

Web: ezmd5

打开实例，是一个 md5 强比较

The screenshot shows a web debugger interface with the following details:

- Source Code:**

```
<?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.";
}
?> Wrong! Hacker!
```
- URL:** <http://ctf.furryctf.com:36523/>
- Method:** Use POST method
- Content-Type:** application/x-www-form-urlencoded
- Body:** user=s878926199a&&pass=s155964671a

一开始以为是弱比较，所以用了常见的哈希碰撞，但是错了
再仔细看才发现是强比较，所以尝试数组绕过

```
<?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.";
}
?> Congratulations! Here is your flag:
POFP{1ac0dc3f-7a12-46b0-943c-5d1649484598}
```

The screenshot shows a web-based penetration testing or reverse engineering environment. At the top, there's a code editor window displaying the provided PHP script. Below it is a browser-like interface with various tabs and toolbars. The URL bar contains `http://ctf.furryctf.com:36523/`. The toolbar includes buttons for LOAD, SPLIT, EXECUTE, TEST, SQLI, XSS, LFI, SSRF, SSTI, and others. A dropdown menu for 'Hack' is open. In the main body area, there are sections for 'URL' (with the current URL), 'Body' (containing the POST data `user[]=1&&pass[]=2`), and 'Headers' (showing 'Content-Type: application/x-www-form-urlencoded').

Misc 签到题：

既然题目说了是在投票页后面，那就查看源码

签到题 100 pts

本题flag头: furryCTF{}

这里是今年的签到题~

nwn不整太多花里胡哨的了，今年的签到题题目是一个投票：

<https://tp.wjx.top/vm/tUv4AXj.aspx#>

a? 你说过期了?

那不关我的事nwn，我已经把flag写在投票后的页面了，怎么拿是你的问题哦~

时刻记得，flag的格式为furryCTF{}哦~

因为这个题目太简单，所以初始分数只有100分，最高衰减20% zwz

```
<div class="wall-container-content">
    <div class="headerTitle">
        <span class="title-icon1"></span>
        <span id="lblHeader">嘴？flag是什么？好吃的嘴</span>
        <span class="title-icon2"></span>
    </div>
    <div style="margin:0 auto;" id="set_outerwidth">
        <div id="divResult"><div style="margin-bottom:15px;" class="defdisplay">furryCTF{Cro5s_The_Lock_Of_Time}</div><div style="text-align:left;padding-bottom:10px;font-size:14px">
        </div>
    </div>
</div>
<script type="text/javascript" src="https://image.wjx.cn/cdn/jquery/1.10.2/jquery_min.js"></script><script type="text/javascript">window.jQuery&&document.write('<script src="/js/jquery.js?v=6885" type="text/javascript"></script>
<script>
    var isPar = 0;
</script>
<script type="text/javascript">
    function loadLocalRes() {
        var cdnDomain = "//image.wjx.cn";
        try {
            if (typeof setWallStyle == 'undefined' && cdnDomain) {
                var links = document.getElementsByTagName("link");
                var arrlink = [];
                for (var i = 0; i < links.length; i++) {
                    if (links[i].href)
                        arrlink.push(links[i].href);
                }
                for (var i = 0; i < arrlink.length; i++) {
                    if (arrlink[i].indexOf(cdnDomain) > -1) {
                        var newHref = arrlink[i].split(cdnDomain)[1];
                        var newLink = document.createElement('link');
                        newLink.setAttribute('rel', 'stylesheet');

```

即可找到

Misc CyberChef

The screenshot shows the CyberChef interface with a challenge titled "Fried Chicken". The challenge description includes a note about Misc handlers being fond of cooking, a download link for "Fried Chicken.txt", and a message stating "该题目已被解出" (The question has been solved). A "提交 flag" button is also visible.

The screenshot shows the "Fried Chicken.txt" file in Sublime Text. The content is a list of ingredients and steps for making fried chicken. The file is 1224 lines long and 39983 characters selected. The text is as follows:

```
Crazy Thursday Fried Chicken.

Ingredients.
2 g salt
34 g sage
27 g oil
37 g ginger
13 g milk
5 g butter
7 g flour
45 g paprika
32 g turmeric
29 g pepper
19 g vanilla
35 g thyme
9 g rosemary
11 g eggs
26 g cheese
40 g cinnamon
23 g honey
43 g nutmeg
31 g basil
14 g oregano
22 g tomato
16 g garlic
42 g parsley
10 g onions
8 g potatoes
1 g sugar
12 g cumin
49 g coriander
17 g chicken

Method.
Clean the mixing bowl.
Clean the 2nd mixing bowl.
Clean the 3rd mixing bowl.
Clean the 4th mixing bowl.
Clean the 5th mixing bowl.
Clean the mixing bowl.
Put honey into the mixing bowl.
Add honey to the mixing bowl.
Add milk to the mixing bowl.
Add salt to the 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.
```

拿到文本，排除文本水印隐写，零宽字符隐写等文本隐写术，从文本内容中找线索
有点晕英文，丢进谷歌翻译

Method.	制作方法：
Clean the mixing bowl.	清洗搅拌碗。
Clean the 2nd mixing bowl.	清洗第二个搅拌碗。
Clean the 3rd mixing bowl.	清洗第三个搅拌碗。
Clean the 4th mixing bowl.	清洗第四个搅拌碗。
Clean the 5th mixing bowl.	清洗第五个搅拌碗。
Clean the mixing bowl.	清洗搅拌碗。
Put honey into the mixing bowl.	将蜂蜜放入搅拌碗。
Add honey to the mixing bowl.	将蜂蜜加入搅拌碗。
Add milk to the mixing bowl.	将牛奶加入搅拌碗。
Add salt to the mixing bowl.	将盐加入搅拌碗。
Liquify contents of the mixing bowl.	将搅拌碗中的内容物液化。
Pour contents of the mixing bowl into the baking dish.	将搅拌碗中的内容物倒入烤盘。
Clean the mixing bowl.	清洗搅拌碗。
Put honey into the mixing bowl.	将蜂蜜放入搅拌碗。
Add honey to the mixing bowl.	将蜂蜜加入搅拌碗。
Add milk to the mixing bowl.	将牛奶加入搅拌碗。
Add salt to the mixing bowl.	将盐加入搅拌碗。
Clean the 2nd mixing bowl.	清洗第二个搅拌碗。
Put thyme into the 2nd mixing bowl.	将百里香放入第二个搅拌碗。
Put rosemary into the 2nd mixing bowl.	将迷迭香放入第二个搅拌碗。
Clean the 2nd mixing bowl.	清洗第二个搅拌碗。
Liquify contents of the mixing bowl.	将搅拌碗中的内容物液化。
Pour contents of the mixing bowl into the baking dish.	将搅拌碗中的内容物倒入烤盘。
Clean the mixing bowl.	清洗搅拌碗。
Put honey into the mixing bowl.	将蜂蜜放入搅拌碗。
Add honey to the mixing bowl.	将蜂蜜加入搅拌碗。
Add honey to the mixing bowl.	将蜂蜜加入搅拌碗。
Add eggs to the mixing bowl.	将鸡蛋加入搅拌碗。
Add essence to the mixing bowl.	将精油加入搅拌碗。

使用箭头按钮可查看全文。

< > 完成

看了翻译过的菜谱，第一反应是，这不像是人类真的会写的菜谱，正常人谁做饭做几步就频繁清洗搅拌碗，往搅拌碗里加了东西又立马清洗了。

仔细观察文本会发现，每一步都出现 mixing bowl，总是出现结构相同的句式，反复出现的几个动词 put, add, clean, liquify, pour

Google mixing bowl liquify

Threads https://www.threads.com/post/video-ho... · 翻译此页

How a professional chef stabilizes a mixing bowl.
2025年10月30日 — you have two hands for a reason, hold the bowl with one hand, the whisk with the other, BOOM, problem solved Worked in professional kitchens ...

Esolang Wiki https://esolangs.org/wiki/Chef · 翻译此页

Chef
2024年7月1日 — Stir the mixing bowl for 4 minutes. Liquify the contents of the mixing bowl. Pour contents of the mixing bowl into the baking dish. bake the ...

123RF https://www.123rf.com/photo_43555738... · 翻译此页

Mixing Liquified Eggs Into An Industrial Mixing Bowl At A Local ...
Description. In a bustling kitchen, a skilled chef pours a rich, creamy batter into a large mixing bowl, preparing for a delectable baking session.

在浏览器搜索关键词组合，发现 Chef，和题目 CyberChef 相关联

页面 讨论 厨师 阅读 视图源 查看历史 搜索 Esolang

厨师

Chef厨师是一种基于组合的语言,其中程序看起来像烹饪食谱。David Morgan-Mar厨师由大卫·摩根·马尔于2002年设计。

设计原则

根据主旨主页,主题的设计原则如下:

- 食谱食谱不仅会产生有效的输出,而且易于准备且美味可口。
- 食谱可能对厨师不熟悉的厨师有吸引力。
- 食谱将是衡量标准,但可以使用传统的烹饪措施,例如杯子和汤匙。

你好,世界!在厨师中

维恩·摩根·马尔编写了示例脚本《Hello World Souffle》,它确实能生成有效的输出效果,但未能达到“易于准备且美味”的设计目标。后来,迈克·沃思编写了一个Hello世界项目,可以将其作为巧克力蛋糕的实用(尽管略带神奇)食谱进行跟踪。

巧克力酱的Hello World蛋糕。

这能打印出你好的世界,同时又比Hello World Souffle更美味。主要厨师送一个“世界”蛋糕,他放进了烤盘里,当他得到厨师制作“你好”的巧克力酱,然后放入烤盘中。然后当他将酱料冷藏时,整个东西就被打印出来了。什么时候真的在做饭,我把巧克力酱供给菜园就好了。与蛋糕分开,用液表示根据具体含义融化或混合上下文。

成分:

- 33克巧克力豆
- 100克黄油
- 54毫升双乳霜
- 2个鸡蛋打粉
- 1/4茶匙
- 110克白糖
- 119克面粉
- 32克可可粉
- 0克蛋糕混合物

烹饪时间:25分钟。

将烤箱预热至180摄氏度。

可以确认就是文本内容就是 Chef 语言了解了下, ingredients 相当于数值表, put, add 在加减, pour 在输出, 只需关注主碗, 只有它会 liquify 和 pour 叫 ai 写个脚本将计算出的 ascii 还原成字符串

```
PS C:\Users\LENOVO> cd Desktop
PS C:\Users\LENOVO\Desktop> python solve.py
==QfBdVQf9UNf9kVJZ1X5VDZzJXdoR1X5dTYYN0Xu90XzdTZhWd09Fb542bs92QfVWbwM1XltWMM9FZxU3bX9VS7ZEVdIncyVnZ
```

猜测是反转的 base64

The screenshot shows a browser window with three tabs: "furryCTF 2025 高校联合新神器", "在线文本反转工具 - 字符串倒序", and "From Base64 - CyberChef". The main content area has tabs for "文本反转" (Text Reversal) and "空格转回车" (Space to Carriage Return). The "文本反转" tab is selected. It has two large buttons: "反转文本" (Reverse Text) and "镜像文本" (Mirror Text). Below these buttons, there are two sections: "完全反转: 将整个字符串完全颠倒" (Full Reverse: Reverse the entire string) and "单词颠倒: 每个单词的字母顺序颠倒" (Word Reverse: Reverse the order of letters in each word). A "处理结果" (Processing Result) section shows the input string "==QfBdVQf9UNf9kVJZ1X5VDZzJXdoR1X5dTYYN0Xu90XzdTZhWd09Fb542bs92QfVWbwM1XltWMM9FZxU3bX9VS7ZEVdIncyVnZ" and the output string "ZnVycnlDVEZ7SV9Xb3UxF9MMWtI1MwbWVfQ29sb245bF9OdWdnZTdzX09uX0NyYTd5X1RodXJzZDV5X1ZJvk9fNU9fQVdBfQ==". There is a "Copy" button next to the output.

The screenshot shows the CyberChef interface. On the left, under "Recipe", it says "From Base64" with "Alphabet" set to "A-Za-zA-9+=". There is a checked checkbox for "Remove non-alphabet chars" and an unchecked checkbox for "Strict mode". The "Input" field contains the base64-reversed string "ZnVycnlDVEZ7SV9Xb3UxF9MMWtI1MwbWVfQ29sb245bF9OdWdnZTdzX09uX0NyYTd5X1RodXJzZDV5X1ZJvk9fNU9fQVdBfQ==". On the right, under "Output", it shows the decoded string "furryCTF{I_Wou1d_L1ke_S0me_Colon91_Nugge7s_0n_Cra7y_Thursd5y_V1V0_50_AWA}".

尝 试 , 得 到
furryCTF{I_Wou1d_L1ke_S0me_Colon91_Nugge7s_0n_Cra7y_Thursd5y_V1V0_50_AWA}

MISC 赛后问卷

送分，填问卷得 flag

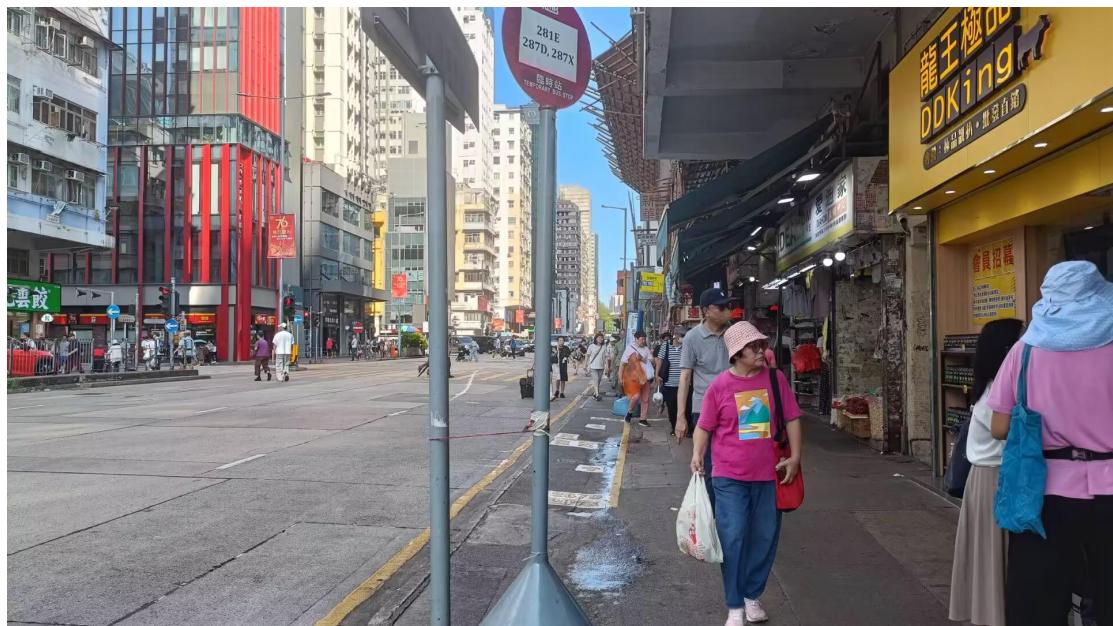
OSINT 独游

🔍 独游 350 pts

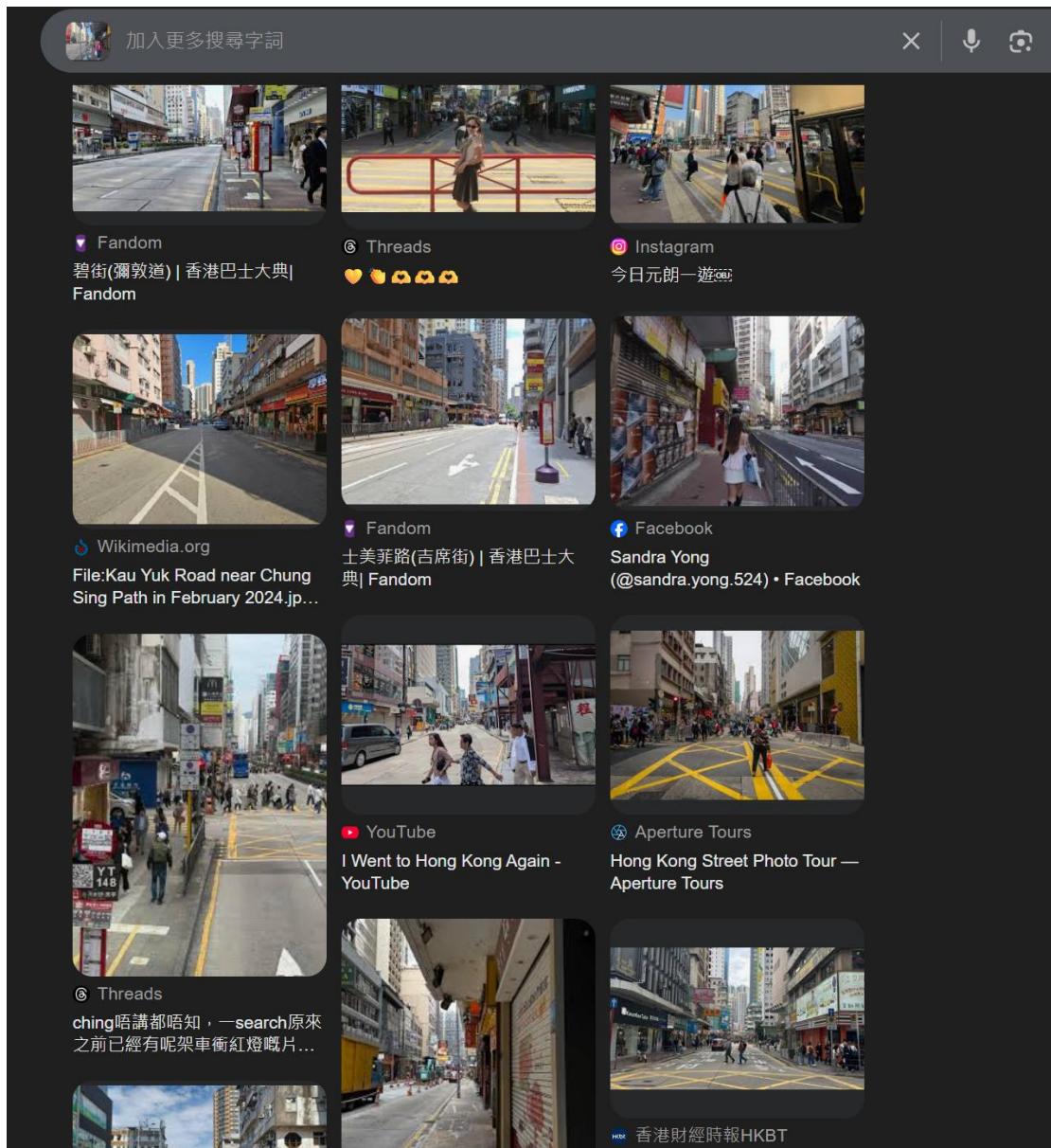
本题flag头: **furryCTF{}**
在一个稀松平常的下午，有一只laggy出去逛街.....
flag格式为**furryCTF{谷歌地球上拍摄者所处位置的经纬度，精确到整数秒}**
以下是一个示例（天安门城楼的经纬度）：
furryCTF{39°54'30"N 116°23'51"E}
本题的精度较高，请仔细确定拍摄者坐标后再提交。

下载附件 独游.jpg

该题目已被解出 提交 flag



这题挺简单的说实话，首先打开图片，看到最近的商铺店名是繁体字，锁定港澳台三地，直觉告诉我我是香港，谷歌识图一下



香港没跑了，图上明显文字线索



下面三个繁体字，前两个看不懂，但没关系，下面的
英语很清楚 TEMPORARY BUS STOP

浏览器搜索 281E 287D, 287X TEMPORARY BUS STOP

国内版 国际版

281E 287D, 287X TEMPORARY BUS STOP 0

网页 图片 视频 学术 词典 地图 更多

约 4 个结果 自适应缩放

拇指图标 反馈图标

Mongkok Market Complex

Bus Stop Mongkok Market Complex **Bus Stop**, Argyle Street (Sha Tin Bound) From 28 June 2025 From the first departure To 31 December 2025 23:59 (Estimated)
Details Relocates backward about 45 metres

search.kmb.hk
281E, 287D, 287X

kmb.hk
<https://search.kmb.hk> › KMBWebSite › AnnouncementPicture...

[PDF] 281E, 287D, 287X
Bus Stop Mongkok Market Complex Bus Stop, Argyle Street (Sha Tin Bound) From 28 June 2025 From the first departure To 31 December 2025 23:59 (Estimated) Details Relocates backward about 45 ...

fandom.com
<https://hkbust.fandom.com/wiki>

九巴281E線 | 香港巴士大典 | Fandom

歷史 行車路線

站位遷移
Stop Relocation

顧客服務熱線 2745 4466
Customer Service Hotline 2745 4466

KMB
www.kmb.hk

281E, 287D, 287X

巴士站暫時遷移

Bus Stop Temporary Relocation

巴士站	亞皆老街 · 旺角街市巴士站 (往沙田方向)
由	2025 年 6 月 28 日 頭班車起
至	2025 年 12 月 31 日 23:59 (預計)
詳情	向後移約 45 米

Bus Stop	Mongkok Market Complex Bus Stop, Argyle Street (Sha Tin Bound)
From	28 June 2025 From the first departure
To	31 December 2025 23:59 (Estimated)
Details	Relocates backward about 45 metres

亞皆老街 Argyle Street

← 往沙田方向 Sha Tin bound



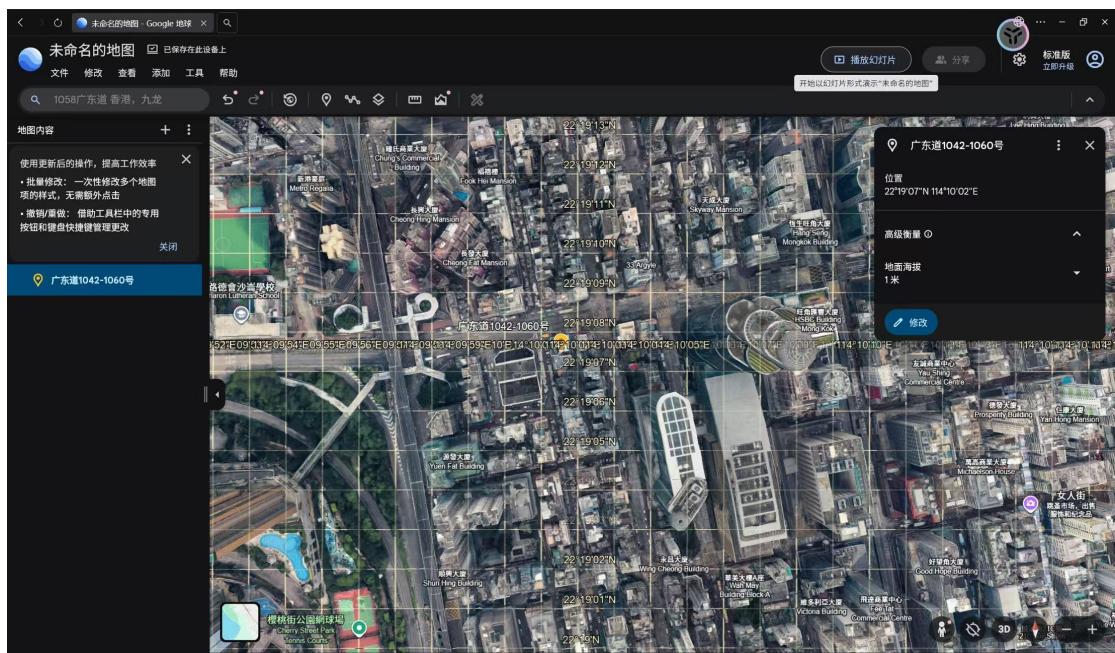
亞皆老街，旺角街市巴士站（往沙田方向）
在谷歌地球看街景



基本在这一块范围内了，在往前看看



基本是在这个位置，1058 广东道 看看经纬度



22° 19' 07"N 114° 10' 02"E

furryctf-wp

REVERSE

ezvm

1. 主函数结构分析

1.试运行附件程序，flag验证

```
D:\Dataself\ctf\2026\furryctf\ezvm-126\VM\EZ_VM.exe
```

```
input the flag:
```

2.Ida反编译，f5生成伪代码，分析main函数的结构

```

s1_1 = (char *)operator new(0x11u);
flag_init = "POFP{327a6c4304}";
s1 = s1_1;
do
{
    v6 = *flag_init;
    flag_init[s1_1 - "POFP{327a6c4304}"] = *flag_init;
    ++flag_init;
}
while ( v6 );
v7 = 0;
n976364816 = 976364816;
v8 = 0;
strcpy(vm_code, "\v%c:\vf\rA1Uf");
v9 = 0;
v10 = 0;
vm_code[12] = -1;
v11 = 1;
while ( 1 )
{
    switch ( v10 )
    {
        case 0:
            v9 = s1[v8];
            if ( s1[v8] )
                goto LABEL_13;
            goto LABEL_14;
            v10 = \unint8_t((int)v10 + v11) & 15;
            break;
        default:
LABEL_14:
            sub_140001510(std::cout, "input the flag: ");
            sub_140001730(std::cin, v16, s2);
            v17 = strcmp(s1, s2) == 0;
            right_flag_ = "right flag!";
            if ( !v17 )
                right_flag_ = "wrong flag!";
            v19 = sub_140001510(std::cout, right_flag_);
            std::ostream::operator<<(v19, sub_1400016F0);
            j_j_free(s1);
            return 0;
    }
}
}

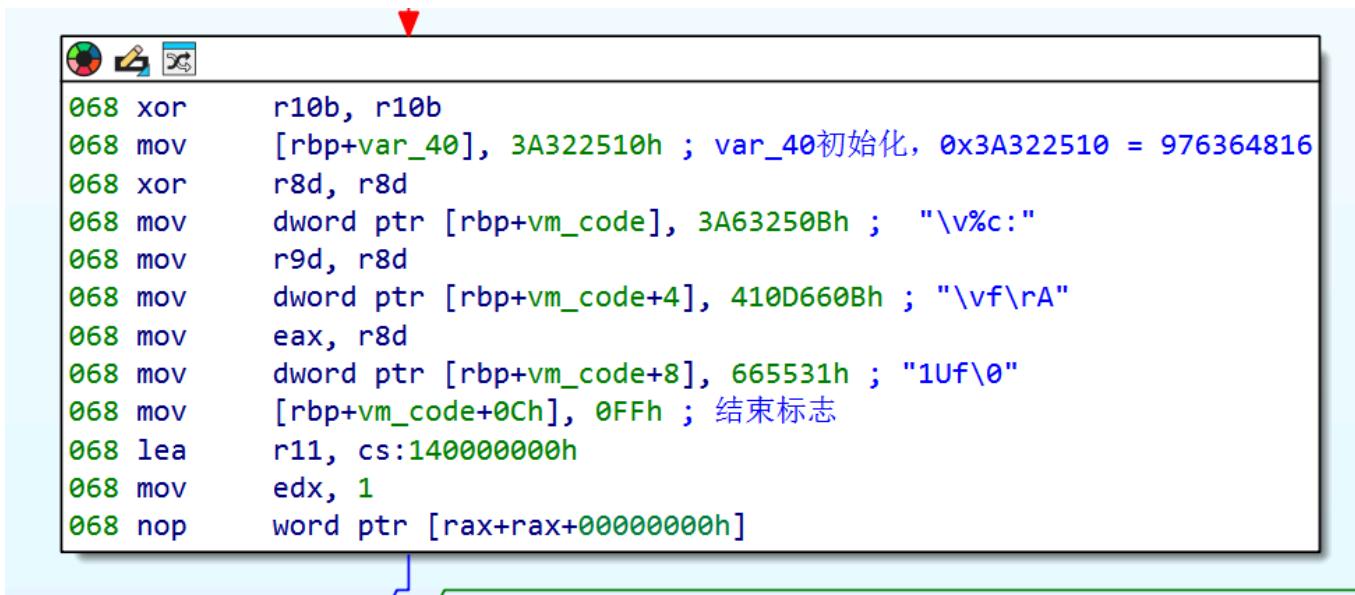
```

整体结构为：

- 初始化flag字符串：POFP{327a6c4304}
- VM指令数组初始化（vm_code）

- VM执行逻辑
- 详细VM代码
- 验证输出

2, VM初始化代码分析



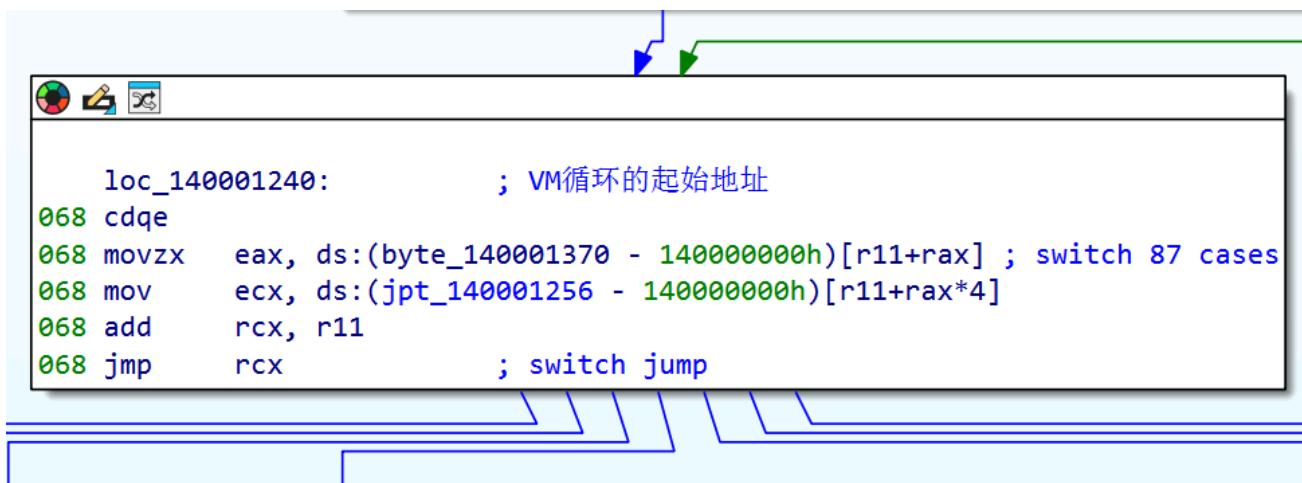
```

068 xor    r10b, r10b
068 mov    [rbp+var_40], 3A322510h ; var_40初始化, 0x3A322510 = 976364816
068 xor    r8d, r8d
068 mov    dword ptr [rbp+vm_code], 3A63250Bh ; "\v%c:"
068 mov    r9d, r8d
068 mov    dword ptr [rbp+vm_code+4], 410D660Bh ; "\vf\rA"
068 mov    eax, r8d
068 mov    dword ptr [rbp+vm_code+8], 665531h ; "1Uf\0"
068 mov    [rbp+vm_code+0Ch], 0FFh ; 结束标志
068 lea    r11, cs:140000000h
068 mov    edx, 1
068 nop    word ptr [rax+rax+00000000h]

```

3, VM循环详细分析

1. vm循环入口点



```

loc_140001240:          ; VM循环的起始地址
068 cdqe
068 movzx   eax, ds:(byte_140001370 - 140000000h)[r11+rax] ; switch 87 cases
068 mov     ecx, ds:(jpt_140001256 - 140000000h)[r11+rax*4]
068 add     rcx, r11
068 jmp     rcx           ; switch jump

```

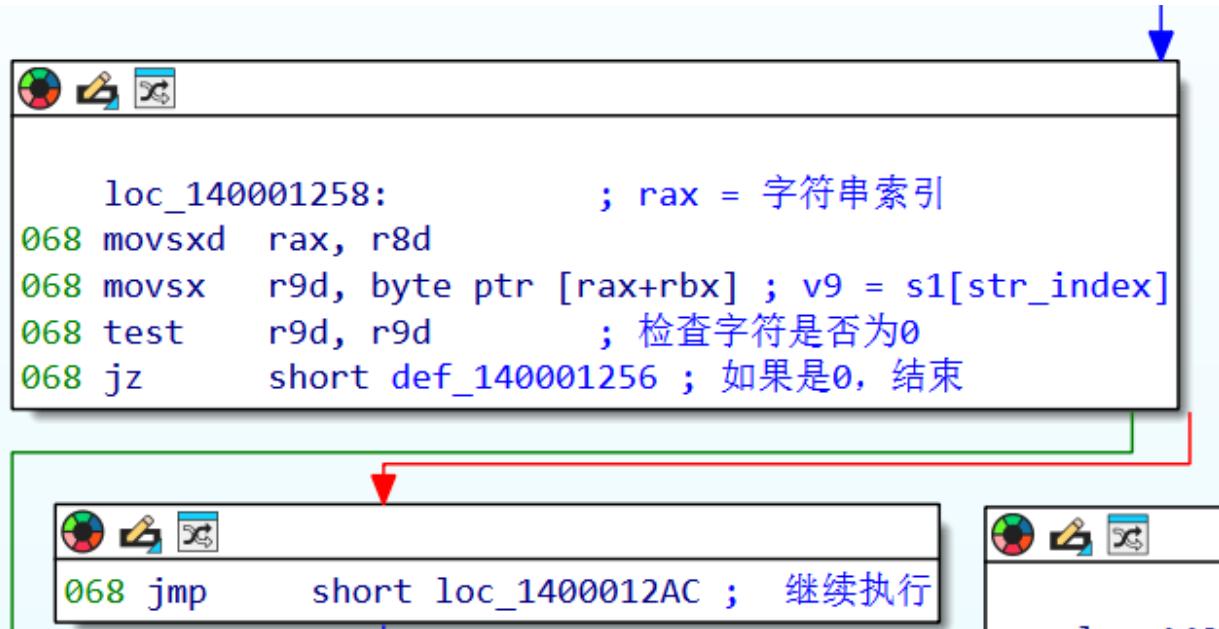
2. 通过分析汇编代码，识别出VM的虚拟寄存器：

寄存器	物理寄存器	用途	初始值
v8	r8d	字符串索引指针	0
v9	r9d	当前字符值	未定义
v10	eax	当前操作码	0
v11	edx	指令指针	1

寄存器	物理寄存器	用途	初始值
v7	r10b	比较结果标志	0

3. 各操作码处理逻辑详细分析

- 操作码 0 (0x10) - 读取字符



The screenshot shows two assembly code snippets in a debugger interface. The top snippet is highlighted with a blue box and shows the following code:

```

loc_140001258:          ; rax = 字符串索引
068 movsxrd  rax, r8d
068 movsx    r9d, byte ptr [rax+r8d] ; v9 = s1[str_index]
068 test     r9d, r9d           ; 检查字符是否为0
068 jz       short def_140001256 ; 如果是0, 结束

```

The bottom snippet is highlighted with a green box and shows the following code:

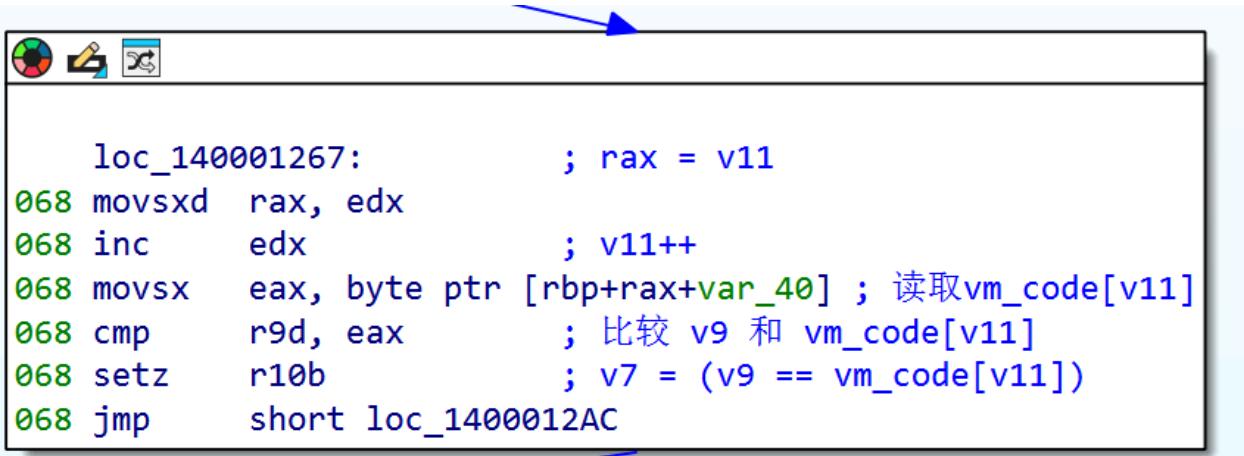
```

068 jmp     short loc_1400012AC ; 继续执行

```

作用：从字符串 s1 中读取当前索引位置的字符到 v9 寄存器。

- 操作码 21 (0x25) - 比较字符



The screenshot shows assembly code for character comparison. The code is highlighted with a blue box and includes comments explaining the operations:

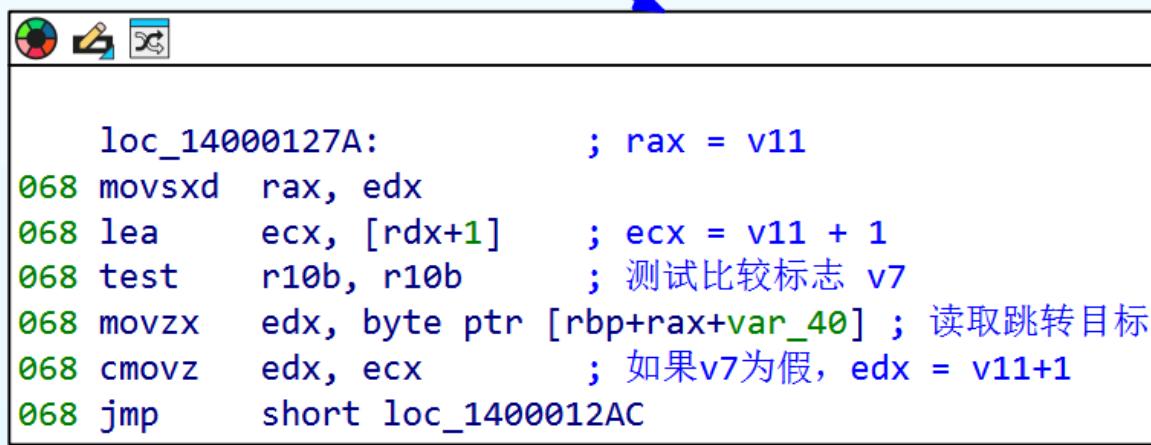
```

loc_140001267:          ; rax = v11
068 movsxrd  rax, edx
068 inc      edx           ; v11++
068 movsx    eax, byte ptr [rbp+rax+var_40] ; 读取vm_code[v11]
068 cmp      r9d, eax       ; 比较 v9 和 vm_code[v11]
068 setz    r10b           ; v7 = (v9 == vm_code[v11])
068 jmp     short loc_1400012AC

```

作用：比较当前字符 v9 与 vm_code[v11] 的值，结果存储在 v7 中。

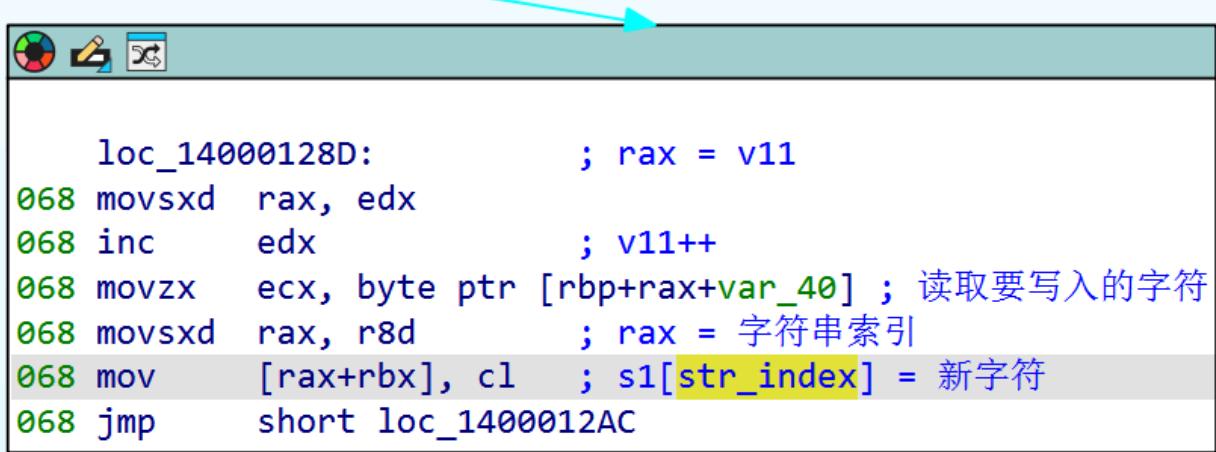
- 操作码 42 (0x3A) - 条件跳转



```
loc_14000127A:          ; rex = v11
068 movsxd  rax, edx
068 lea      ecx, [rdx+1]    ; ecx = v11 + 1
068 test     r10b, r10b      ; 测试比较标志 v7
068 movzx   edx, byte ptr [rbp+rax+var_40] ; 读取跳转目标
068 cmovz   edx, ecx        ; 如果v7为假, edx = v11+1
068 jmp     short loc_1400012AC
```

作用：如果比较标志 v7 为真，则跳转到 `vm_code[v11]` 指定的地址；否则继续顺序执行。

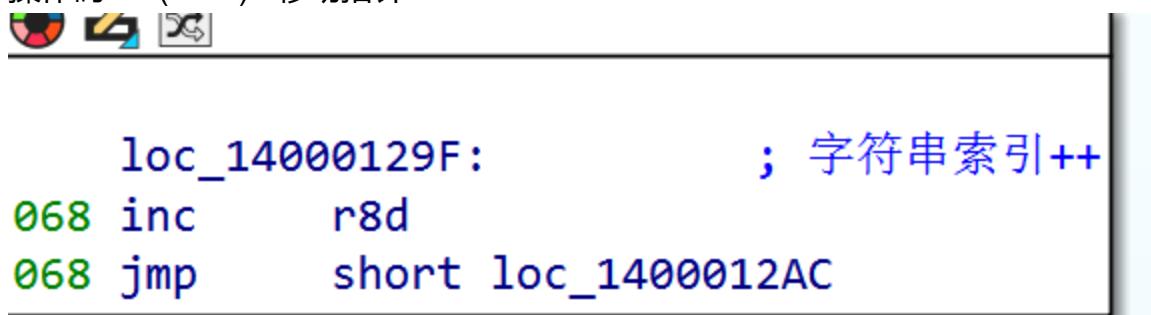
- 操作码 49 (0x41) - 写入字符



```
loc_14000128D:          ; rex = v11
068 movsxd  rax, edx
068 inc      edx           ; v11++
068 movzx   ecx, byte ptr [rbp+rax+var_40] ; 读取要写入的字符
068 movsxd  rax, r8d         ; rex = 字符串索引
068 mov     [rax+rbx], cl    ; s1[str_index] = 新字符
068 jmp     short loc_1400012AC
```

作用：将 `vm_code[v11]` 的值写入字符串 `s1` 的当前位置。

- 操作码 69 (0x55) - 移动指针



```
loc_14000129F:          ; 字符串索引++
068 inc      r8d
068 jmp     short loc_1400012AC
```

作用：将字符串索引 `v8` 加1，移动到下一个字符。

- 操作码 86 (0x66) - 无条件跳转

```
loc_1400012A4:          ; jumptable 0000000140001256 case 102
068 movsx rax, edx
068 movzx edx, byte ptr [rbp+rax+var_40] ; 读取跳转目标
```

作用：无条件跳转到 `vm_code[v11]` 指定的地址。

4. 基于以上分析，让ai编写详细的Python模拟器：

```
#!/usr/bin/env python3
"""
EZ_VM.exe虚拟机详细模拟器
包含完整的VM状态跟踪和调试输出
"""

class EZ_VM_Simulator:
    def __init__(self):
        # VM指令数组
        self.vm_code = [
            0x10, 0x25, 0x32, 0x3A,      # var_40
            0x0B, 0x25, 0x63, 0x3A,      # \v % c :
            0x0B, 0x66, 0x0D, 0x41,      # \v f \r A
            0x31, 0x55, 0x66, 0x00,      # 1 U f \0
            0xFF                         # 结束标志
        ]

        # 初始字符串
        self.s1 = list("POFP{327a6c4304}")

        # VM状态寄存器
        self.v8 = 0          # 字符串索引
        self.v9 = None       # 当前字符
        self.v10 = 0          # 当前操作码
        self.v11 = 1          # 指令指针
        self.v7 = False       # 比较标志

        # 执行历史
        self.steps = []
        self.step_count = 0

    def get_opcode_name(self, opcode):
        """获取操作码名称"""

```

```

opcode_names = {
    0: "READ_CHAR",
    21: "COMPARE_CHAR",
    42: "CONDITIONAL_JUMP",
    49: "WRITE_CHAR",
    69: "MOVE_POINTER",
    86: "UNCONDITIONAL_JUMP"
}
return opcode_names.get(opcode, f"UNKNOWN({opcode})")

def execute_step(self):
    """执行单步VM指令"""
    if self.v11 >= len(self.vm_code) or self.vm_code[self.v11] == 0xFF:
        return False

    # 记录当前状态
    step_info = {
        'step': self.step_count,
        'v11': self.v11,
        'v8': self.v8,
        'v7': self.v7,
        'v9': self.v9,
        's1': ''.join(self.s1)
    }

    # 读取操作码
    raw_byte = self.vm_code[self.v11]
    self.v10 = raw_byte - 0x10
    step_info['raw_byte'] = f"0x{raw_byte:02x}"
    step_info['opcode'] = self.v10
    step_info['opcode_name'] = self.get_opcode_name(self.v10)

    # 执行操作
    if self.v10 == 0:  # 读取字符
        if self.v8 < len(self.s1):
            self.v9 = self.s1[self.v8]
            step_info['action'] = f"读取 s1[{self.v8}] = '{self.v9}'"
        else:
            step_info['action'] = "字符串索引超出范围"

    elif self.v10 == 21:  # 比较字符
        self.v11 += 1
        if self.v11 < len(self.vm_code):
            expected = self.vm_code[self.v11]
            if self.v9 is not None:
                self.v7 = (ord(self.v9) == expected)

```

```
step_info['action'] = f"比较 '{self.v9}' ({0x{ord(self.v9)}:02x})\n与 {expected:02x}, 结果: {self.v7}"  
else:  
    step_info['action'] = "当前字符未定义, 无法比较"  
    self.v7 = False  
else:  
    step_info['action'] = "VM代码数组越界"  
  
elif self.v10 == 42: # 条件跳转  
    self.v11 += 1  
    if self.v11 < len(self.vm_code):  
        jump_target = self.vm_code[self.v11]  
        if self.v7:  
            old_v11 = self.v11  
            self.v11 = jump_target  
            step_info['action'] = f"条件跳转: 比较为真, 跳转到  
0x{jump_target:02x} (原v11={old_v11})"  
            step_info['jump'] = True  
        else:  
            step_info['action'] = f"条件跳转: 比较为假, 继续执行 (跳转目标:  
0x{jump_target:02x})"  
            step_info['jump'] = False  
    else:  
        step_info['action'] = "VM代码数组越界"  
  
elif self.v10 == 49: # 写入字符  
    self.v11 += 1  
    if self.v11 < len(self.vm_code) and self.v8 < len(self.s1):  
        new_char_val = self.vm_code[self.v11]  
        new_char = chr(new_char_val)  
        old_char = self.s1[self.v8]  
        self.s1[self.v8] = new_char  
        step_info['action'] = f"写入字符: s1[{self.v8}] = '{new_char}'  
(0x{new_char_val:02x}), 原字符: '{old_char}'"  
    else:  
        step_info['action'] = "VM代码数组越界或字符串索引越界"  
  
elif self.v10 == 69: # 移动指针  
    old_v8 = self.v8  
    self.v8 += 1  
    step_info['action'] = f"移动指针: v8从{old_v8}增加到{self.v8}"  
  
elif self.v10 == 86: # 无条件跳转  
    self.v11 += 1  
    if self.v11 < len(self.vm_code):  
        jump_target = self.vm_code[self.v11]
```

```

        old_v11 = self.v11
        self.v11 = jump_target
        step_info['action'] = f"无条件跳转: 跳转到 0x{jump_target:02x} (原
v11={old_v11})"
            step_info['jump'] = True
        else:
            step_info['action'] = "VM代码数组越界"

    else:
        step_info['action'] = f"未知操作码: {self.v10}"
        return False

# 如果不是跳转指令, v11自增
if self.v10 not in [42, 86] or not step_info.get('jump', False):
    self.v11 += 1

# 保存步骤信息
self.steps.append(step_info)
self.step_count += 1

return True

def run(self, max_steps=200):
    """运行VM模拟器"""
    print("== EZ_VM虚拟机详细模拟 ==")
    print(f"初始字符串: {''.join(self.s1)}")
    print(f"VM代码长度: {len(self.vm_code)} 字节")
    print(f"初始状态: v8={self.v8}, v11={self.v11}")
    print("-" * 80)

    step_num = 0
    while step_num < max_steps and self.execute_step():
        step_info = self.steps[-1]
        print(f"步骤 {step_info['step'][:3]}: v11={step_info['v11'][:2]}, 操作码=
{step_info['opcode_name'][:20]}")
        print(f"    原始字节: {step_info['raw_byte']}, 操作:
{step_info['action']}")
        print(f"    当前状态: v8={step_info['v8']}, v7={step_info['v7']}, v9=
{step_info['v9']}")
        print(f"    当前字符串: {step_info['s1']}")
        print("-" * 80)
        step_num += 1

    print(f"\n模拟完成! 总步数: {self.step_count}")
    print(f"最终字符串: {''.join(self.s1)}")
    print(f"最终状态: v8={self.v8}, v11={self.v11}, v7={self.v7}")

```

```

        return ''.join(self.s1)

def main():
    """主函数"""
    simulator = EZ_VM_Simulator()
    result = simulator.run()

    print(f"\n==== 分析结果 ===")
    print(f"VM修改后的字符串: {result}")

    # 验证flag格式
    if result.startswith("POFP{") and result.endswith("}"):
        print(f"✓ 有效的flag格式: {result}")
    else:
        print(f"⚠ 非标准flag格式: {result}")

    return result

if __name__ == "__main__":
    flag = main()
    print(f"\n最终flag: {flag}")

```

5，运行脚本得到flag

```

模拟完成！总步数: 200
最终字符串: POFP{317a614304}
最终状态: v8=25, v11=5, v7=False

==== 分析结果 ===
VM修改后的字符串: POFP{317a614304}
✓ 有效的flag格式: POFP{317a614304}

最终flag: POFP{317a614304}

```

Lua

题目信息

- 文件: hello.lua
- 类型: Lua脚本逆向
- Flag格式: POFP{...}

分析

1. 脚本结构分析

首先查看hello.lua的内容：

```
local b = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
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%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 args = {...}

if #args ~= 1 then
    print("[-] use `lua hello.lua flag{fake_flag}`")
    return
end

print(load(dec("G0x1YVQAGZMNChoKBAgIeFYAAAAAAAAAAACh3QAGAoa4BAA6gkwAAAIFIABgf9
/tAEAAJUBA36vAYAHQIAgEqBCQALAwAADgMGAYADAQAVBAwArwKABosEAAKOBakDCwUAAg4FCgSABQAAFQ
YFgk8CgAaVBgWArwKABkQFBADEBAAcNwQJBbAEBQ9EAwQBSQEKA8BAABFgQEAEoEAAEaBAQCBIZ0YWjsZ
QSHaw5zZXJ0BIdzdHJpbmcEhWJ5dGUEhHN1YgNyAAAAAAAIEAAACBgKetAAADjQsAAAAOAAAbiQABAAMB
AQBEAMCPAADADgBAIADAAIASAACALgAAIADgAIASAACAEcAAQCBIZ0YWjsZQSHY29uY2F0BIItFL0yMC0
zM0x0S0yMS05LTM5LTQ1LTAtNDUtNjItNy03MC0zOC00NS02My03MC0xLTYtNjUtMzItODMtMTUEj1lvdS
BBcmUgUmlnaHQhBIdXcm9uZyGCAAAAQEAgICAgnICA"))(args[1]))
```

2. 关键逻辑理解

脚本的主要逻辑：

1. 定义了一个自定义的base64解码函数 `dec()`
2. 检查命令行参数数量，要求输入一个参数（flag）
3. 使用 `dec()` 函数解码一个长字符串
4. 使用 `load()` 加载解码后的内容（Lua字节码）

5. 执行加载的字节码，传入用户输入的参数

3. 解码base64字符串

首先需要理解 `dec()` 函数的作用。这是一个自定义的base64解码实现。我们可以用赛博厨子解码：

The screenshot shows the CyberChef interface with two sections: 'Input' and 'Output'.
Input section:
The input is a long base64 encoded string. A portion of it is highlighted in yellow: 'MC0xOS0yMS05LTM5LTQ1LTAtNDUTNjItNy03MC0z0C00NS02My03MC0xLTYtNjUtMzItODMtMTUEj1lv'.
Output section:
The output shows the decoded ASCII characters. A portion of the output is highlighted in green: '20-30-19-21-9-39-45-0-45-62-7-70-38-45-63-70-1-6-65-32-83-15'.
Below the output, a message says 'You Are Right!'.

关键发现：数字序列 20-30-19-21-9-39-45-0-45-62-7-70-38-45-63-70-1-6-65-32-83-15

6. 尝试xor解密

```
numbers = [20, 30, 19, 21, 9, 39, 45, 0, 45, 62, 7, 70, 38, 45, 63, 70, 1, 6, 65, 32, 83, 15]

for key in range(256):
    result_chars = []
    valid = True
    for n in numbers:
        c = n ^ key
        if 32 <= c <= 126: # 可打印ASCII范围
            result_chars.append(chr(c))
        else:
            valid = False
            break
    if valid:
        result_str = ''.join(result_chars)
```

```
if 'flag' in result_str.lower() or '{' in result_str or '}' in result_str:  
    print(f'XOR密钥 {key:3d} ({0x{key:02x}}): {result_str}')
```

运行得到：

```
XOR密钥 105 (0x69): }wz|`NDiDWh/OD  
V/ho(I:f  
XOR密钥 110 (0x6e): zp}{gICnCPi(HC  
Q(oh/N=a  
XOR密钥 111 (0x6f): {q|zfHBoBQh)IB  
P)ni.O<`  
XOR密钥 114 (0x72): flag{U_r_Lu4T_  
M4st3R!}  
XOR密钥 116 (0x74): `jga}SYtYJs2RYK2ur5T'{  
XOR密钥 122 (0x7a): ndios]WzWD}<\WE<{|;Z)u  
XOR密钥 123 (0x7b): oehnr\V{VE|=]VD=z}:[(t
```

可以看到其中密钥为114的时候，结果是
flag{U_r_Lu4T_M4st3R!}

但是比赛说：flag头是POFP{

所以最终flag是POFP{U_r_Lu4T_M4st3R!}

RRRacket

信息收集

搜索.zo文件

- 是 Racket 语言的编译字节码文件
- 由 .rkt (源文件) 编译生成

类似.py文件编译生成.pyc文件

分析

1. 下载安装便携版Racket

2. 使用raco decompile反编译chall.zo

```
D:\Downloads\racket-minimal-9.0-x86_64-win32-cs\racket>raco decompile chall.zo > decompiled.rkt
```

```
D:\Downloads\racket-minimal-9.0-x86_64-win32-cs\racket>
```

打开rkt文件发现包含大量机器码

```
(require (lib "racket/main.rkt") (lib "racket/format.rkt"))
'(recurs: (lib "racket/main.rkt"))
(provide)
(define byte->hex
  (lambda (a0)
    "D:\\\\Downloads\\\\racket-minimal-9.0-x86_64-win32-cs\\\\racket\\\\1.rkt:11:1"
    (#%machine-code
     #\"I\\203nh\\1\\17\\204\\314\\1\\0\\0H\\203\\375\\1\\17\\205\\266\\1\\0\\0I\\211u\\20M9nH\\17\\206
(define bytes->hex
  (lambda (a0)
    "D:\\\\Downloads\\\\racket-minimal-9.0-x86_64-win32-cs\\\\racket\\\\1.rkt:14:1"
    (#%machine-code
     #\"I\\203nh\\1\\17\\204\\_3\\0\\0H\\203\\375\\1\\17\\205I\\3\\0\\0I\\211u\\20M\\211}\\bM9nH\\17\\2
(define rc4-bytes
  (lambda (a0 a1)
    "D:\\\\Downloads\\\\racket-minimal-9.0-x86_64-win32-cs\\\\racket\\\\1.rkt:20:1"
    (#%machine-code
     #\"I\\203nh\\1\\17\\204\\215\\24\\0\\0H\\203\\375\\2\\17\\205w\\24\\0\\0I\\211u\\20I\\211U\\30M\\2
(define read-line/trim
  (lambda ()
    "D:\\\\Downloads\\\\racket-minimal-9.0-x86_64-win32-cs\\\\racket\\\\1.rkt:59:1"
    (#%machine-code
```

但可以看到有byte -> hex 和 rc4-bytes模块

3， 使用Linux命令行， 查找关键字符串

```
strings chall.zo | grep -i -E "(key|target|flag|rc4|crypt)"
```

```
[root@kali]# strings chall.zo | grep -i -E "(key|target|flag|rc4|crypt)"
popkey
$      rc4-bytes
key;
KEY-STR#
TARGET-HEX#
      rc4-bytes#
keyword-procedure-extract
struct:keyword-procedure
      rc4-bytes
TARGET-HEX
KEY-STR
KEY-STR
TARGET-HEX$
      rc4-(
key[
Input flag:
key#
TARGET-HEX
      rc4-bytes
KEY-STR
```

这明显是一个加密算法：

1. 程序会要求输入 flag (Input flag:)
2. 使用 RC4 加密 (rc4-bytes)
3. 与目标值比较 (TARGET-HEX)
4. 输出 Correct! 或 Wrong!

4, 查找关键数据

1. 寻找加密的目标值

在文件中搜索十六进制字符串：

```
import re

with open('chall.zo', 'rb') as f:
    data = f.read()

# 查找长十六进制字符串

hex_pattern = re.compile(b'[0-9a-fA-F]{32,}')

matches = hex_pattern.findall(data)

print(matches)
```

运行

```
PS D:\Dataself\code> & "D:/Program Files/Python/Python313/python.exe" d:/Dataself/code/find_target.py
[b'd31fa2c26c024feddef9b38853790c00285e367b916d49a111bfc2bcfb74']
```

找到目标密文：

```
d31fa2c26c024feddef9b38853790c00285e367b916d49a111bfc2bcfb74
```

2. 寻找密钥

搜索出现“key”的地址

```
with open('chall.zo', 'rb') as f:
    data = f.read()
```

```

# 搜索所有包含 "key" 的位置
key_positions = []
pos = 0
while True:
    pos = data.find(b'key', pos)
    if pos == -1:
        break
    key_positions.append(pos)
    pos += 1

print(f"Found 'key' at positions: {[hex(p) for p in key_positions]}")

```

```

PS D:\Dataself\code> & "D:/Program Files/Python/Python313/python.exe" d:/Dataself/code/find_key.py
Found 'key' at positions: [ '0xd8d', '0x19b8', '0x1fa8', '0x1fca', '0x34dd', '0x3f4c', '0x4006' ]

```

先查看 0xd8d 附近的上下文

```

with open('chall.zo', 'rb') as f:
    data = f.read()

key_pos = 0xd8d
start = max(0, key_pos - 50)
end = min(len(data), key_pos + 50)
context = data[start:end]

print(f"Context around 'key' at {hex(key_pos)}:")
for i in range(0, len(context), 16):
    chunk = context[i:i+16]
    hex_part = ' '.join(f'{b:02x}' for b in chunk)
    ascii_part = ''.join(chr(b) if 32 <= b < 127 else '.' for b in chunk)
    print(f" {start+i:04x}: {hex_part:<48} {ascii_part}")

```

```

Context around 'key' at 0xd8d:
0d5b: b8 32 00 51 ff e0 ee 03 00 01 00 43 d1 7f fb 03 .2.Q.....C....
0d6b: 31 02 00 2e 00 0a b6 03 57 e9 5a fc ff ff 36 00 1.....W.Z...6.
0d7b: 3a d0 e9 3e 11 00 ff 37 e0 19 00 74 27 07 70 6f :...>....7...t'.po
0d8b: 66 70 6b 65 79 00 0a 11 00 02 08 63 6f 6e 73 74 fpkey.....const
0d9b: 61 6e 74 00 18 11 01 13 14 76 61 72 69 61 62 6c ant.....variabl
0dab: 65 2d 73 65 74 21 2f 64 65 66 69 6e 65 26 72 6b e-set!/define&rkt
0dbb: 74 2d 6c 69 t-li

```

在文件中发现 "popkey" 字符串，可能是密钥。

5, 总结

rc4加密

密文 (hex) : d31fa2c26c024feddef9b38853790c00285e367b916d49a111bfc2bcfb74

密钥: pofpkey

6, 解密脚本

```
import binascii

def rc4(key, data):
    """
    由于 RC4 是对称加密，加密和解密使用相同的函数
    """

    # 初始化 S-box
    S = list(range(256))
    j = 0
    key_length = len(key)
    # KSA (Key Scheduling Algorithm)
    for i in range(256):
        j = (j + S[i] + key[i % key_length]) % 256
        S[i], S[j] = S[j], S[i]

    # PRGA (Pseudo-Random Generation Algorithm)
    i = j = 0
    result = bytearray()

    for byte in data:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        k = S[(S[i] + S[j]) % 256]
        result.append(byte ^ k)

    return bytes(result)

target_hex = "d31fa2c26c024feddef9b38853790c00285e367b916d49a111bfc2bcfb74"
ciphertext = binascii.unhexlify(target_hex)
key = b"pofpkey"

decrypted = rc4(key, ciphertext)
print(f"Decrypted: {decrypted}")
print(f"ASCII: {decrypted.decode('ascii')}"")
```

7, 运行得到flag

```
PS D:\Dataself\code> & "D:/Program Files/Python/Python313/python.exe" d:/Dataself/code/rc4.py
Decrypted: b'POFP{Racket_and_rc4_you_know!}'
ASCII: POFP{Racket_and_rc4_you_know!}
PS D:\Dataself\code>
```

POFP{Racket_and_rc4_you_know!}