发, 导致 key 被修改.

使用正确的 key 来解密就能得到 flag 了

> moectf{f0r3v3r_JuMp_1n_7h3_a$m_a9b35c3c}

2048_master_re

非常常规的 xxtea 解密. 难点在于找到检验的位置.

在 strings 里找到 flag.txt 并通过交叉引用找到下面的函数

```
  1  int64 sub_401C83()
  2  {
  3    size_t v1; // rax
  4    char Str[136]; // [rsp+20h] [rbp-D0h] BYREF
  5    unsigned __int64 i_1; // [rsp+A8h] [rbp-48h] BYREF
  6    char _2048master2048ma[32]; // [rsp+B0h] [rbp-40h] BYREF
  7    void *Block; // [rsp+D0h] [rbp-20h]
  8    FILE *Stream; // [rsp+D8h] [rbp-18h]
  9    unsigned __int64 i; // [rsp+E0h] [rbp-10h]
 10    unsigned int v8; // [rsp+ECh] [rbp-4h]
 11
 12    Stream = fopen("flag.txt", "r");
 13    sub_428CB0(Stream, "%100s", Str);
 14    fclose(Stream);
 15    if ( strlen(Str) != 37 )
 16      return 1LL;
 17    strcpy(_2048master2048ma, "2048master2048ma");
 18    v1 = strlen(Str);
 19    Block = (void *)sub_401A81(Str, v1, _2048master2048ma, &i_1);
 20    if ( Block )
 21    {
 22      v8 = 0;
 23      for ( i = 0LL; i < i_1; ++i )
 24      {
 25        if ( *((_BYTE *)Block + i) != byte_495260[i] )
 26          v8 = 1;
 27      }
 28      free(Block);
 29      return v8;
 30    }
 31    else
 32    {
 33      sub_428D00("Encryption failed\n");
 34      return 1LL;
 35    }
 36  }
```

发现有一个字符串 `2048master2048ma`, 然后是加密和比较, 先看加密函数, 进入可以发现是典型的 xxtea 加密

```
if ( a2 <= 1 )
{
  if ( a2 < -1 )
  {
    v20 = -a2;
    v10 = 52 / -a2 + 6;
    v14 = 1050114489 * v10;
    v19 = *a1;
    do
    {
      v8 = (v14 >> 2) & 3;
      for ( i = v20 - 1; i; --i )
      {
        v16 = a1[i - 1];
        v6 = &a1[i];
        *v6 -= (((4 * v19) ^ (v16 >> 5)) + ((v19 >> 3) ^ (16 * v16))) ^ ((v19 ^ v14)
                                                                       + (v16 ^ *(_DWORD *)(4LL * (v8 ^ i & 3) + a3)));
        v19 = *v6;
      }
      v17 = a1[v20 - 1];
      *a1 -= (((4 * v19) ^ (v17 >> 5)) + ((v19 >> 3) ^ (16 * v17))) ^ ((v19 ^ v14) + (v17 ^ *(_DWORD *)(4LL * v8 + a3)));
      v19 = *a1;
      v14 -= 1050114489;
      result = --v10 != 0;
    }
    while ( v10 );
  }
}
else
```

对比标准 xxtea 算法发现只有 delta 有修改, 正常解密即可, 脚本如下

```
from regadgets import *

data = bytes([0x35, 0x79, 0x77, 0xCC, 0x1B, 0x13, 0x41, 0x34, 0xF9, 0xFF,
0x9F, 0x91, 0xFF, 0x5B, 0x94, 0x78, 0x86, 0x2A, 0xAF, 0xAE, 0xD7, 0x9E, 0x31,
0x4D, 0x7A, 0xC4, 0xA5, 0x51, 0xD1, 0xD9, 0x6E, 0x44, 0x18, 0x52, 0x86, 0x1B,
0x42, 0x8A, 0xC9, 0x63])

flag = xxtea_decrypt(data, b'2048master2048ma', 0x3E9779B9)
print(flag)
```

> moectf{@_N1c3_cup_0f_XXL_te4_1n_2O48}

A cup of tea

tea加密改了一下delta

```
#include<stdio.h>
#include<stdlib.h>
#include<stdint.h>
#include<string.h>
#define DELTA 0x114514

void tea(uint32_t* flag, uint32_t* Key) {
        uint32_t sum = DELTA * 32;
        int i;
        uint32_t v1 = flag[0];
        uint32_t v2 = flag[1];
        for (i = 0; i < 32; i++) {
                v2 -= ((v1 << 4) + Key[2]) ^ (v1 + sum) ^ ((v1 >> 5) +
Key[3]);
                v1 -= ((v2 << 4) + Key[0]) ^ (v2 + sum) ^ ((v2 >> 5) +
Key[1]);
                sum -= DELTA;

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

}

int main() {
        uint32_t Key[4] = { 0x11451419, 0x19810114, 0x51419198, 0x10114514
};
        uint32_t enc[11] = {
0x78C594AB,0x22813B59,0x472A3144,0xF255108A,0x045CFB34,0x3949EA0C,0xCB760968
,0x1559C979,0xDEF9929D,0x071D1AAB,0x00000000 };
        for (int i = 0; i < 5; i++) {
                tea(&enc[i * 2], Key);
        }
        printf("%s", enc);
        return 0;
}
```

```
//moectf{h3r3_4_cuP_Of_734_fOR_yOU!!!!!!}
```

ezpy

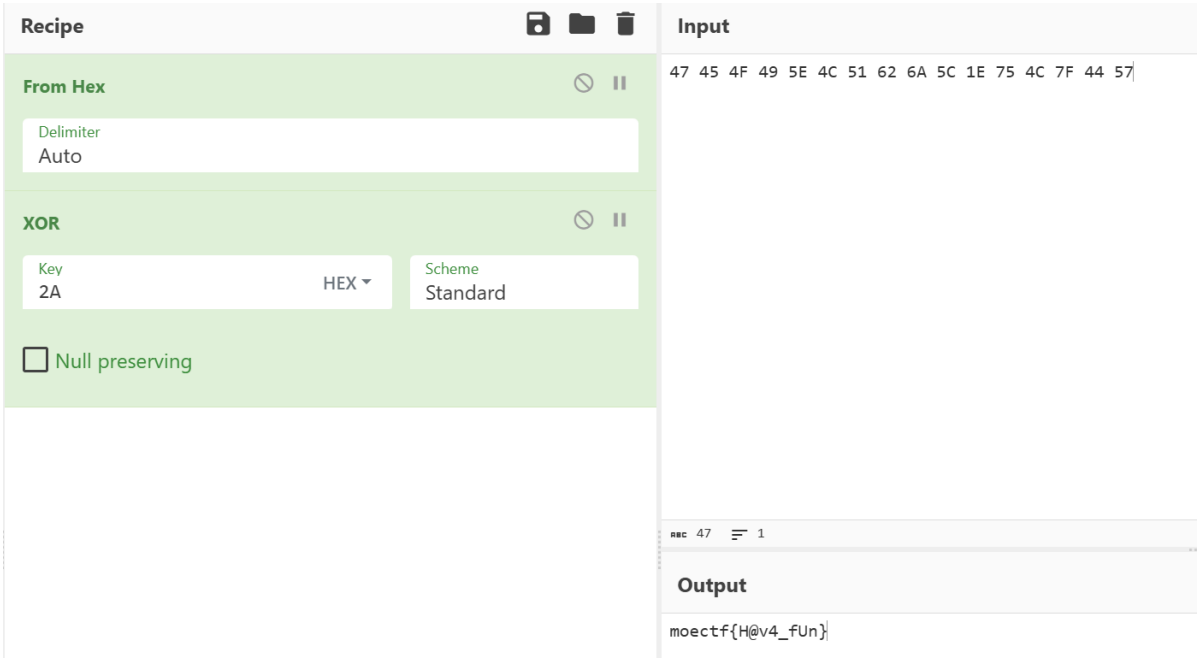
网上随便找一个`pyc`反编译网站就能得到源码，里面是一个凯撒加密
[在线Python pyc文件编译与反编译]

题解：

```python
def caesar_cipher_decrypt(text, shift):
    result = []

    for char in text:
        if char.isalpha():
            if char.islower():
                new_char = chr((ord(char) - ord('a') - shift) % 26 + ord('a'))
            elif char.isupper():
                new_char = chr((ord(char) - ord('A') - shift) % 26 + ord('A'))
            result.append(new_char)
        else:
            result.append(char)

    return ''.join(result)

# 示例
ciphertext = "wyomdp{IOe_UxOG_zim}"
shift = 114514

decrypted_text = caesar_cipher_decrypt(ciphertext, shift)
print(f"解密后的文本: {decrypted_text}")
#moectf{YOu_KnOW_pyc}
```

have_fun

窗口程序，分析程序可以知道对输入进行了简单的异或



mazegame

BFS，广度优先搜索

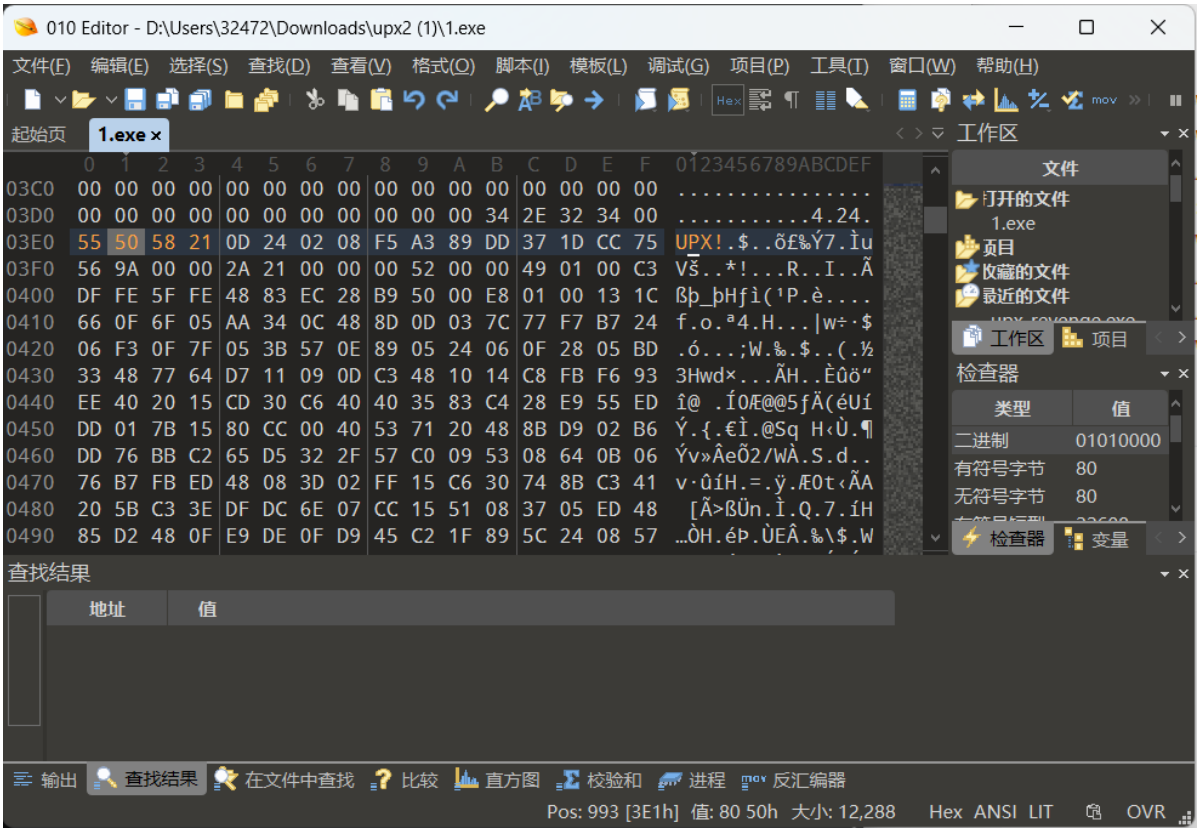
```
# 迷宫数据
binaryMap = [
    "11111111111111111111111111111111111111111111111111111111",
    "10100000000000000100000110111010111111111010111100000111",
    "10111010111111111010111011000001000001000001000101110111",
    "10000010000010000010001011011111111011101110111011110111",
    "10111111111011101110111011010000000000010100010001110111",
    "10100000001000101000100011010101111111011101110101110111",
    "10101011111110111011101011010101000001000000010101110111",
    "10101010000010100000101011110101110101111101111111110111",
    "10111010111010101111101011100101000100000101000101110111",
    "10000010001010001000001011001111011111010101011101110111",
    "11111011101011111011111111101000100000101100101001110111",
    "10001010001000100010000010001010011000100010010011000001",
    "10111010111110101010111011011001011111010101011101011101",
    "10001010001000000101000101100010100010000010100010101101",
    "11101011101111111011101011110101110111111110111010101101",
    "10001000101000001010001011000100010100000101000101011101",
    "10111111101011101110111011011111110101110111011101011101",
    "10001000001000100000001011000100000100010000000101011001",
    "11101011111011111111101011110101111101111111110101011011",
    "10101000000010001000101011010100000001000100010101011011",
    "10101111111110101010101011010111111111010101010101011011",
    "10100000000000100010101011010000000000010001010101011011",
    "10111111111111111111011101101111111111111111111011101011",
    "10000000001111000000000011110111010000111100011111011011",
    "11101111100000011011011111111010110111011101100001011011",
    "11101111111111111101101111111110111011110110110000101101l",
    "10001000111111000010000011111010110111011101100001011011",
    "10111010111111111010111011101110111010000111101100001010011",
    "10000010000010000010001011111111111111111101100001010111",
    "10111111111011101110111011101110001000110001101100001010001",
```

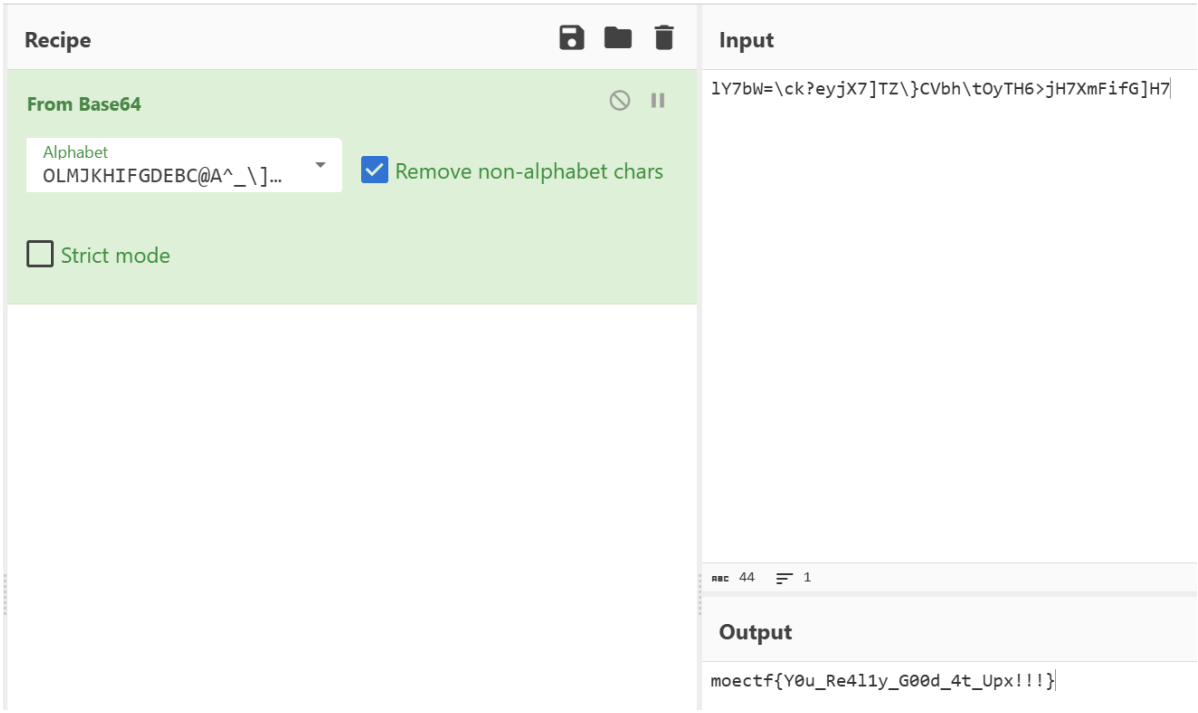
```
    "10100000001000101000100011110111011101111101100001011101",
    "10101011111110111011101011110001000101111101100001011101",
    "10101010000010100000101011111010111011111101100001011101",
    "10111010111010101111101011110001000110001101100001011101",
    "10000010001010101000001011111111111111111101100001011101",
    "11111011101011111011111100000000000000001101100001011101",
    "10001010001000100010000011111111111111111001100110111101",
    "10111010111110101010111010010000000011111110001111011101",
    "10001010001000001010001010110111000001111110100101011101",
    "11101011101111111011010001100110011111111001101110111101",
    "10001000101000001010001011111111111111111111110111010001",
    "10111111101011101110111010100001001100000000000011011011",
    "10001000001000100000001011111111111101011101111001011011",
    "10101011111011111111101011000000000001000100010111011011",
    "10101000000010001000101010010111111111111111111111011011",
    "10101111111110101010101010110111111111111111111101011011",
    "10100000000000100010101011100000000000000000000011011011",
    "10111111111111111110011011111111111111111111111011011011",
    "10000011111111111111000010000000000000000000000000011001",
    "11111011111111111111111111111111111111111111111111111101",
    "11111011100001100110110111000000000000000000000111111101",
    "11111011101111011010000111011111111111111111110111111101",
    "11111011100001000010110110000011111111111111111000000001",
    "11111011101111011010110111101111111111111111111111111111",
    "11110000000000011000110000000000000000000000000000000011",
    "11111111111111111111111111111111111111111111111111111111"
]

# 初始化参数
ROWS, COLS = 56, 56
start, end = (1,1), (15,32)
maze = [list(row) for row in binaryMap]
visited = [[False]*COLS for _ in range(ROWS)]

# DFS求解
def solve():
    stack = [(start[0], start[1], "")]
    dirs = [(-1,0,'W'),(1,0,'S'),(0,-1,'A'),(0,1,'D')]  # 方向映射

    while stack:
        x, y, path = stack.pop()
        if (x, y) == end:
            return path
        if visited[x][y]:
            continue
        visited[x][y] = True
        # 逆序添加方向以保持探索顺序
        for dx, dy, c in reversed(dirs):
            nx, ny = x+dx, y+dy
            if 0<=nx<ROWS and 0<=ny<COLS and maze[nx][ny]=='0' and not
visited[nx][ny]:
                stack.append((nx, ny, path+c))
    return None

# 执行并输出结果
path = solve()
```

```
print(f"moectf{{{path}}}")
'''
moectf{SSDDDDWWDDSSDDDDSSDDSSSSDDWWDDWWDDWWWWDDDDSSSSAASSSSAAAASSAASSAAWWAAW
WWWAAAASSDDSSAASSDDSSSSAAAASSDDDDDDWWWWDDDDSSDDDDWWDDWWAAWWDDDDSSSSSSSSSSSSA
AASSSDDDSSSSAASSSSAAAASSAASSAAWWAAWWWWAAAASSDDSSAASSDDSSSSAAAASSDDDDDDWWWWDD
DDSSDDDDWWDDWWAAWWDDDDSSSSSSSSSSSSAAAWAWWWAASSAAWWAASSAAAAAAAAAAWWWWAASSSSSS
DDDDSSSSSSDDDDDDDDWWDDDSSDDWWWDDDSSSDDDDDWAWWDDDDDDDDDDDDDDDDDDSSDDDDDDD
DWWWWAWWWWWWWWDWWWWWWWWWWWWAAWWDWWWWWWWWWWDWWWWWWAAAASSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSAAAWWAAAAAAAAAAAAAAAAAAWWWDDDDDDDDWWDDDDDDDDDDWWWAWAAWAWWWWWWW
WWWWWWDDWWWWAASSAAWWAASSAAAAAAAAAAWWWWAAWWDDWWAAWWDWWWDWWWWDDDDDDDDDDSSDDDDS
SSSDSDSSDDSSAASSAAAAWWAAAASSSSAAAAAAWWDDDDWWWWAAWWAWAASSDSSSDD}
'''

```

upx_revenge

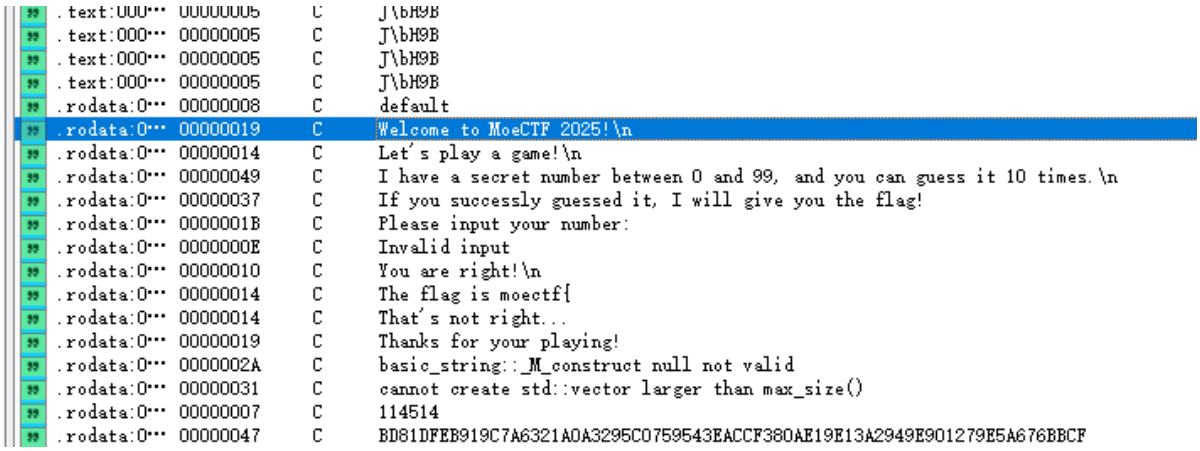
标志位中删掉了`UPX!`，补上就能正常脱壳了里面是一个换表base64

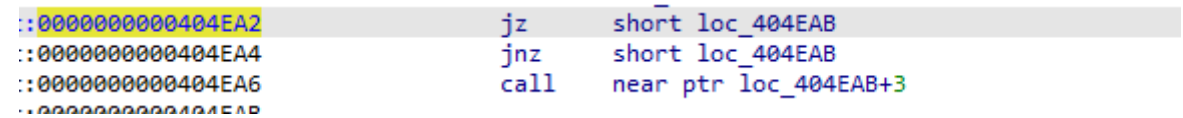




guess

首先 IDA 打开, 通过 strings 及 xrefs 找到主函数, 去掉花指令后反编译可以看到整体逻辑.





```
v18 = __readfsqword(0x28u);
sub_407F20(0LL);
sub_4513C0(&unk_5DE5D0, 0LL);
std::operator<<<std::char_traits<char>>(&qword_5DE4A0, "Welcome to MoeCTF 2025!\n");
std::operator<<<std::char_traits<char>>(&qword_5DE4A0, "Let's play a game!\n");
std::operator<<<std::char_traits<char>>(
  &qword_5DE4A0,
  "I have a secret number between 0 and 99, and you can guess it 10 times.\n");
v0 = std::operator<<<std::char_traits<char>>(&qword_5DE4A0, "If you successly guessed it, I will give you the flag!");
sub_46F480(v0, std::endl<char,std::char_traits<char>>);
sub_405422(v17);
v1 = sub_405504(v17);
sub_4058C2(v16, v1);
sub_4054E4(v17);
n0x64_1 = sub_4058EC(v16) % 0x64uLL;
for ( i = 0; i <= 9; ++i )
{
  std::operator<<<std::char_traits<char>>(&qword_5DE4A0, "Please input your number: ");
  sub_46F530(&qword_5DE4A0);
  v2 = (_QWORD *)sub_454B60(&unk_5DE5C0, &n0x64);
  if ( (unsigned __int8)sub_4512C0((char *)v2 + *(_QWORD *)(*v2 - 24LL)) || n0x64 >= 0x64 )
  {
    v4 = std::operator<<<std::char_traits<char>>(&qword_5DE4A0, "Invalid input");
    sub_46F480(v4, std::endl<char,std::char_traits<char>>);
    std::ios::clear(&unk_5DE5D0, 0LL);
    v5 = sub_40538B();
    sub_408580(&unk_5DE5C0, v5, 10LL);
  }
  else
  {
    if ( n0x64_1 == n0x64 )
    {
      std::operator<<<std::char_traits<char>>(&qword_5DE4A0, "You are right!\n");
      v6 = std::operator<<<std::char_traits<char>>(&qword_5DE4A0, "The flag is moectf{");
      sub_404CC4((__int64)v15, (__int64)&unk_5DD560);
      sub_404B6A(v17, v15, &unk_5DD540);
      v7 = sub_479790(v6, v17);
      v8 = std::operator<<<std::char_traits<char>>(v7, "}.");
      sub_46F480(v8, std::endl<char,std::char_traits<char>>);
      sub_476D60(v17);
      sub_476D60(v15);
      break;
```

是的没错又是猜数字.

一个自然的想法是套用 web 题猜数字的做法, 通过调试拿到 secret_number 的值, 输进去就好了.

下面我们发扬没苦硬吃的精神, 来分析一下程序是如何加密 flag 的.

首先看到判断正确的位置

```
if ( n0x64_1 == n0x64 )
{
  std::operator<<<std::char_traits<char>>(&qword_5DE4A0, "You are right!\n");
  v6 = std::operator<<<std::char_traits<char>>(&qword_5DE4A0, "The flag is moectf{");
  sub_404CC4((__int64)v15, (__int64)&unk_5DD560);
  sub_404B6A(v17, v15, &unk_5DD540);
  v7 = sub_479790(v6, v17);
  v8 = std::operator<<<std::char_traits<char>>(v7, "}.");
  sub_46F480(v8, std::endl<char,std::char_traits<char>>);
  sub_476D60(v17);
  sub_476D60(v15);
  break;
}
```

不难发现 sub_404B6A 就是解密函数, 进入查看主要成分.

```
int *__fastcall sub_404795(__int64 a1, __int64 a2)
{
  int *result; // rax
  int v3; // ebx
  __int64 v4; // kr00_8
  __int64 v5; // rbx
  __int64 v6; // rax
  int v7; // [rsp+10h] [rbp-20h]
  int i; // [rsp+14h] [rbp-1Ch]
  int j; // [rsp+18h] [rbp-18h]
  int v10; // [rsp+1Ch] [rbp-14h]

  result = (int *)sub_476F90(a1);
  v10 = (int)result;
  v7 = 0;
  for ( i = 0; i <= 255; ++i )
  {
    result = (int *)sub_40569A(a2, i);
    *result = i;
  }
  for ( j = 0; j <= 255; ++j )
  {
    v3 = *(_DWORD *)sub_40569A(a2, j) + v7;
    v4 = v3 + *(unsigned __int8 *)sub_477290(a1, j % v10) + 42;
    v7 = (unsigned __int8)(HIBYTE(v4) + v4) - HIBYTE(HIDWORD(v4));
    v5 = sub_40569A(a2, v7);
    v6 = sub_40569A(a2, j);
    result = (int *)sub_4056D0(v6, v5);
  }
  return result;
}
```

```
__int64 __fastcall sub_40489E(__int64 a1, __int64 a2, __int64 a3)
{
  __int64 v3; // rax
  __int64 v4; // rbx
  __int64 v5; // rax
  _DWORD *v6; // rax
  char v9; // [rsp+23h] [rbp-3Dh]
  int v10; // [rsp+24h] [rbp-3Ch]
  int v11; // [rsp+28h] [rbp-38h]
  int v12; // [rsp+2Ch] [rbp-34h]
  __int64 v13; // [rsp+30h] [rbp-30h] BYREF
  _QWORD v14[2]; // [rsp+38h] [rbp-28h] BYREF
  unsigned __int64 v15; // [rsp+48h] [rbp-18h]

  v15 = __readfsqword(0x28u);
  v10 = 0;
  v11 = 0;
  sub_476C50(a1);
  v3 = sub_476F90(a2);
  std::string::reserve(a1, v3);
  v14[1] = a2;
  v13 = sub_476EA0(a2);
  v14[0] = sub_476EC0(a2);
  while ( (unsigned __int8)__gnu_cxx::operator!=<char *,std::string>(&v13, v14) )
  {
    v9 = *(_BYTE *)sub_4057A6(&v13);
    v10 = (v10 + 1) % 256;
    v11 = (*(_DWORD *)sub_40569A(a3, v10) + v11) % 256;
    v4 = sub_40569A(a3, v11);
    v5 = sub_40569A(a3, v10);
    sub_4056D0(v5, v4);
    LODWORD(v4) = *(_DWORD *)sub_40569A(a3, v10);
    v6 = (_DWORD *)sub_40569A(a3, v11);
    v12 = *(_DWORD *)sub_40569A(a3, ((int)v4 + *v6) % 256);
    std::string::operator+=(a1, (unsigned int)(char)(v9 ^ v12));
    sub_405782(&v13);
  }
  if ( v15 != __readfsqword(0x28u) )
    _fortify_fail_();
  return a1;
}
```

可以发现用 `114514` 构造了一个长度 256 的Sbox, 然后返回跟输入字符串一个个异或的结果.

通过搜索或者询问 AI 都能发现这是 RC4 解密, 要注意的就是这里的 +42 与标准有所不同. 之后就是根据 key 和 enc 反解 flag 了.

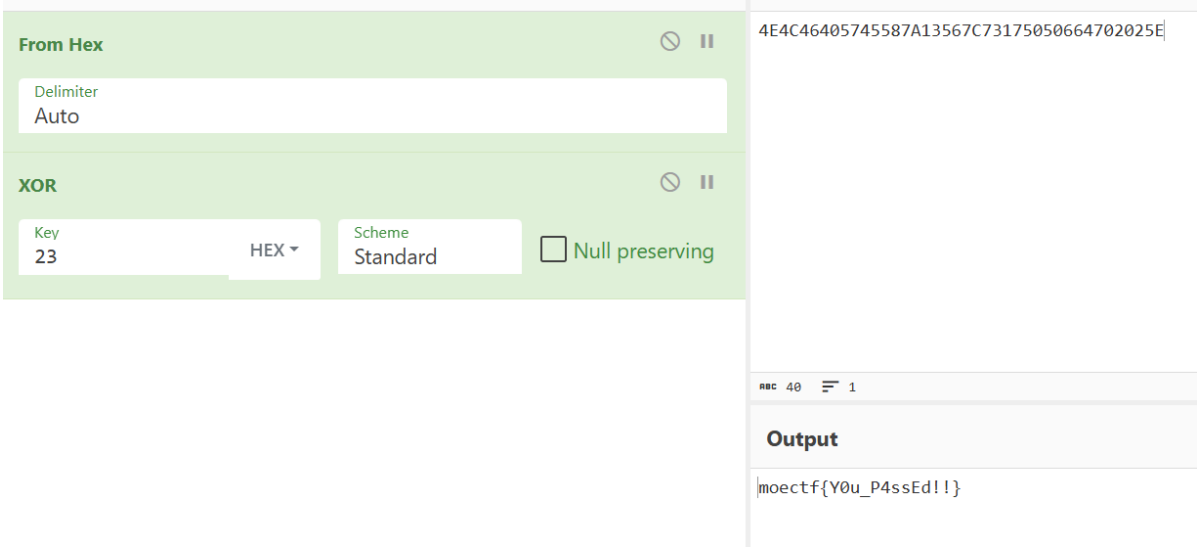
```
for ( j = 0; j <= 255; ++j )
{
  v3 = *(_DWORD *)sub_40569A(a2, j) + v7;
  v4 = v3 + *(unsigned __int8 *)sub_477290(a1, j % v10) + 42;
  v7 = (unsigned __int8)(HIBYTE(v4) + v4) - HIBYTE(HIDWORD(v4));
  v5 = sub_40569A(a2, v7);
  v6 = sub_40569A(a2, j);
  result = (int *)sub_4056D0(v6, v5);
}
```

```
.rodata:0··· 00000007     C     114514
.rodata:0··· 00000047     C     BD81DFEB919C7A6321A0A3295C0759543EACCF380AE19E13A2949E901279E5A676BBCF
```

> moectf{RrRRccCc44$$*w1th_fl0w3r!!*3c6a11b5}

A simple program

`TlsCallback` 中修改了 `strncmp` 的函数逻辑，变成了一个简单异或



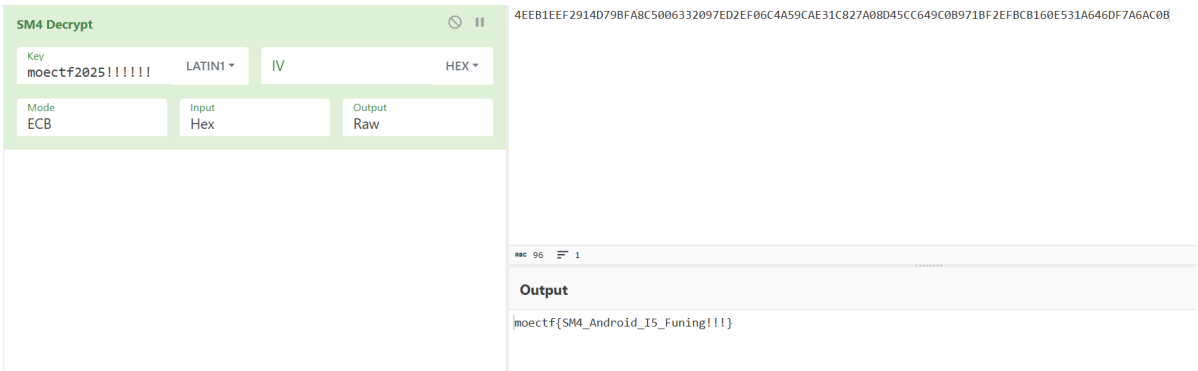
Two cups of tea

xtea生成密钥`moectf!!`，存到数组 k 里，然后与 k[2] = 0x12345678， k[3] = 0x9ABCDEF0 组成完整密钥传入xxtea进行加密

```python
from regadgets import *
enc_bytes = bytes([0xAF, 0x1C, 0x94, 0xB8, 0xA7, 0xC6, 0xFC, 0xD0])
k1 = [2, 0, 2, 5]
buff = [1566723124, 2250899117, 2635151259, 4241830417, 1175413710,
        3313593960, 4266852525, 167777774, 2550586299, 4014614088]

decrypted_enc_bytes = xtea_decrypt(enc_bytes, k1, rounds=32)

k = byte2dword(decrypted_enc_bytes[:8]) + [0x12345678, 0x9ABCDEF0]

buff_bytes = dword2byte(buff)
flag_bytes = xxtea_decrypt(buff_bytes, k, shift_func=xxtea_std_shift)

print(f"flag: {flag_bytes.decode('utf-8')}")
#flag: moectf{X7e4_And_xx7EA_I5_BeautifuL!!!!!}

```

ezandroid.pro

拖进**jadx**中，可以看到主要加密逻辑在**naitve**层，解压得到`libezandroidpro.so`文件，拖到ida中，可以知道是一个sm4加密



rusty_sudoku

rust逆向.

直接运行可以发现是一个数独游戏.

IDA 打开分析, 不要害怕这一大段, 通过分析可以知道这是在判断有没有修改隐藏的初始棋盘, 并且容易发现这就是标准数独.

```
n1114112_2 = (unsigned __int8)a68718379181594[n81_1];
if ( (n1114112_2 & 0x80u) != 0LL )
{
  v40 = a68718379181594[n81_1 + 1] & 0x3F;
  if ( (unsigned __int8)n1114112_2 <= 0xDFu )
  {
    n81_1 += 2LL;
    n1114112_2 = v40 | ((unsigned __int8)(n1114112_2 & 0x1F) << 6);
    if ( n1114112_1 == 1114112 )
      break;
  }
  else
  {
    v41 = ((a68718379181594[n81_1 + 1] & 0x3F) << 6) | a68718379181594[n81_1 + 2] & 0x3F;
    if ( (unsigned __int8)n1114112_2 < 0xF0u )
    {
      n81_1 += 3LL;
      n1114112_2 = ((n1114112_2 & 0x1F) << 12) | (unsigned int)v41;
      if ( n1114112_1 == 1114112 )
        break;
    }
    else
    {
      n1114112_2 = ((n1114112_2 & 7) << 18) | (((v40 << 6) | a68718379181594[n81_1 + 2] & 0x3F) << 6) | a68718379181594[n81_1 + 3] & 0x3Fu;
      if ( (_DWORD)n1114112_2 == 1114112 )
        break;
      n81_1 += 4LL;
      if ( n1114112_1 == 1114112 )
        break;
    }
  }
}
else
{
  ++n81_1;
  if ( n1114112_1 == 1114112 )
    break;
}
if ( (_DWORD)n1114112_2 != 46 && n1114112_1 != (_DWORD)n1114112_2 )
{
  *(_QWORD *)&v52[0] = aYouShouldNotCh;
  *(( QWORD *)&v52[0] + 1) = 32LL;
```

```
a68718379181594 db '.6..8..7.18.3......7.9....1...8...15.9..4.2..54...2..9.....3948..'
                                        ; DATA XREF: rusty_sudoku::main+484↑o
                db '...5..7..3....5.'
aYouShouldNotCh db 'You should not change the board!'
                                        ; DATA XREF: rusty_sudoku::main+633↑o
aWelcomeToMoect db 'Welcome to MoeCTF 2025!',0Ah
                                        ; DATA XREF: .rdata:off_1400C2438↓o
                db 'Please **find** my sudoku and fill it correctly.',0Ah
                db 'And then I will give you the flag.',0Ah
                db 'Input your answer in one line (without spaces).',0Ah
                db 'for example, 854219763397865421261473985785126394649538172132947856926384517513792648478651239 represents:',0Ah
                db '854|219|763',0Ah
                db '397|865|421',0Ah
                db '261|473|985',0Ah
                db '-----------',0Ah
                db '785|126|394',0Ah
                db '649|538|172',0Ah
                db '132|947|856',0Ah
                db '-----------',0Ah
                db '926|384|517',0Ah
                db '513|792|648',0Ah
                db '478|651|239',0Ah
                db 0Ah
                db 'Your answer:',0Ah
                db    0
                align 8
```

于是可以发现这个字符串对应的就是初始的 board. 找一个数独求解器就能得到结果, 输入即可得到 flag.

作为 300 分题似乎简单了, 不过由于读 rust 的反汇编本身就不是一件轻松的事，所以没有加入什么怪东西, 怕让新生不敢做题. 也可以如下用 rust 求解.

```
```

```rust
use sudoku::Sudoku;

fn solve() {
    const BOARD: &str =
".6..8..7.18.3......7.9....1...8...15.9..4.2..54...2..9.....3948.....5..7..3
....5.";

    let sudoku = Sudoku::from_str_line(BOARD).unwrap();
    if let Some(solution) = sudoku.solution() {
        println!("moectf{{{:x}}}", md5::compute(format!("{}", solution)));
    }
}

fn main() {
    solve();
}
// moectf{a8c79927d4e830c3fe52e79f410216a0}
```