《安全程序设计与实践》
实验报告

实验名称：Cookie 注入攻击和 HTTP 头部注入攻击

实验类型：实操型

指导教师：孙海峰

专业班级：

姓　　名：

学　　号：

实验地点：东 11C227

实验日期：2025/5/14

一、 实验目的

1. 可以解释和分析 Cookie 注入漏洞产生的原理及危害
2. 能够应用转义函数或者参数化查询实现对 cookie sql 注入的防护
3. 可以解释和分析 HTTP 头部 SQL 注入漏洞产生的原理及危害
4. 应用转义函数和参数化插入实现 HTTP 头部 SQL 注入攻击防护

二、 实验过程

Cookie 注入攻击

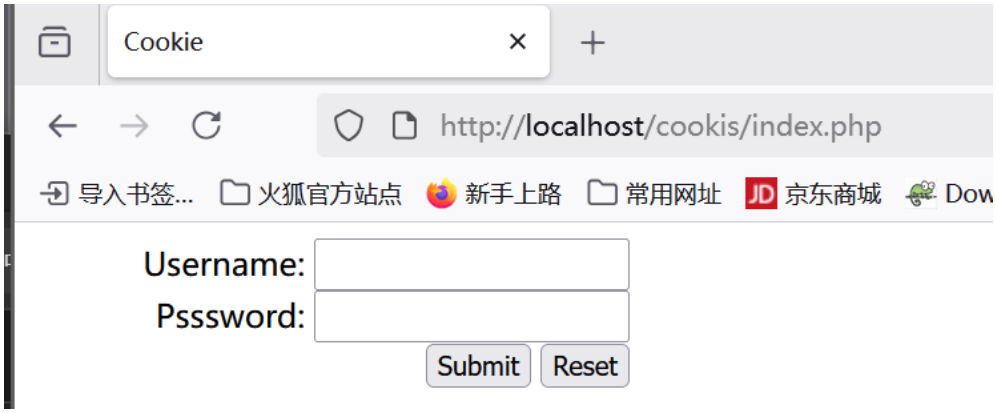
任务一：建立具有 Cookie 验证功能的网站

1-1 任务实现

更新 admin 的密码为 admin123

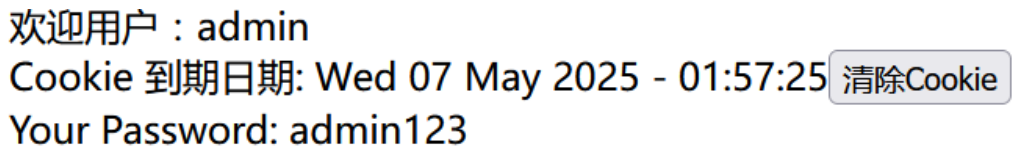
```
mysql> update users set passcode='admin123' where id='1';
Query OK, 1 row affected (5.48 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from users;
+----+----------+----------+
| id | username | passcode |
+----+----------+----------+
|  1 | admin    | admin123 |
|  2 | alice    | alice456 |
|  3 | test     | 123      |
|  6 | admin'#  | 123456   |
|  7 | ttt      | 111      |
+----+----------+----------+
5 rows in set (0.00 sec)
```

网页效果：



1-2 Cookie 验证功能测试

登录用户 admin



关闭浏览器后再次打开页面，发现不需要输入账户密码，直接登录进去。

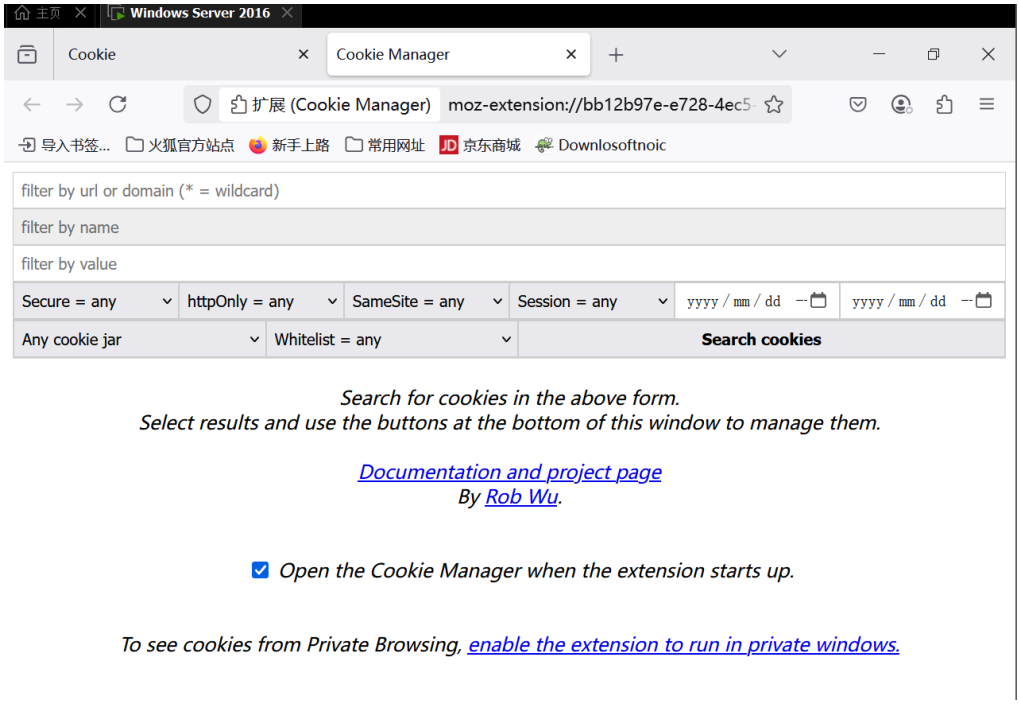
这是因为浏览器存储了用户的 cookie，记录了用户登录凭证，即使关闭浏览器，cookie 仍会保存，直到过期或被清理。

点击清除 cookie 按钮，发现页面刷新后需要登录。

任务二：Cookie 注入攻击测试

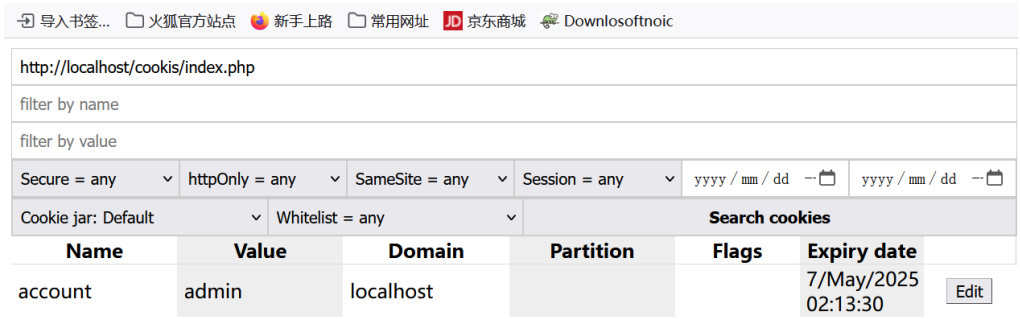
2-1 安装浏览器插件

安装 Cookie Manager 插件

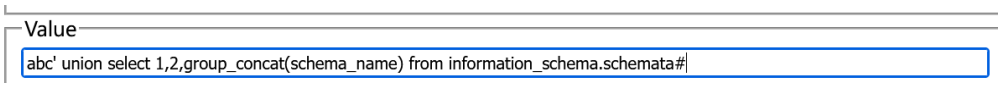


2-2 Cookie 注入攻击过程

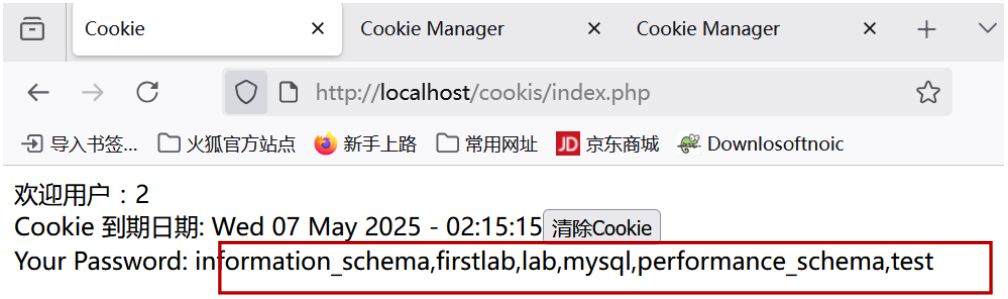
登陆 admin 用户之后打开插件，可以看到该插件记录了相关 cookie



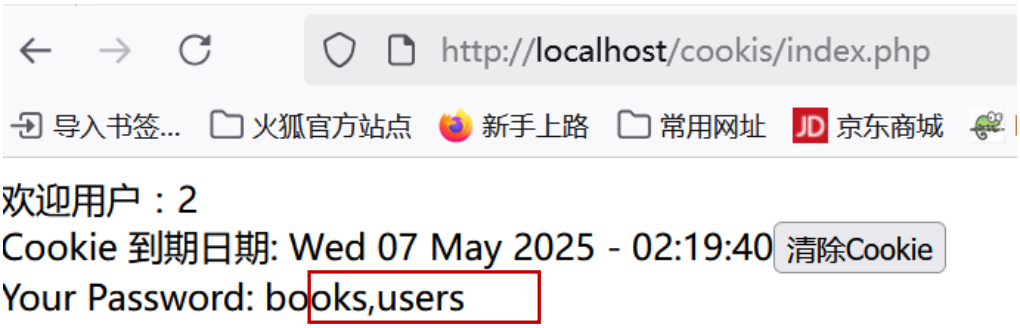
将 value admin 修改为 abc' union select 1,2,group_concat(schema_name) from information_schema.schemata#进行 sql 注入攻击，获取当前数据库服务器上的所有数据库名称。



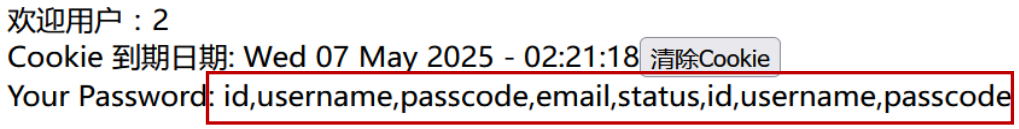
刷新页面，发现登录成功页面“Your Password”处出现了数据库名。



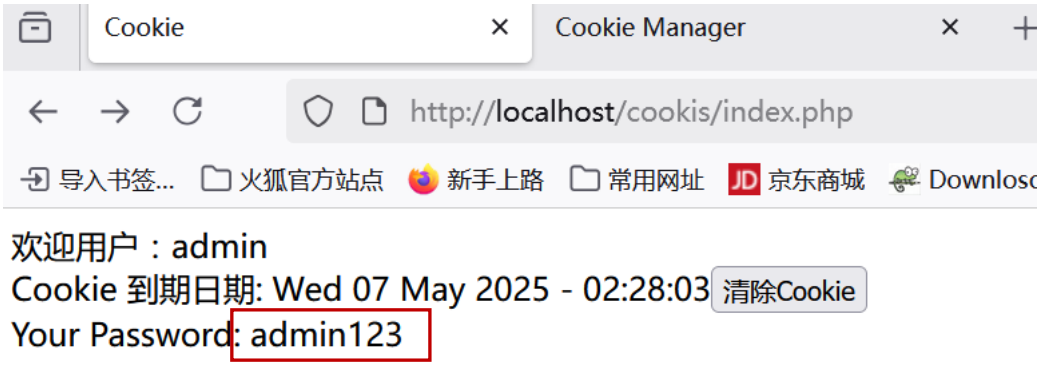
进行暴表名 abc' union select 1,2,group_concat(table_name) from information_schema.tables where table_schema='lab'#



暴列名 abc' union select 1,2,group_concat(column_name) from information_schema.columns where table_name='users'#



读取 username 和 passcode。abc' union select 1,username,passcode from users#



2-3 测试分析

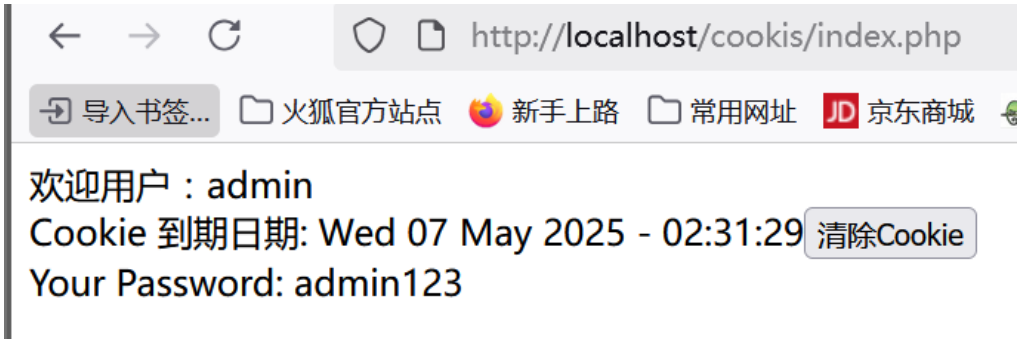
```
mysql> select * from books where id='abc' union select 1,username,passcode from users#
    -> ;
+----+----------+----------+
| id | bookname | author   |
+----+----------+----------+
|  1 | admin    | admin123 |
|  1 | alice    | alice456 |
|  1 | test     | 123      |
|  1 | admin'#  | 123456   |
|  1 | ttt      | 111      |
+----+----------+----------+
5 rows in set (0.00 sec)
```

任务三：Cookie 注入攻击防护

使用 mysqli_real_escape_string 对 Cookie 值进行转义，特殊字符会被转义，无法构造有效注入语句，避免恶意 SQL 注入。

```
62      else
63      {
64          $cookee = mysqli_real_escape_string($con,$_COOKIE['account']);
65          //根据Cookie查询用户信息
66          $sql = "select * from users where username = '$cookee' ";
```

进行测试，发现 sql 注入后，欢迎页面无敏感信息出现。



HTTP 头部注入攻击

任务一：创建数据库

我们将 sql 脚本文件导入数据库，查看 uagents 表结构。

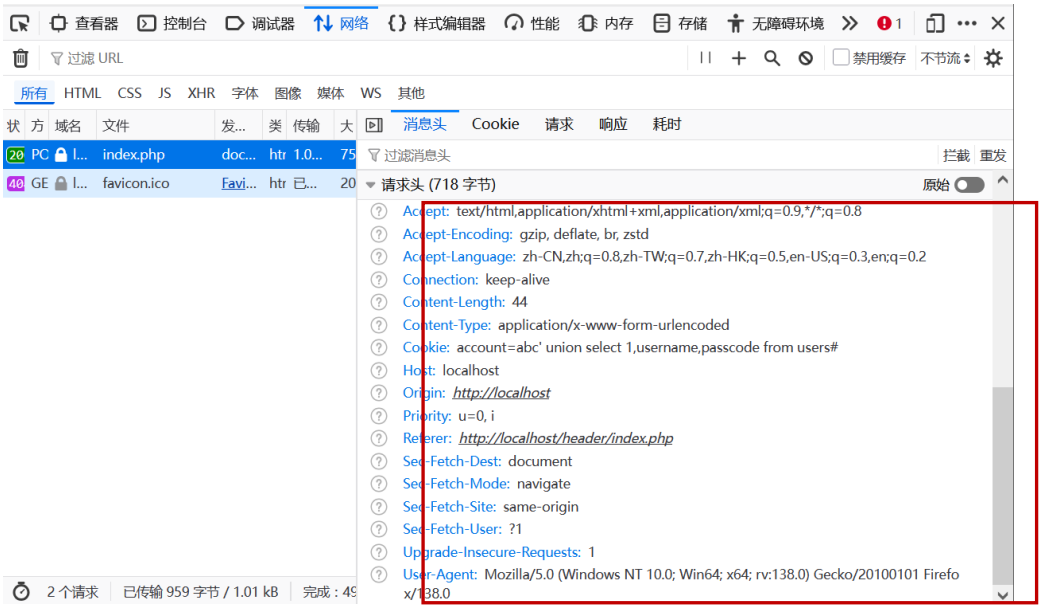
```
mysql> source C:\Users\Administrator\Desktop\Web\apache2.4\httpd-2.4.33-o102o-x86-vc14-r2\Apache24
ocs\header\lab.sql
Query OK, 1 row affected, 1 warning (0.09 sec)

Database changed
Query OK, 0 rows affected, 1 warning (0.01 sec)

Query OK, 0 rows affected (0.22 sec)

mysql> use lab;
Database changed
mysql> desc uagents;
+------------+-----------+------+-----+---------+----------------+
| Field      | Type      | Null | Key | Default | Extra          |
+------------+-----------+------+-----+---------+----------------+
| id         | int(11)   | NO   | PRI | NULL    | auto_increment |
| username   | char(32)  | YES  |     | NULL    |                |
| uagent     | char(128) | YES  |     | NULL    |                |
| ip_address | char(32)  | YES  |     | NULL    |                |
+------------+-----------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

任务二：建立具有 HTTP 头部信息保存功能的网站

2-1 任务实现

2-2 HTTP 头部信息保存功能测试

打开开发人员工具，点击“网络”，点击 index.php 的网络流量，可以看到该流量的相关消息头，有 Accpet、Cookie、Referer、User-Agent 等头部信息。

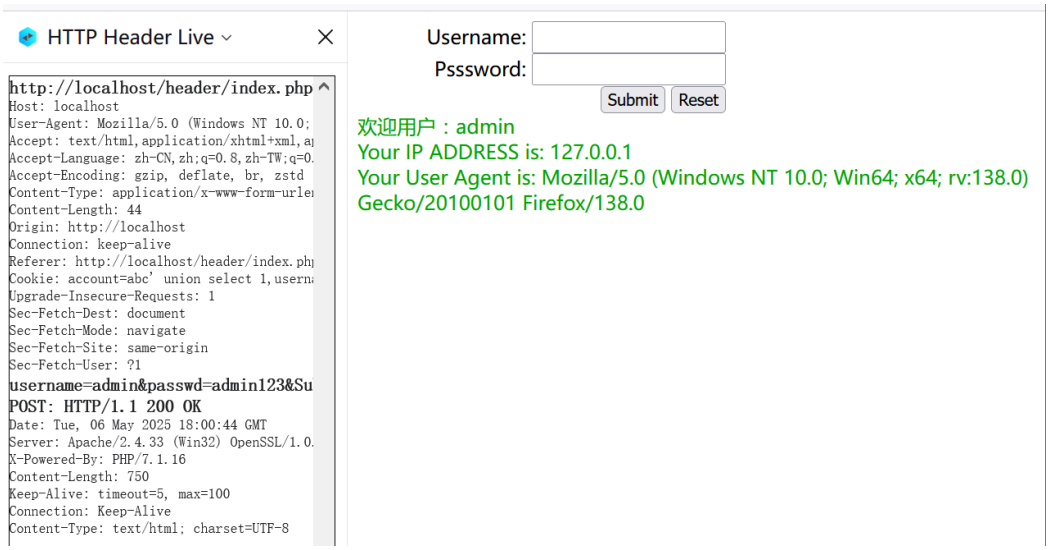


任务三：HTTP 头部注入攻击测试

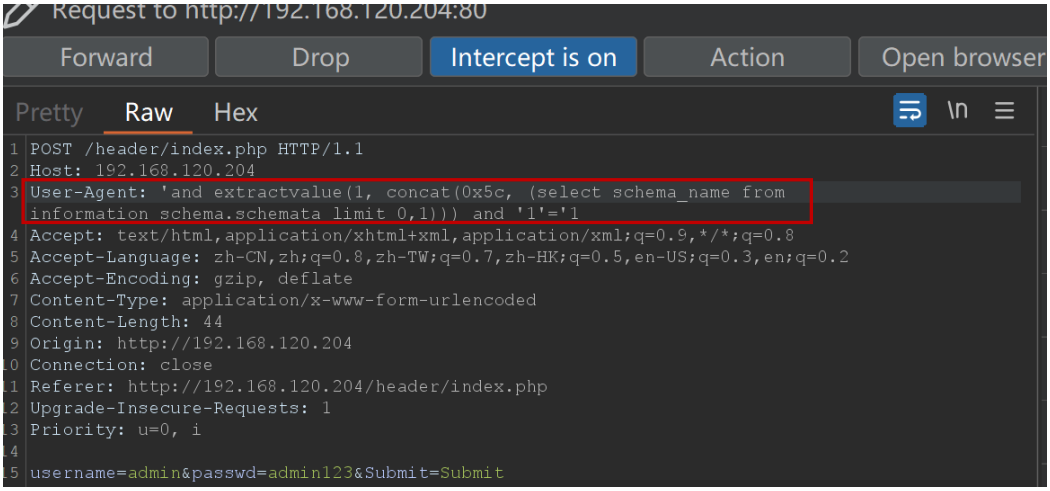
3-1 安装浏览器插件：

3-2 HTTP 头部注入攻击

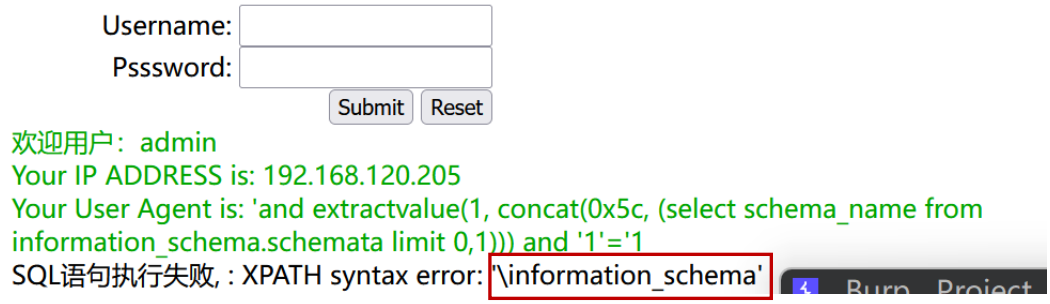
使用插件，发现无法修改头部信息之后进行重发



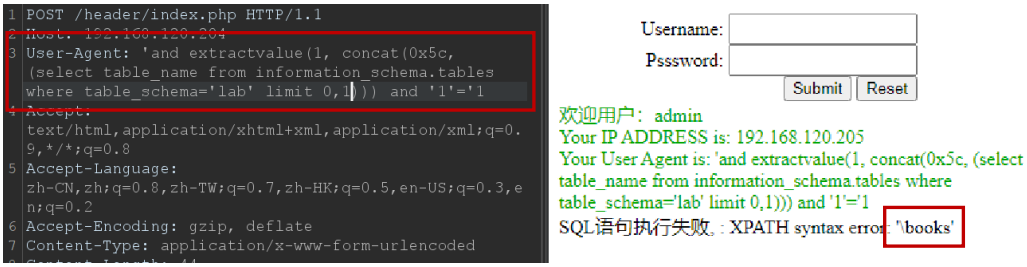
我们使用 bp 抓包修改



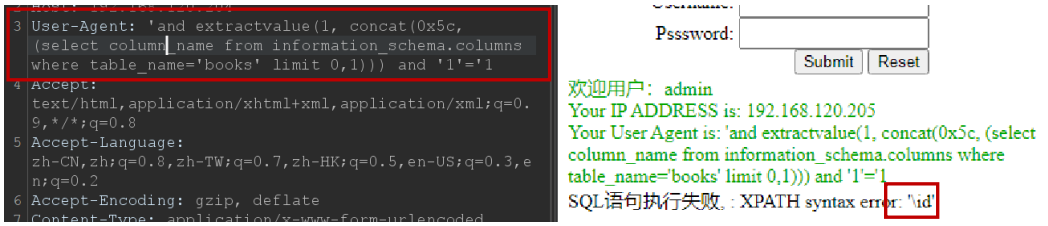
点击 forword 放包，可以看到欢迎页面出现报错，报错信息有数据库名



执行成功，接下来我们进行暴表名'and extractvalue(1, concat(0x5c, (select table_name from information_schema.tables where table_schema='lab' limit 0,1))) and '1'='1



执行成功，存在 books 表，接下来我们进行暴列名，'and extractvalue(1, concat(0x5c, (select column_name from information_schema.columns where table_name='books' limit 0,1))) and '1'='1



存在 id 列名。

3-3 测试分析

使用的是报错注入，extractvalue()是 MySQL 的 XML 处理函数，用于故意引发错误，通过 concat(0x5c,...)在返回数据前添加反斜杠(\)，导致 XML 解析错误，从而导致错误信息中包含查询结构。

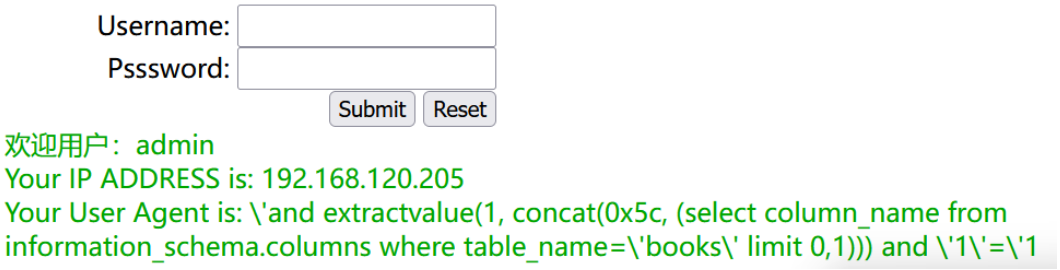
任务四：HTTP 头部注入攻击防护

4-1 转义函数：

使用 mysql_escape_string 对 UserAgent 进行转义，防止 HTTP 头部注入攻击

```
$uagent = mysqli_escape_string($con,$_SERVER['HTTP_USER_AGENT']);
$IP = mysqli_escape_string($con,$_SERVER['REMOTE_ADDR']);
```

进行注入测试，可以发现 User Agent 处输出注入语句（特殊字符进行了转义），注入失败。



4-2 MySQLi 参数化插入防注入：

MySQLi 参数化查询使用预处理语句，把 SQL 语句的结构和用户输入的数据分开。

然后通过 mysqli_stmt_bind_param 将相关输入参数（如 useragent）作为参数绑定到占位符，数据库会将这些参数视为纯数据，不会被当作 sql 语句执行，从而防止注入。

```
//mysqli_query($con,$insert) or die('SQL语句执行失败, : '.mysqli_error($con));
$stmt=$con->prepare($insert);
if(!$stmt)
    echo 'prepare执行错误';
else
{
    $stmt->bind_param("sss",$username,$uagent,$IP);
    $stmt->execute();
}
$stmt->close();
```

进行测试，发现注入失败

Username:
Psssword:
Submit Reset

欢迎用户：admin
Your IP ADDRESS is: 192.168.120.205
Your User Agent is: 'and extractvalue(1, concat(0x5c, (select schema_name from information_schema.schemata limit 0,1))) and '1'='1

三、 实验结论

本实验验证了 HTTP 头部存在的 SQL 注入漏洞及其安全风险。通过篡改 HTTP 头部数据，成功实施了报错注入攻击，获取了数据库名、表名及列名等敏感信息，证明传统 SQL 语句拼接方式存在严重安全隐患。实验采用 mysql_escape_string 转义函数和 MySQLi 参数化查询两种防护方案，均有效阻断注入攻击，其中参数化查询通过预编译机制实现 SQL 指令与数据分离，从根本上消除注入可能。

实验同时验证了 Cookie 的存储特性及其潜在风险：用户登录后，服务器生成的 Cookie 若未经安全处理直接拼接到 SQL 语句，攻击者可篡改 Cookie 值构造恶意 SQL，绕过身份验证或窃取数据。通过输入过滤与参数化查询，可有效防御此类攻击。

综上，HTTP 头部注入与 Cookie 注入均源于未对用户输入进行严格过滤，防护需遵循“数据输入必过滤、SQL 查询必参数化”原则。实验证实转义函数、MySQLi/PDO 参数化查询均为有效防护手段，为 Web 应用安全开发提供了实践依据。

四、 思考题

1. 如何使用参数化查询来防护 Cookie 注入攻击？

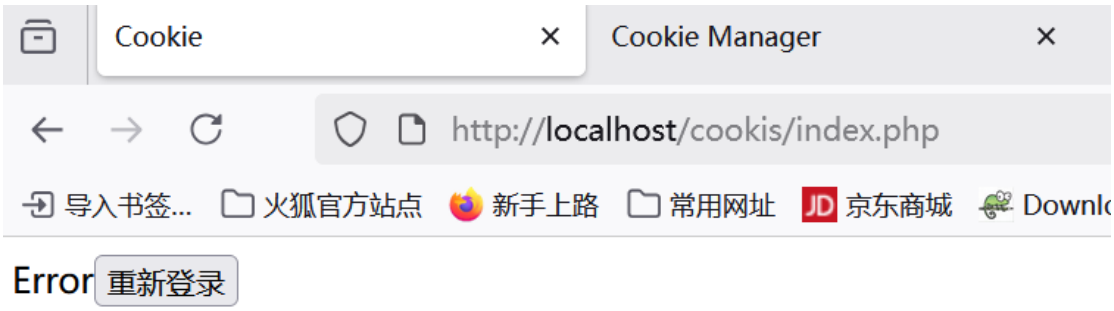
```php
<?php
$dsn = 'mysql:dbname=lab;host=127.0.0.1';
$user = 'root';
$password = '123456';

try {
    $pdo = new PDO($dsn, $user, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die('数据库连接失败: '. $e->getMessage());
}
```

分离 sql 逻辑与数据，绑定参数，即使 Cookie 值被篡改，也会被当作普通字符串。

```php
// 使用 PDO 参数化查询
$sql = "SELECT * FROM users WHERE username = :username AND passcode = :passwd";
$stmt = $pdo->prepare($sql);
$stmt->bindParam(':username', $username, PDO::PARAM_STR);
$stmt->bindParam(':passwd', $passwd, PDO::PARAM_STR);
$stmt->execute();

$row = $stmt->fetch(PDO::FETCH_ASSOC);
```

```php
if ($row) {
    echo '欢迎用户：' . $row['username'];
    echo '<br>';

    $timestamp = time() + $exipre_time;
    setcookie('account', $cookee, $timestamp);

    $format = 'D d M Y - H:i:s';
    date_default_timezone_set('PRC');
    echo 'Cookie 到期日期: ' . date($format, $timestamp);
    echo '<button type="button" onclick="clearCookies()">清除Cookie</button>';
    echo '<br>';
    echo ' Your Password: ' . $row['passcode'];
} else {
    echo 'Error';
    echo '<button type="button" onclick="clearCookies()">重新登录</button>';
}
```

进行 cookie 注入攻击测试，欢迎页面变成 error，并要求重新登录，即注入失败。



2. 利用 Cookie 注入还可以实现哪些攻击？

攻击者可直接利用获取到的 Cookie 冒充用户身份，无需密码即可登录用户账户，进而执行转账、发布内容等操作，完全控制用户账号。

通过注入恶意 SQL 指令，攻击者可删除或修改数据库中的数据。

攻击者可结合 XSS，利用 Cookie 注入恶意脚本，进一步窃取用户浏览器中的其他 Cookie 信息，或实施 XSS 外带攻击。

攻击者可通过注入修改 Cookie 中的权限字段，绕过权限校验，获取更高权限。

注入大量恶意数据或构造特殊的 Cookie 请求，可导致服务器处理异常，消耗过多资源，最终引发拒绝服务，使正常用户无法访问系统。

3. HTTP 头部注入攻击除了利用 User-Agent 的内容，还有哪些内容可以利用？

Referer 记录了用户访问当前页面的来源页面地址。

Cookie 在客户端和服务器之间传递状态信息。

服务器根据 Host 头部来确定请求的目标站点。

X - Forwarded - For 记录经过代理服务器时的客户端真实 IP 地址。

4. 如何使用 PDO 参数化插入的方式防止 http 头部注入攻击

首先创建与数据库的连接之后，编写预处理 sql 语句，使用占位符代替 useragent，数据库会把传入的参数当作普通数据处理，而不是 SQL 代码的一部分，即使攻击者在 HTTP 头部中插入恶意的 SQL 代码，也只会被当作字符串存储，不会被数据库解析执行。

```
$insert = "INSERT INTO uagents (username, uagent, ip_address) VALUES (:username, :uagent, :ip_address)";
$stmtInsert = $pdo->prepare($insert);
$stmtInsert->bindParam(':username', $username, PDO::PARAM_STR);
$stmtInsert->bindParam(':uagent', $uagent, PDO::PARAM_STR);
$stmtInsert->bindParam(':ip_address', $IP, PDO::PARAM_STR);
$stmtInsert->execute();
```

进行测试，注入失败。