Patching SQL Injection Vulnerability in DVWA

# Introduction

SQL Injection is one of the most dangerous and common web application vulnerabilities. In this document, we will explain how to patch a basic SQL Injection vulnerability found in the DVWA (Damn Vulnerable Web Application) under the SQL Injection (Low) module.

# Problem Overview

The 'User ID' input in DVWA's SQL Injection module allows direct injection of user-controlled data into SQL queries.

Vulnerable SQL Query:

SELECT first\_name, last\_name FROM users WHERE id = '$id';

Exploit Payload:

1 OR 1=1 -- -

# Patch Strategy

To mitigate SQL Injection, developers should adopt secure coding practices and defense-in-depth strategies as explained below.

## 1. Use Prepared Statements / Parameterized Queries

Prepared statements separate the SQL logic from user input. This prevents malicious input from being interpreted as SQL commands.

Secure PHP Example (PDO):

$id = $\_GET['id'];  
$stmt = $pdo->prepare("SELECT first\_name, last\_name FROM users WHERE id = ?");  
$stmt->execute([$id]);  
$data = $stmt->fetchAll();

Secure PHP Example (MySQLi):

$id = $\_GET['id'];  
$stmt = $conn->prepare("SELECT first\_name, last\_name FROM users WHERE id = ?");  
$stmt->bind\_param("i", $id);  
$stmt->execute();  
$result = $stmt->get\_result();

## 2. Input Validation and Sanitization

Always validate user input to ensure it matches expected data types. For example, only integers should be accepted for ID fields.

PHP Example:

$id = filter\_input(INPUT\_GET, 'id', FILTER\_VALIDATE\_INT);

## 3. Use Stored Procedures

Stored procedures keep SQL logic on the database side, minimizing exposure of dynamic queries.

Example:

CREATE PROCEDURE getUserById(IN userId INT)  
BEGIN  
 SELECT first\_name, last\_name FROM users WHERE id = userId;  
END;

## Principle of Least Privilege

Ensure the database user account only has permissions it needs.

- Don't use root or admin-level accounts.  
- Only allow SELECT access where necessary.

## Implement a Web Application Firewall (WAF)

Use tools like ModSecurity, AWS WAF, or Cloudflare to detect and block malicious SQL patterns at the application perimeter.

# Summary

|  |  |
| --- | --- |
| Vulnerability | Solution |
| Raw SQL Queries | Use Prepared Statements |
| No Input Filtering | Validate and Sanitize Input |
| Overprivileged Accounts | Enforce Least Privilege |
| Direct SQL Access | Use Stored Procedures |
| No Application Filtering | Add a Web Application Firewall (WAF) |

# Conclusion

SQL Injection vulnerabilities can compromise an entire database if left unpatched. By implementing secure coding practices, database-level protection, and application firewalls, developers can ensure their applications are resistant to these attacks.  
  
Securing user inputs and avoiding dynamic SQL queries is not just a best practice—it's essential for safeguarding user data and maintaining application integrity.