It's Mygo!!!!-writeup

Misc

signin

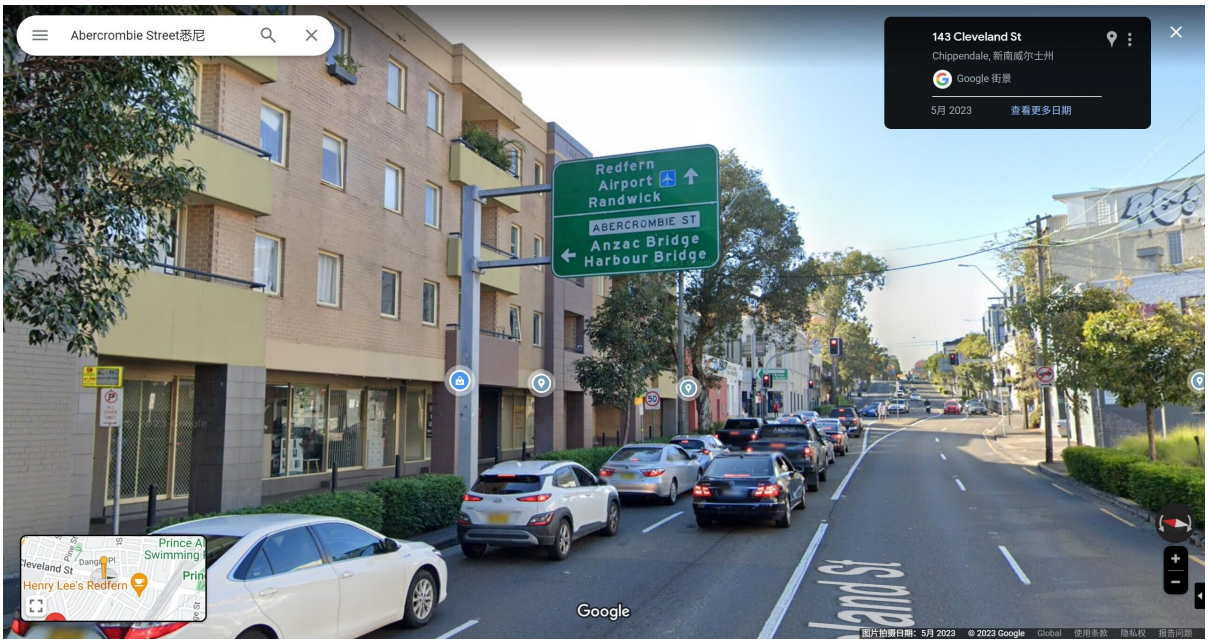
nc 之后猜中数字就拿到了flag

magic

将“127 string hello helloworld”写入magic，执行命令file -m magic -C，查看二进制，在0x017c位置得到0x3d，在0x0198位置得到hello\0，大胆猜测这是.mgc文件的规律，查看题目所给的magic.mgc，按照以上规律可得flag。

lunar

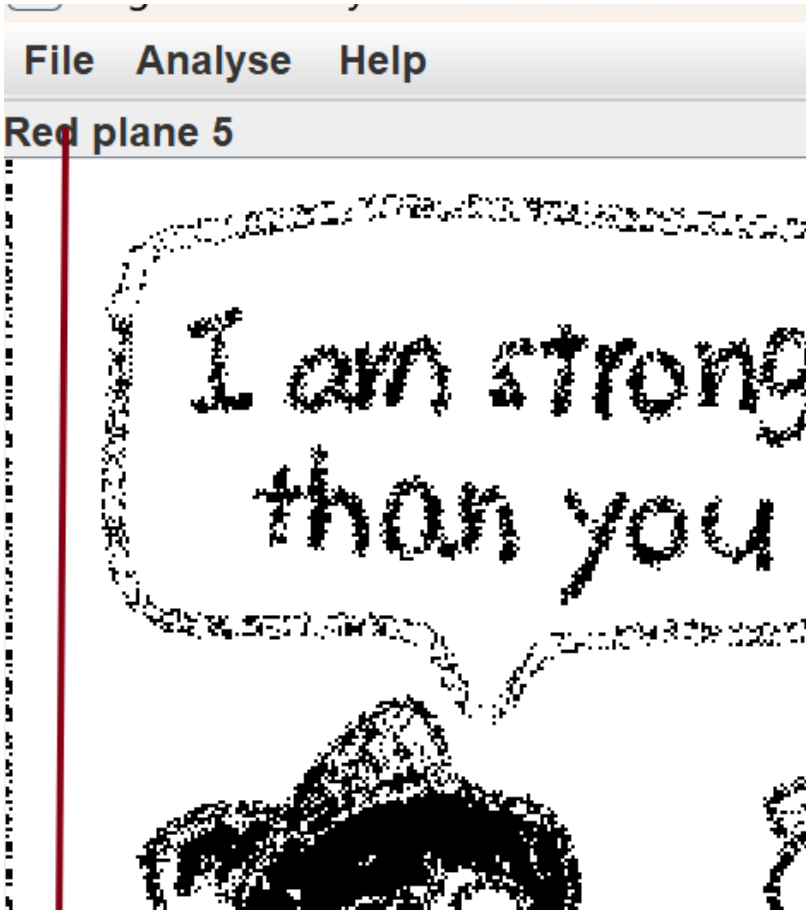
截取路牌部分，使用谷歌识图，根据背景颜色和字体可以判断这是澳大利亚的路牌，获取澳大利亚有名的机场和桥梁，结合图片，可知悉尼海港大桥（Sydney Harbour Bridge）与悉尼机场（Sydney Airport）符合要求，又在附近找到了Anzac Bridge，可知照片所在地为悉尼市，然后直接在谷歌地图搜索路牌上的信息，找到 Abercrombie st，然后就是通过谷歌街景顺着街道慢慢找路牌的过程



514和⑨

分离出图片后，根据Hint通过stegsolve分析，联系题目名字，发现第一张图片的 5 1 4 位有问题，然后通过data extract提取，需要注意的是从列提取，第二张图片同样的操作。从两张图片得到`the key is koishie`,`pziynmrccsqvvsliaajgwwozljqvxc`，通过维吉尼亚密码解密得到

flag
`



Crypto

miaES

解密脚本如下

在已知key和iv的情况下，直接使用CBC解密即可

PYTHON

```python
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 inv_sub_bytes(s, s_box_inv):
    return [[s_box_inv[e] for e in r] for r in s]


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


def inv_mix_columns(s):
    for i in range(4):
        u = xtime(xtime(s[i][0] ^ s[i][2]))
        v = xtime(xtime(s[i][1] ^ s[i][3]))
        s[i][0] ^= u
        s[i][1] ^= v
        s[i][2] ^= u
        s[i][3] ^= v


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)])
        decrypted_block = xor(ciphertext[i:i + 16], stream)
        plaintext += decrypted_block
        s = stream
    pad_len = plaintext[-1]
    plaintext = plaintext[:-pad_len]
```

```
    return plaintext


ciphertext = '\xc6*\xe0^\xeb\xd7}\xcf\x99?
IT;j{\xf6\x08\xc3J\xad\x0b6\xaa\x82\xb5[]X>m\xcf\xbc7V\xc1(s\xa2>\xf1
5\xa5\x91kg\xa4IT\xa4I\xc5*7B\x8f\x8clw\xa3\xc8\xb2\xe9\x1a\x1b\x1d\x
89\xab\xee\x83\xa4\xce\x9fY\xff\xa3IR\xb0\xbb1'
iv = 0x774cc3853843c29d337fc2a83519c289c29bc28d
key = 0x39c2831369c39b41c3b0c288c2a962245e7f1bc390c2b8
decrypted_text = decrypt_flag(iv, ciphertext)
print(f'decrypted_text = {decrypted_text}')
```

imitate

对于jwt可以对数字签名进行验证，满足

PYTHON

```
pow(signature,e,n) = to_sig
```

对于两个数字签名，**进行移项后求gcd，即可得到n**

PYTHON

```
t1 = pow(sign1,e
t2 = pow(sign2,e)
print(math.gcd(t1-s1,t2-s2))
```

解密脚本如下

已知n，**可以根据其中一个签名和gift进行暴力枚举得到d**，因为k的位数不会特别的，实际计算k应该在几千的大小

PYTHON

```
for k in range(1,10000):
    d = (k * n + gift) // (e * n) # 可以直接对d取整方便计算
    if pow(s1,d,n) == sign1:
        print(d)
```

已知e,d可d是e对于phi(n)的逆元 可以枚举从而可对n进行分解
分解代码如下

PYTHON

```python
import random
from math import gcd

def attack(e, d, n):
    result = [0, 0]
    k = d * e - 1
    random.seed()
    while True:
        g = random.randint(2, n - 1)
        k1 = k
        while k1 % 2 == 0:
            k1 //= 2
            x = pow(g, k1, n)
            result[0] = gcd(x - 1, n)
            if x > 1 and result[0] > 1:
                result[1] = n // result[0]
                return result
e = 65537
d =
24867392909478407453962340644459746686954800315880573307459565031751832940304617559529319449657860630260444420377868570652451551379163077336704674676463433 7
n =
62933824880618102769166277371638802078427430811780395924118686804368237388352884187970021624089426289458713635879617312722371243939224025234274543753658 93
p, q = attack(e, d, n)

print("分解得到的p和q: ", p, q)
```

已知p，通过逆元运算可求解得到flag

PYTHON

```
import math
s1 = 332347826739
c =
345493210845357991613139943050251158823061954531888143099134990103890
261160545373408014905823087871443390906704942924439745730330190324604
607536938266367079808006301848264470108499531968168253360898869342451
1755031
Mod =
139783530492499989366806186190970201707045784617955510994670668264365
125613780534146207604662234972781205415792315794694804125806787962550
504358642185835486015743979918457538944006966377376366330483194448343
751231567467051085883090503081476738133080325775972135216360160821424
559840800702984626714613403410341
n =
629338248806181027691662773716388020784274308117803959241186868043682
373883528841879700216240894262894587136358796173127223712439392240252
3427454375365893
sign1 =
0x4b97b0b52cf4e9d12ee1d68da1914ee50ce18dc1eb4fdd19afb4230f79a6fb9c405
27dc58d4921fb9eb7f75df6f0adb61e5ad51c82083f93b3efed5ce36adc44
sign2 =
0x1bc5284234e6281ff8d68be0e3f1c7fd493f40003359224f6709efe342368c4d79d
0f560c03dd8d58d4a69675505765ed0c5809bd25befe59f86756dc246bff
s2 = 289683370039
e = 65537
d =
248673929094784074539623406444597466869548003158805733074595650317518
329403046175595293194496578606302604444203778685706524515513791630773
3670467764634337

gift =
623854769787005012140895681851956496592749343630597442119311658347814
354148492284469787427963632277600305545664023072530320046232121963656
209747754305160219588607477451932744562460172642629992714110316010115
501972439510588249911296799276636925402860124247272604281379087062710
759962694631326501476927077774 00

mod =
139783530492499989366806186190970201707045784617955510994670668264365
```

```
125613780534146207604662234972781205415792315794694804125806787962550
504358642185835486015743979918457538944006966377376366330483194448343
751231567467051085883090503081476738133080325775972135216360160821424
559840800702984626714613403410341

p =
103680668771730609245630123974535846151128117311784292994494066264158
419199927

q =
606996710439598649301926534586674919481822603867344369789951137295051
81882659

print(pow(s2,d,n))

# t1 = pow(sign1,e)
# t2 = pow(sign2,e)
# print(math.gcd(t1-s1,t2-s2))
e = 65537  # 公钥指数

def extended_gcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = extended_gcd(b % a, a)
        return (g, x - (b // a) * y, y)
def mod_inverse(a, m):
    g, x, y = extended_gcd(a, m)
    if g != 1:
        raise Exception('Modular inverse does not exist')
    else:
        return x % m
p_inverse = mod_inverse(p, Mod)
c * p_inverse) % Mod
flag = flag_bytes_to_long.to_bytes((flag_bytes_to_long.bit_length() +
7) // 8, byteorder='big')
print("解出的flag: ", flag)
```

LATTICE

该题为格密码中的背包密码，**使用格基规约算法可对其进行破解**，最后算出的矩阵就是由flag的二进制信息位表示

解密脚本如下

PYTHON

```python
from itertools import product
from sage.matrix.constructor import Matrix
from sage.rings.integer_ring import ZZ
# 打开文件 1.txt 并读取内容
with open('task.txt', 'r') as file:
    pubKey_str = file.readline().strip()
    pubKey = list(map(int, pubKey_str.split(',')))
pubkey_length = len(pubKey)
with open('task.txt', 'r') as task_file:
    pubkey_str = task_file.readline().strip()
    pubkey = list(map(int, pubkey_str.split(',')))
with open('enc.txt', 'r') as enc_file:
    enc = int(enc_file.readline().strip())
nbit = len(pubKey)
A = Matrix(ZZ, nbit + 1, nbit + 1)
for i in range(nbit):
    A[i, i] = 1
for i in range(nbit):
    A[i, nbit] = pubKey[i]
A[nbit, nbit] = -int(enc)
res = A.LLL()
print(res[-1][:-1])

#   去除矩阵的括号和逗号后余下的就是 flag(res)信息位
100111001001111010000110101010001000110011110110100110101100101011100
100110101101101100001100110101111101001000011001101101100011010001100011
011010110000101101110010111110100000101110100011101000011010001100011
011010110111001101011111001100010111001101011111001101000101111101101
100011000010111010001110100011010010110001100110011010111110110011101
101111001100000110010001011111011100110111010000110100011100100111010
001111101
```

Pwn

It's Mygo!!!!

在IDA打开，在sys_exe()找到危险函数__isoc99_scanf()，存在栈溢出漏洞，构造`payload=b"a"*0x30 + b"cat flag\0"`，输入给v3，得到flag。

Web

Tetris (checkin)

flag就在js代码里，仔细找一找就有

ping test

用&组合命令用转义字符绕过`ca\t`在目录中找到flag即可

Re

segments

用二进制查看器打开，在`0x8d, 0xdd, 0x12d, 0x17d`位置找到flag片段，拼接可得flag。