

Skill Horizon Internship Project

Vulnerable Web Application Module - Security Analysis

Project Overview

This project demonstrates SQL injection vulnerabilities in web applications and their security fixes. It includes both vulnerable and secure versions of a PHP login page.

Submitted By- Prashant Sengar

Files Included

vulnerable_login.php - Login page with SQL injection vulnerability

secure_login.php - Fixed version with proper security measures

setup_database.php - Database setup script

README.md - Complete documentation

Prerequisites & Setup Requirements

System Requirements

Operating System: Windows 10/11, macOS, or Linux

Web Server: XAMPP

PHP Version: PHP 7.4 or higher

Database: MySQL/MariaDB

Web Browser: Chrome, Firefox, Safari, or Edge

Software Installation

--> XAMPP Installation

Download XAMPP from <https://www.apachefriends.org/>

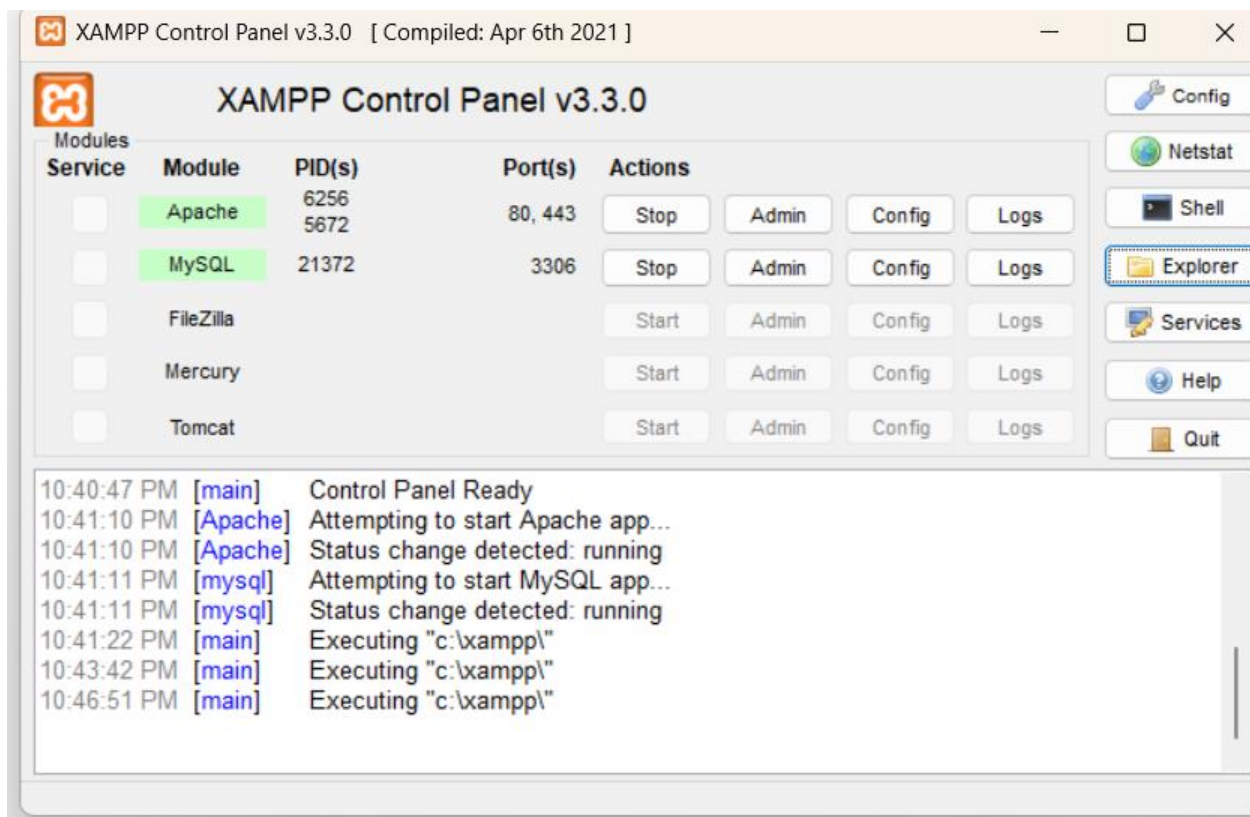
Install XAMPP with default settings

Start Services:

Apache (Web Server)

MySQL (Database Server)

Verify Installation: Open <http://localhost>



Step 1: Start XAMPP Services

2: Open XAMPP Control Panel

3: Start Apache service

4: Start MySQL service

5: Verify both services are running (green status)

6:Step 2: File Placement

7:Copy all PHP files to C:\xampp\htdocs\ folder

Files required:

1: vulnerable_login.php

2: secure_login.php

3: setup_database.php

4: README.md

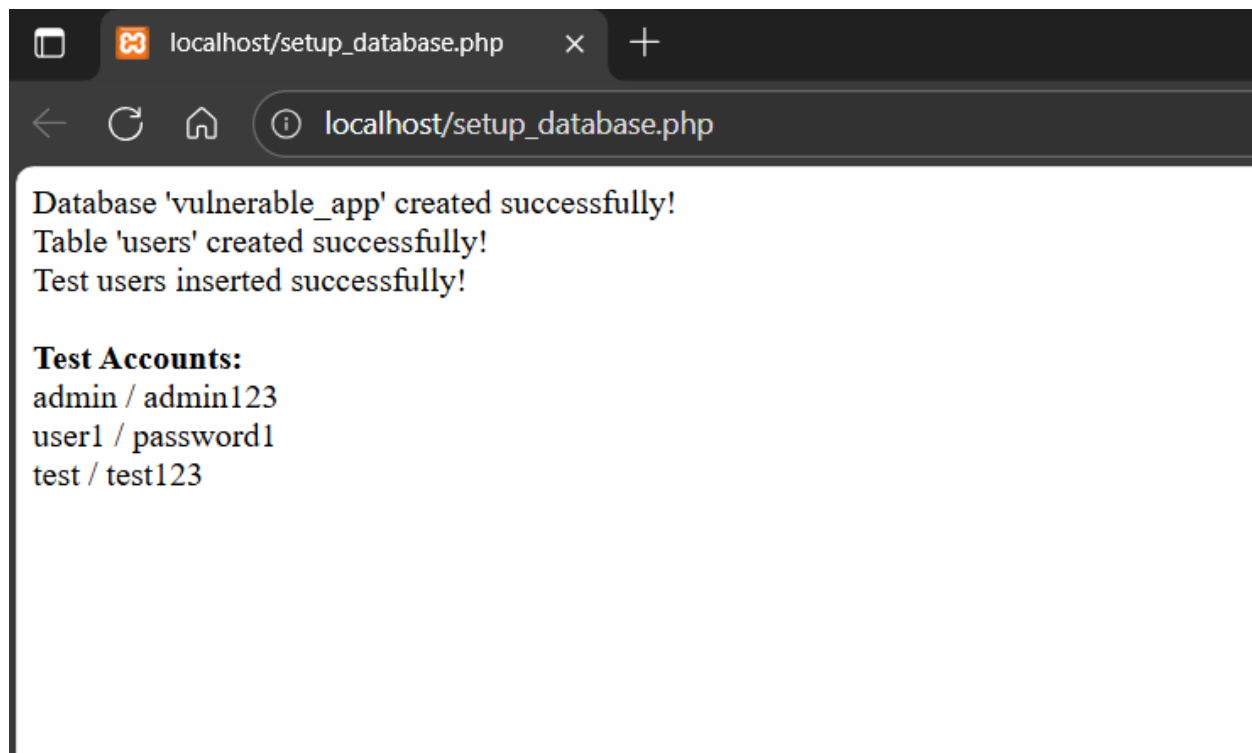
Step : Database Setup

Open browser

Navigate to: http://localhost/setup_database.php

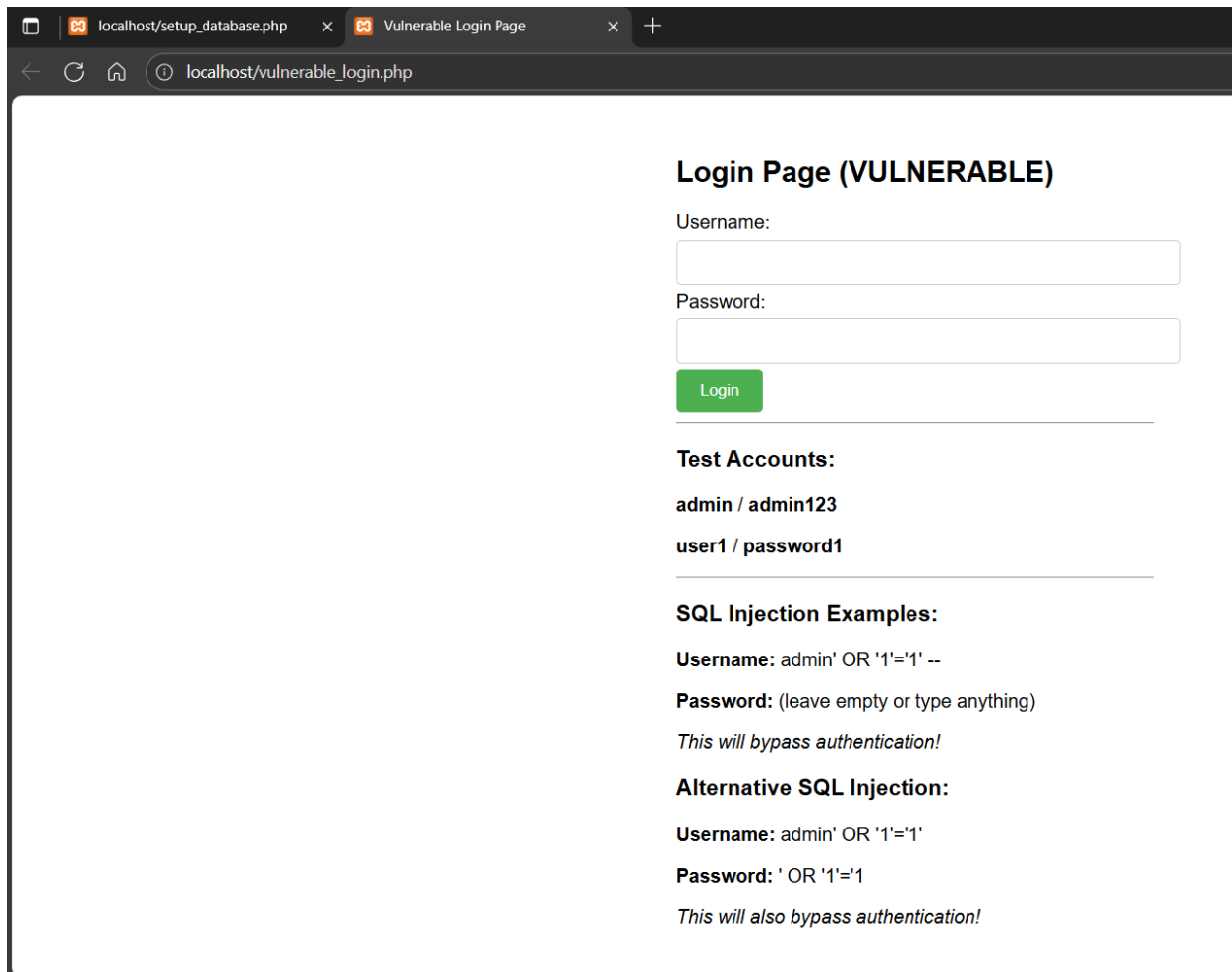
Verify database creation success message

Test accounts will be created automatically



Step : Testing

- 1: Vulnerable Version: http://localhost/vulnerable_login.php
- 2: Secure Version: http://localhost/secure_login.php
- 3: Test SQL Injection: Use provided attack payloads



--> Login as normal user with legitimate given password

Login Page (VULNERABLE)

Executing query: SELECT * FROM users WHERE
username = 'user1' AND password = 'password1'

Login successful! Welcome user1

User ID: 2

Email: user1@example.com

Username:

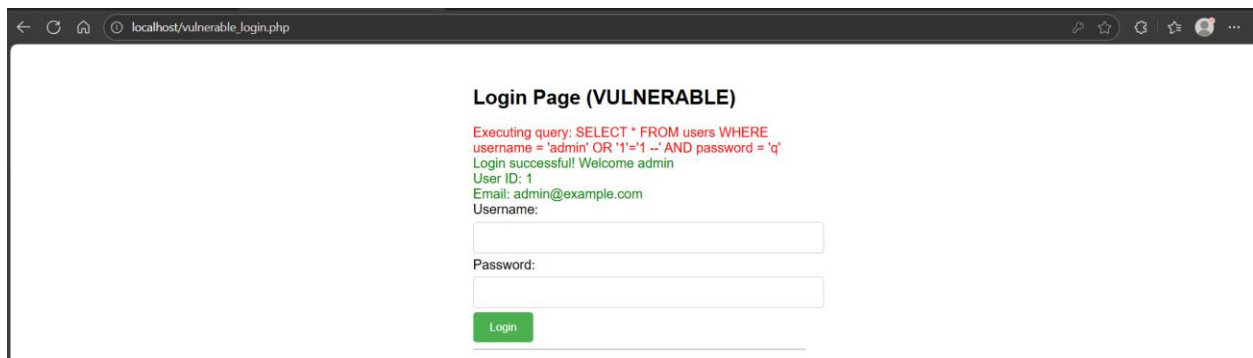
Password:



Login

Login successful

--> By passing login page via SQL injection



--> HERE payload used

Username: admin' Or'1'='1 --

Password: q

BY PASSED LOGIN SUCCESSFULLY

**Vulnerable Sql code : \$sql = "SELECT * FROM users WHERE username =
'\$user' AND password = '\$pass'";**

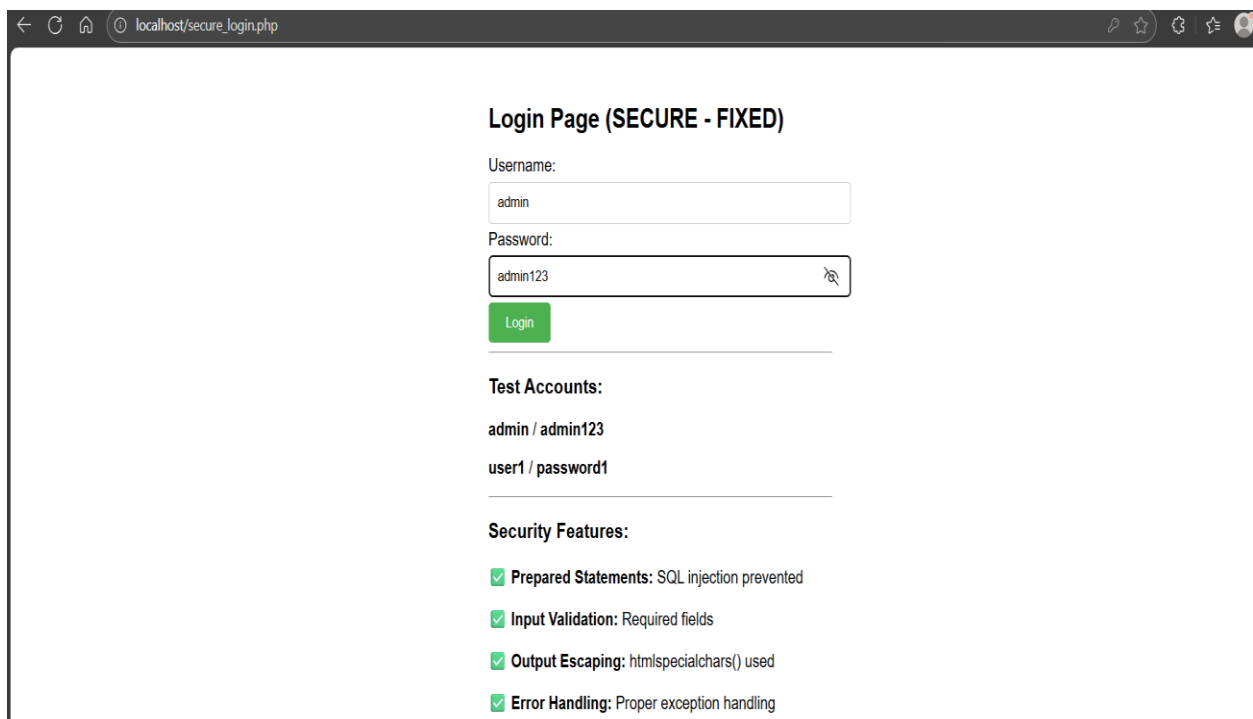
Attack Vector:

Username: admin' OR '1'='1' --

Password: anything OR empty

This bypasses authentication by manipulating SQL query logic

Secure Version (secure_login.php)



The screenshot shows a web browser window with the address bar displaying 'localhost/secure_login.php'. The page content is as follows:

Login Page (SECURE - FIXED)

Username:

Password:

Test Accounts:

admin / admin123

user1 / password1

Security Features:

- ✓ **Prepared Statements:** SQL injection prevented
- ✓ **Input Validation:** Required fields
- ✓ **Output Escaping:** htmlspecialchars() used
- ✓ **Error Handling:** Proper exception handling

Login Page (SECURE - FIXED)

Using prepared statement with parameters

Invalid username or password!

Username:

Password:

Login

Test Accounts:

--> Here the payload doesn't work

Security Measures:

1: Prepared statements with parameterized queries

2: Input validation for required fields

3: Output escaping using htmlspecialchars()

4: Proper error handling

5: No direct string concatenation

--> Secure Code :

```
$sql = "SELECT * FROM users WHERE username = ? AND password = ?";
```

```
$stmt = $pdo->prepare($sql);
```

```
$stmt->execute([$user, $pass]);
```

Security Comparison

--> Security Comparison

Aspect	Vulnerable Version	Secure Version
SQL Queries	String concatenation	Prepared statements
Input Validation	None	Required field validation
Output Escaping	Basic	htmlspecialchars()
Error Handling	Exposes errors	Proper exception handling
SQL Injection	Vulnerable	Protected

--> Test Results

1: Vulnerable version: SQL injection attack successful

2; Secure version: SQL injection attack blocked

3: Database setup: Successfully created with test accounts

--> Security Best Practices Implemented

1: Use prepared statements for all database queries

2: Validate and sanitize all user inputs

3: Escape output to prevent XSS attacks

4: Implement proper error handling

5: Use least privilege principle for database access

--> Conclusion

This module effectively demonstrates the critical importance of secure coding practices. SQL injection remains one of the most common web vulnerabilities but can be easily prevented with proper coding techniques like prepared statements and input validation.