《安全程序设计与实践》
实验报告

实验名称：万能密码登录—Post 型注入攻击

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：2025/4/23

一、 实验目的

1. 能够解释和分析万能密码登录漏洞的 Post 型 SQL 注入原理及危害，并应用多种方式实现 SQL 注入攻击的防护

二、 实验过程

任务一：建立用户信息数据库。

首先进入 mysql 数据库，通过执行 source 命令，将 SQL 脚本导入，完成数据库创建与数据初始化。

该 SQL 脚本逻辑为：先判断是否存在 lab 数据库，若存在则删除，重新创建 lab 库并切换至该库；接着创建 users 表，定义 id（自增主键）、username、passcode 字段；最后向表中插入 admin/admin123、alice/alice456 两条用户数据 。

导入后用 select * from users 查询，验证数据成功插入。

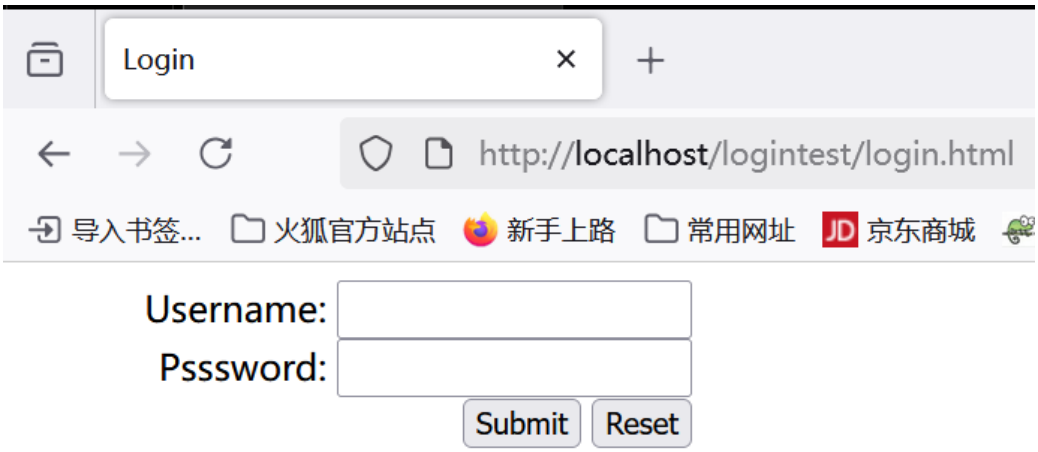
```
mysql> source C:\Users\Administrator\Desktop\Web\apache2.4\httpd-2.4.33-o102o-x86-vc14-r2\Apache24\htd
ocs\logintest\lab.sql
Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 1 row affected (0.01 sec)

Database changed
Query OK, 0 rows affected (0.03 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

mysql> select * from users;
+----+----------+----------+
| id | username | passcode |
+----+----------+----------+
|  1 | admin    | admin123 |
|  2 | alice    | alice456 |
+----+----------+----------+
2 rows in set (0.00 sec)
```

任务二：开发基于 Session 验证的用户登录功能网站，使用 Post 方式提交用户参数。

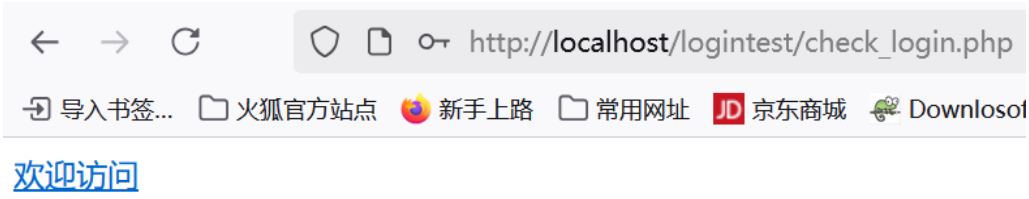
修改 login.html 中的登录表单，通过 form 标签设置 method="post" 、action="check_login.php" ，实现用户名、密码的 Post 方式提交。

```
12      <div id=a>
13          <form name="form_login" method="post" action="check_login.php">
14              Username: <input type="text" class=b name="username" /> <br>
15              Psssword: <input type="password"  class=b name="passwd" /> <br>
16              <input type="submit" name="Submit" value="Submit" />
17              <input type="reset" name="Reset" value="Reset" />
18          </form>
19      </div>
```

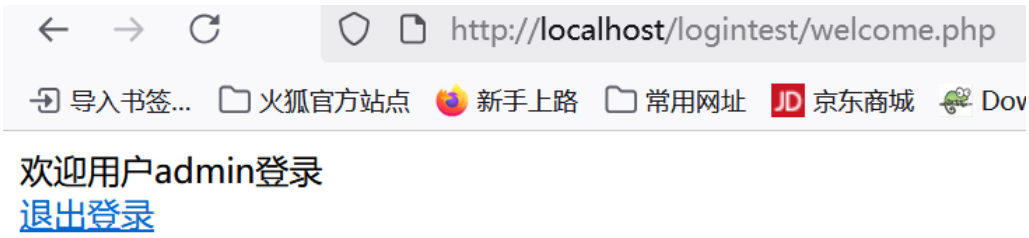
访问 login.html 后，输入账号密码（admin/admin123）提交。



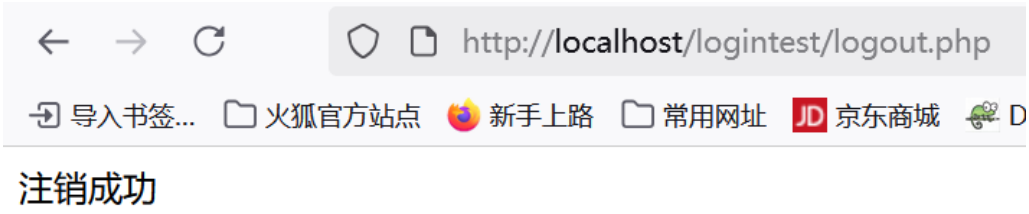
请求发送至 check_login.php 验证，与数据库信息一致，登录成功。



跳转 welcome.php ，利用 Session 标识用户身份，显示 “欢迎用户 admin 登录”



点击 “退出登录”，访问 logout.php 销毁 Session ，提示 “注销成功” 。

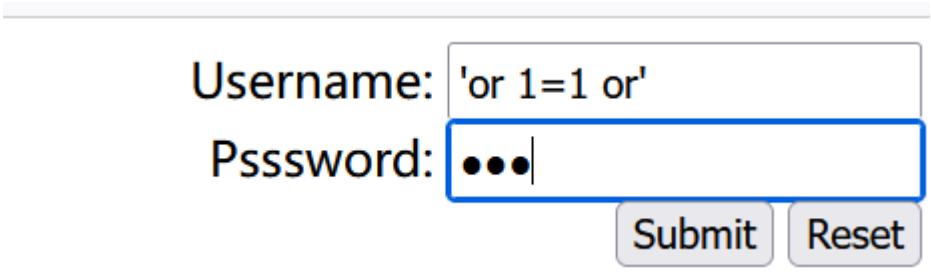


实现了基于 Session 的用户登录、验证、登出功能。

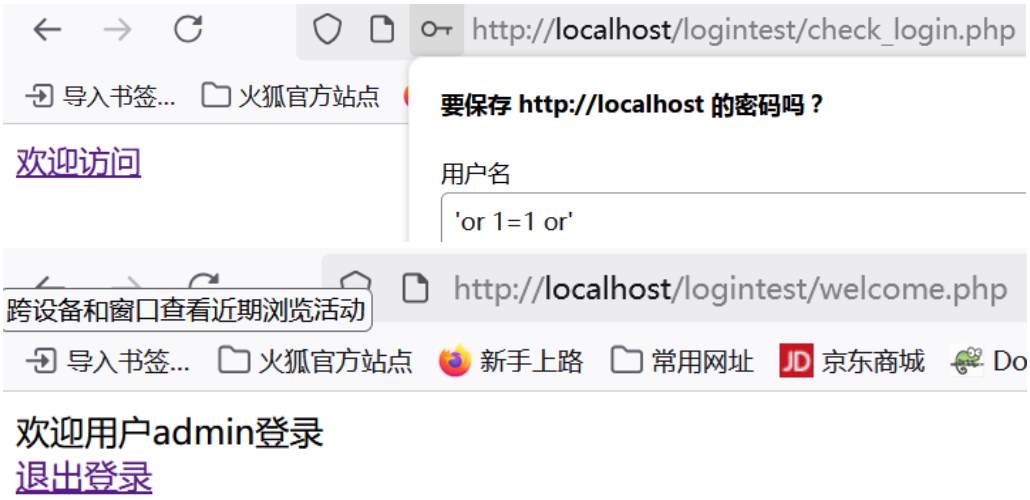
任务三：实现基于 SQL 注入的万能密码登录。

访问 login.html，尝试万能密码登录。

在登录表单中，用户名输入 'or 1=1 or'语句，由于数据库后台未严格过滤输入，让后台拼接的 SQL 查询（SELECT * FROM users WHERE username =’’or 1=1 or’’ AND passcode = '123'）恒成立。



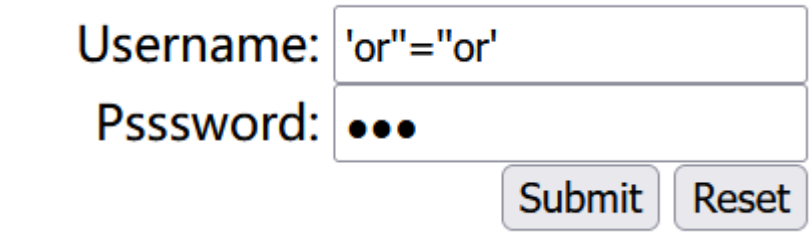
提交后，成功绕过验证跳转到 welcome.php ，模拟以 admin 身份登录（因万能密码常匹配数据库首位用户，一般第一位用户都是 admin）



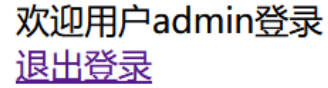
再测试使用其他万能密码进行登录。

测试'or"='or'。

SELECT * FROM users WHERE username =’’or’’=’or’’ AND passcode =’123’恒成立。



登录成功



任务四：分析万能密码 SQL 注入原理，实现多种防护方式。

4-1 正则表达式：

先从**输入格式校验**入手，利用正则表达式限制用户名、密码的合法字符与格式，阻

断恶意注入语句的输入。

本实验中我们限制 username 输入为英文字母、数字和下划线，且长度在 5-16 字符之间，避免单引号引起万能密码绕过。通过正则表达式 "/^[a-zA-Z0-9_]{5,16}$/"限制。

```
if (!preg_match("/^[a-zA-Z0-9_]{5,16}$/",$username))
{
    echo "<script>alert('用户名输入格式错误！')</script>";
    exit;
}
```

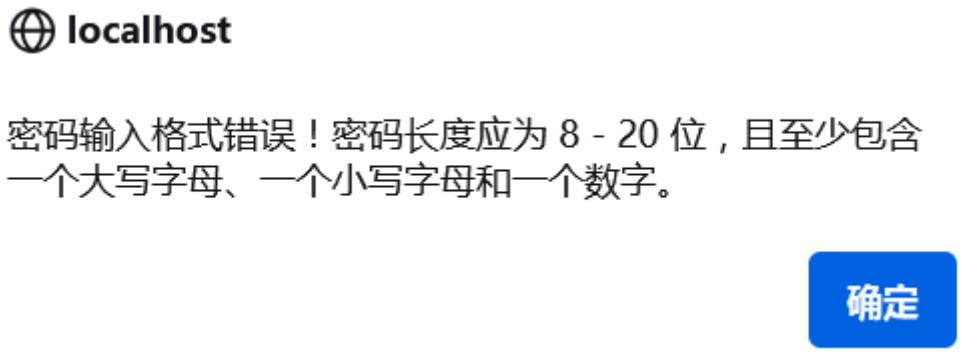
此时我们输入万能密码'or 1=1 or'，因包含单引号、非法字符，正则匹配失败，触发“用户名输入格式错误”提示，直接阻断注入尝试。

此规则通过限制字符集，避免用户输入含 SQL 逻辑的特殊符号（如单引号、等号 ），从源头减少注入风险。

我们尝试对密码进行更严格的正则校验：要求密码长度为 8 - 20 位，且至少包含一个大写字母、一个小写字母和一个数字。正则表达式为 "/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d@$!%*?&]{8,20}$/"。

```
if (!preg_match("/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d!@#$%^&*()_+=-]{8,20}$/", $passwd)) {
    echo "<script>alert('密码输入格式错误！密码长度应为 8 - 20 位，且至少包含一个大写字母、一个小写字母和一
    exit;
```

使用 admin 登录，密码 123，触发“密码输入格式错误”提示。此规则强制密码复杂度，既提升账户安全性，也间接增加注入难度。



4-2 php 安全函数

PHP 提供了专门的安全函数，用于处理用户输入中的特殊字符，阻断 SQL 注入的拼接逻辑。

addslashes()：在预定义字符（`'`、`\`、`"`）前加反斜杠，确保这些字符被视为数据，而非 SQL 代码。

```
$username = isset($_POST['username']) ? addslashes($_POST['username']) : '';
$passwd = isset($_POST['passwd']) ? addslashes($_POST['passwd']) : '';
```

当输入含单引号的万能密码（'or 1=1 or' ）进行测试时，addslashes()会将单引号转义为“\'”，使拼接后的 SQL 语句无法形成注入逻辑——SQL 查询会按“字符串匹配”执行，而非逻辑恒成立，最终触发“登录失败”（因实际无此用户名）。

mysql_escape_string()：MySQLi 扩展提供的转义函数，需传入数据库连接对象和待转义字符串。它会根据当前数据库的字符集，对特殊字符进行转义，比 addslashes() 更贴合 MySQL 数据库的安全需求。

```
$username = isset($_POST['username']) ? mysqli_escape_string($con,$_POST['username']) : '';
$passwd = isset($_POST['passwd']) ? mysqli_escape_string($con, $_POST['passwd']) : '';
```

输入含单引号的万能密码（'or 1=1 or' ）进行测试，与 addslashes()类似，mysql_escape_string()会转义用户输入中的 SQL 特殊字符，确保输入被当作数据存入查询，使 SQL 查询按正常字符串匹配执行，无法绕过验证。

4-3 mysqli 参数化查询

参数化查询是防御 SQL 注入的核心技术，通过“预编译 SQL 语句+绑定参数”方式，彻底分离用户输入与 SQL 逻辑，从语法层面杜绝注入风险。

先将 login.html 中 action 更改为 check_login_mysqli.php，确保请求发送到新的处理脚本。

```
<form name="form_login" method="post" action="check_login_mysqli.php">
```

查看 check_login_mysqli.php 内容，发现先定义含占位符（?）的 SQL 模板，由数据库提前编译，确定查询结构。

再将用户输入的两个字符串类型变量 username 和 passwd 作为参数，绑定到占位符上，数据库会将其视为纯数据，而非可执行的 SQL 代码。

最后将预编译的语句+绑定的参数共同执行，确保输入无法改变 SQL 逻辑。

```php
$sql = "SELECT * FROM users WHERE username = ? and passcode = ?";

$stmt = $con->prepare($sql);
if (!$stmt) {
    echo 'prepare 执行错误';
}
else{
    $stmt->bind_param("ss",$username, $passwd);
    $stmt->execute();

    $result = $stmt->get_result();
    $row = $result->fetch_row();
    if($row)
    {
        session_start();
        $_SESSION['username'] = $row[1];
        echo $row[1]." <a href='welcome.php'>欢迎访问</a>";
    }else{
        echo "<script>alert('登录失败!'); history.go(-1);</script>";
    }
    $stmt->close();
}
```

尝试输入万能密码（'or 1=1 or'）时，参数化查询会将其视为普通字符串，精确匹配 username 和 passwd 执行查询。因此登录失败。

4-4 PDO 参数化查询

PDO 是 PHP 中更具通用性的数据库抽象层，其参数化查询在防御 SQL 注入同时，具备跨数据库适配优势，为多数据库场景下的安全开发提供支持。

PDO 核心优势：数据库无关性，可以通过统一的 API 操作不同关系型数据库。

先将 login.html 中 action 更改为 check_login_pdo.php，确保请求发送到新的处理脚本。

```
<form name="form_login" method="post" action="check_login_pdo.php">
```

check_login_pdo.php 的内容如下：

- 预编译阶段确定 SQL 结构（WHERE username = :username AND passcode = :passwd ），用户输入无法篡改查询逻辑。
- 绑定参数时，PDO 自动处理特殊字符转义（适配当前数据库字符集），确保输入仅作为字符串参与匹配。

因此输入万能密码（'or 1=1 or'）后，页面提示“登录失败”，证明 PDO 参数化查询成功拦截注入，只有正确的用户名和密码能通过验证。

三、 实验结论

本次实验通过建立用户信息数据库、开发基于 Session 验证的 Post 型用户登录功能网站，成功实现了基于 SQL 注入的万能密码登录，直观验证了此类漏洞的危害。

实验中构造的万能密码，利用未过滤的用户输入拼接 SQL 语句，通过注入永真条件绕过身份验证。分析其原理可知，当用户名或密码参数未做安全处理时，攻击者输入的恶意字符会破坏 SQL 语句逻辑，导致非预期查询。

在防护措施实现上，正则表达式限制输入字符类型和格式，可有效阻止含单引号等恶意字符的输入；Addslashes()、mysql_escape_string()等安全函数通过转义特殊字符，破坏注入语句的构造；而 mysqli 和 PDO 的参数化查询技术，将用户输入作为数据而非 SQL 代码处理，从根本上防止了注入攻击，验证了参数化查询在跨数据库场景下的通用性和高效性。

四、 思考题

1. 如果执行数据库的查询语句 $sql="select * from users where username='".$username."' and passcode='".$passwd."'";，该如何构造万能密码？
   由于点号起连接作用，因此可以分解为
   select * from users where username='"+$username+"' and passcode='"+$passwd+"'"。
   由于双引号是定义字符串的，那么去掉双引号后实际查询语句为
   select * from users where username='admin' and passcode='admin123'"
   可以构造' or '1'='1 万能密码。

2. 你还了解哪些查询语句的形式？如何构造对应的万能密码？
   - 多条件 OR 查询：
     SELECT * FROM users WHERE username = '$username' OR role = '$role';
     ' OR '1'='1 --，构造真语句并注释掉后面的内容。
   - LIKE 模糊查询：
     SELECT * FROM products WHERE name LIKE '%$input%';
     输入'%' OR '1'='1 --，会变成 SELECT * FROM products WHERE name LIKE '%%' OR '1'='1' --%';
     %是通配符，能匹配任意数量字符，然后构造表达式为真（'1'='1'），则整个条件表达式为真，并且将后面的内容注释掉，最终会返回 products 表中的所有记录。