因为没主见所以没名字队题解

Tetris (checkin)

查看原码发现，js文件路径

转到js文件，搜索flag

```
            // 濒唑悃忙  悅蹿剔猖涓  怫鋙惺况躇吪慍瑕义和鋤忽怖鼬堍濒偶堍脾蹿剴  瑕
            fillAfterMove();
        }
    }

    // 灏嗗劍鐓硅  缃  埌褰撳墠鍏冪礁彝

    window.focus();


    //flag{e9677fb4-dd0e-45cb-8e0e-7547171d0a85}

    // 涓虹獙鏖ｇ殑鎸復獹浜嫻欢缁戝晶浜嫻欢鏈戝恙鍣  紝浠巏eyCode鍝逛负code
    // https://developer.mozilla.org/zh-CN/docs/Web/API/KeyboardEvent/keyCode
    window.onkeydown = function (event) {
        switch (event.code) {
            // 鎸变笅浜嗕€滀悍涓嬬嚎鈧濈  澶
            case "ArrowDown":
                if (isPlaying) {
                    moveDown();
                }
```

miaES

代码如下

```
from Crypto.Cipher import AES
from os import urandom



def xor_key_list(arr, K):
    for i in arr:
        if isinstance(i, list):
            xor_key_list(i, K)
        else:
```

```
            i ^= K


def bytes2matrix(text):
    return [list(text[i:i + 4]) for i in range(0, len(text), 4)]


def matrix2bytes(matrix):
    return bytes(sum(matrix, []))


def add_round_key(s, k):
    N = len(s)
    assert N == len(k)
    matrix = [[s[i][j] ^ k[i][j] for j in range(N)] for i in range(N)]
    xor_key_list(matrix, 0xaa)
    return matrix


s_box = (
    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,
```

```
)


def sub_bytes(s, sbox=s_box):
    return [[sbox[e] for e in r] for r in s]


def shift_rows(s):
    s[0][1], s[1][1], s[2][1], s[3][1] = s[3][1], s[0][1], s[1][1], s[2][1]
    s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
    s[0][3], s[1][3], s[2][3], s[3][3] = s[1][3], s[2][3], s[3][3], s[0][3]


xtime = lambda a: (((a << 1) ^ 0x1B) & 0xFF) if (a & 0x80) else (a << 1)


def mix_single_column(a):
    t = a[0] ^ a[1] ^ a[2] ^ a[3]
    u = a[0]
    a[0] ^= t ^ xtime(a[0] ^ a[1])
    a[1] ^= t ^ xtime(a[1] ^ a[2])
    a[2] ^= t ^ xtime(a[2] ^ a[3])
    a[3] ^= t ^ xtime(a[3] ^ u)


def mix_columns(s):
    for i in range(4):
        mix_single_column(s[i])


N_ROUNDS = 10


def expand_key(master_key):
    r_con = (
        0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
        0x80, 0x1B, 0x36, 0x6C, 0xD8, 0xAB, 0x4D, 0x9A,
        0x2F, 0x5E, 0xBC, 0x63, 0xC6, 0x97, 0x35, 0x6A,
        0xD4, 0xB3, 0x7D, 0xFA, 0xEF, 0xC5, 0x91, 0x39,
    )

    key_columns = bytes2matrix(master_key)
    iteration_size = len(master_key) // 4

    i = 1
    while len(key_columns) < (N_ROUNDS + 1) * 4:
        word = list(key_columns[-1])

        if len(key_columns) % iteration_size == 0:
            word.append(word.pop(0))
            word = [s_box[b] for b in word]
            word[0] ^= r_con[i]
```

```
            i += 1
        elif len(master_key) == 32 and len(key_columns) % iteration_size == 4:
            word = [s_box[b] for b in word]

        word = [i ^ j for i, j in zip(word, key_columns[-iteration_size])]
        key_columns.append(word)

    key_matrix = [key_columns[4 * i: 4 * (i + 1)] for i in range(len(key_columns) //
4)]
    xor_key_list(key_matrix, 0xcf)
    return key_matrix


def encrypt(k, m):
    round_keys = expand_key(k)
    m = bytes2matrix(m)

    m = add_round_key(m, round_keys[0])

    for i in range(1, N_ROUNDS):
        m = sub_bytes(m, s_box)
        shift_rows(m)
        mix_columns(m)
        m = add_round_key(m, round_keys[i])

    m = sub_bytes(m, s_box)
    shift_rows(m)
    m = add_round_key(m, round_keys[-1])

    ciphertext = matrix2bytes(m)
    return ciphertext


def encrypt_flag(iv, plaintext):
    s = iv
    ciphertext = b''
    pad_len = 16 - (len(plaintext) % 16)
    plaintext += bytes([0]) * (pad_len - 1) + bytes([pad_len])
    for i in range(0, len(plaintext), 16):
        stream = encrypt(key, s)
        xor = lambda x, y: bytes([a ^ b for a, b in zip(x, y)])
        ciphertext += xor(plaintext[i:i + 16], stream)
        s = stream
    return ciphertext



def decrypt_flag(iv, ciphertext):
    s = iv
    plaintext = b''
    for i in range(0, len(ciphertext), 16):
        stream = encrypt(key, s)
```

```
        xor = lambda x, y: bytes([a ^ b for a, b in zip(x, y)])
        plaintext += xor(ciphertext[i:i + 16], stream)
        s = stream
    return plaintext

ciphertext = b'\xc6*\xe0^\xeb\xd7}\xcf\x99?
IT;j{\xf6\x08\xc3J\xad\x0b6\xaa\x82\xb5[]X>m\xcf\xbc7v\xc1(s\xa2>\xf15\xa5\x91kg\xa4
IT\xa4I\xc5*7B\x8f\x8clw\xa3\xc8\xb2\xe9\x1a\x1b\x1d\x89\xab\xee\x83\xa4\xce\x9fY\xf
f\xa3IR\xb0\xbb1'
iv = b"wL\xc58C\x9d3\x7f\xa85\x19\x89\x9b\x8d`'"
key = b'9\x83\x13i\xdbA\xf0\x88\xa9b$^\x7f\x1b\xd0\xb8'

decrypted_flag = decrypt_flag(iv, ciphertext)
print(f'Decrypted Flag: {decrypted_flag}')
```

```
问题    输出    调试控制台    终端

type: 01, off: 1D, byte: 00 65 65 e
type: 01, off: 1E, byte: 00 65 65 e
type: 01, off: 1F, byte: 00 35 35 5
type: 01, off: 20, byte: 00 35 35 5
type: 01, off: 21, byte: 00 30 30 0
type: 01, off: 22, byte: 00 63 63 c
type: 01, off: 23, byte: 00 30 30 0
type: 01, off: 24, byte: 00 39 39 9
type: 01, off: 25, byte: 00 63 63 c
type: 01, off: 26, byte: 00 37 37 7
type: 0A, off: 27, leshort: 00 30 30 48
type: 01, off: 29, byte: 00 7D 7D }
e124b18-dc26-4c42-96ef-ee550c09c7}
[48]
48 e
Traceback (most recent call last):
  File "/home/bddk/ctf/ctf_2023/magic.py", line 40, in <module>
    real_flag[indexes[i]] = table[i]
IndexError: list assignment index out of range
(ctf) bddk@LAPTOP-TJ96KO8G:~/ctf/ctf_2023$ /home/bddk/anaconda3/envs/ctf/bin/python /home/bddk/ctf/ctf_2023/miaes/1.py
Decrypted Flag: b'NOCTF{0h_mi4_Nev3r_7h0ugHt_h3r_Ae3_w1Ll_8e_DeCryPt3d_c@ngr3tu1at1on3}\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x0b'
(ctf) bddk@LAPTOP-TJ96KO8G:~/ctf/ctf_2023$
```

magic

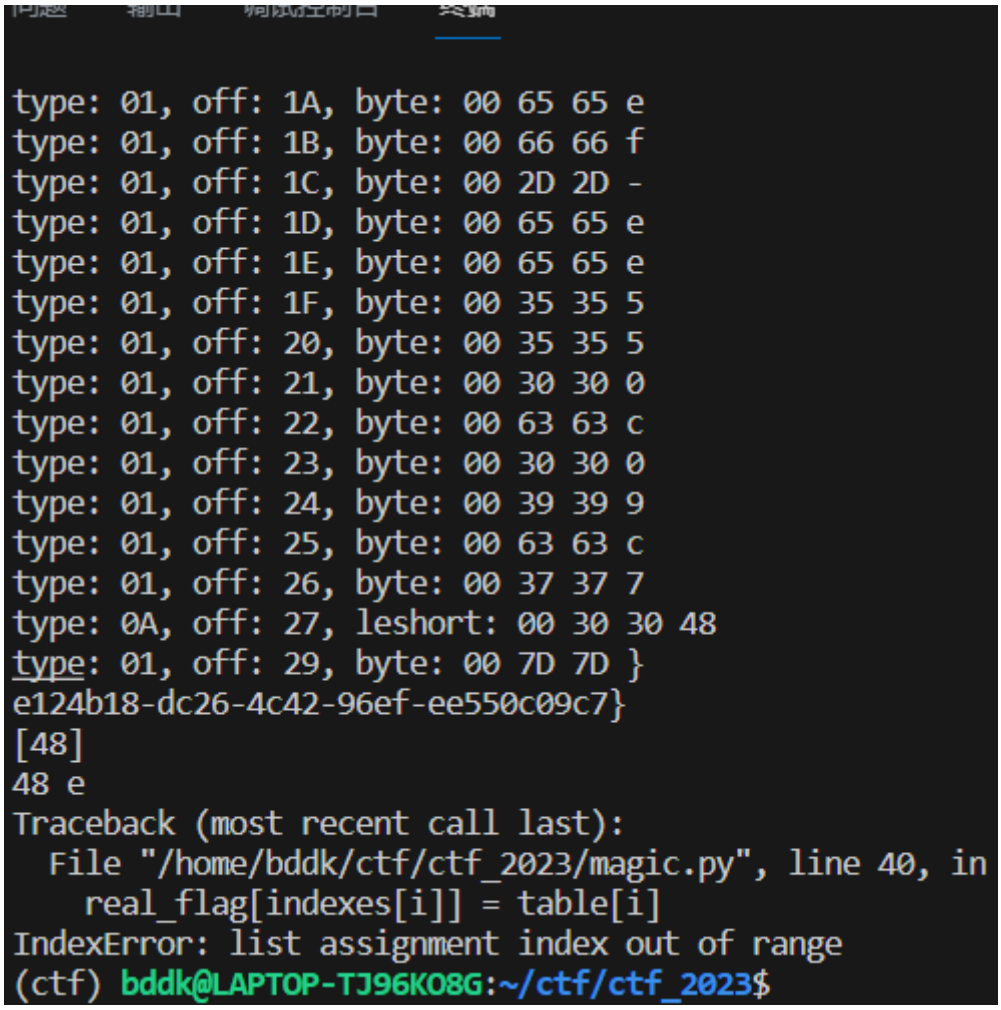
代码如下

```
import hexdump
import struct
import string

f = open('./magic.mgc', 'rb')
b = f.read(0x178)
indexes = []
table = ''
while True:
    b = f.read(0x178)
    if len(b) != 0x178:
        break
    line = b[:0x30]
    _type = line[6]
    off = struct.unpack_from('<I', line, 0x0c)[0]
    s = f'type: {_type:02X}, off: {off:02X}'
```

```
    if _type == 5:
        s += ', str: '+line[0x20:].decode()
    elif _type == 1:
        n1, n2 = line[0x18], line[0x20]
        v = n1 ^ n2
        s += f', byte: {n1:02X} {n2:02X} {v:02X} {chr(v)}'
        table += chr(v)
    elif _type == 10:
        n1, n2 = line[0x18], line[0x20]
        v = n1 ^ n2
        s += f', leshort: {n1:02X} {n2:02X} {v:02X} {v}'
        indexes.append(v)
    print(s)
    # hexdump.hexdump(line)
    # print('')

# f_o_a__lhy_s_y^^hete_ug___goo_t_
print(table)
# [25, 8, 18, 20, 27, 31, 30, 26, 7, 13, 5, 11, 16, 9, 34, 32, 22, 10, 19, 23, 24,
15, 28, 36, 12, 33, 17, 6, 14, 35, 21, 29]
print(indexes)
real_flag = [' ']*40
for i in range(len(indexes)):
    print(indexes[i], table[i])
    real_flag[indexes[i]] = table[i]
print(''.join(real_flag))
# _oh_yes_you_got_the_flag___^_^__
```



signin

由于2的10次方是1024，给了十次输入机会，只需要通过二分法进行判断。得到答案为927（丧失了当时的截图）