Week 5: Ethical Hacking Report

Abdullah Akram
ID: DHC-3606

Task 5: Ethical Hacking & Exploiting Vulnerabilities

July 23, 2025

Cybersecurity Internship Report

Submitted as part of Week 5 deliverables

1 Objective

The objective was to perform ethical hacking on the home-rental PHP web application, identify vulnerabilities using tools like SQLMap, Burp Suite, and Kali Linux, and implement mitigation techniques to secure the application against SQL injection, CSRF, and other vulnerabilities.

2 Tools Used

- **Kali Linux**: Penetration testing environment for reconnaissance and exploitation.
- **SQLMap**: Automated tool for SQL injection testing.
- **Burp Suite**: Web application security testing for CSRF and session management.
- **Nmap**: Network scanner for port and service enumeration.

3 Process

1. **Reconnaissance**: Used Nmap to scan localhost for open ports and services.
2. **Vulnerability Scanning**: Tested login and registration forms for SQL injection using SQLMap.
3. **CSRF Testing**: Intercepted form submissions using Burp Suite to identify CSRF vulnerabilities.
4. **Mitigation**: Implemented CSRF tokens, verified prepared statements, and added input validation.
5. **Validation**: Re-tested the application to ensure vulnerabilities were resolved.

4 Code Implementation

4.1 SQL Injection Protection (login.php)

- **Action**: Ensured all database queries use PDO prepared statements to prevent SQL injection.
- **Code**:

```
<?php
require_once 'config.php';
$username = $_POST['username'];
$password = $_POST['password'];
$stmt = $pdo->prepare("SELECT␣*␣FROM␣users␣WHERE␣username␣=␣?");
$stmt->execute([$username]);
$user = $stmt->fetch();
if ($user && password_verify($password, $user['password'])) {
    session_start();
```

```
    $_SESSION['user_id'] = $user['id'];
    header("Location: ../index.php");
} else {
    echo "Invalid credentials";
}
?>
```

4.2 CSRF Protection (config.php and login.php)

- **Action**: Added CSRF token generation and validation to prevent unauthorized form submissions.
- **Code (config.php)**:

```
<?php
session_start();
if (!isset($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}
function getCsrfToken() {
    return $_SESSION['csrf_token'];
}
function validateCsrfToken($token) {
    return hash_equals($_SESSION['csrf_token'], $token);
}
$pdo = new PDO("mysql:host=localhost;dbname=home_rental", "root", "");
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
?>
```

- **Code (login.php Form)**:

```
<form method="POST" action="login.php">
    <input type="hidden" name="csrf_token" value="<?php echo getCsrfToken(); ?>">
    <input type="text" name="username" placeholder="Username" required>
    <input type="password" name="password" placeholder="Password" required>
    <input type="submit" value="Login">
</form>
```

- **Code (login.php Validation)**:

```
<?php
require_once 'config.php';
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (!validateCsrfToken($_POST['csrf_token'])) {
        die("Invalid CSRF token");
    }
    // Proceed with login logic
```

```
}
?>
```

4.3 Input Validation (register.php)

- **Action**: Added server-side validation to ensure secure input handling.
- **Code**:

```
<?php
require_once 'config.php';
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (!validateCsrfToken($_POST['csrf_token'])) {
        die("Invalid CSRF token");
    }
    $username = filter_var($_POST['username'],
       FILTER_SANITIZE_STRING);
    $email = filter_var($_POST['email'], FILTER_SANITIZE_EMAIL);
    if (strlen($username) < 3 || strlen($_POST['password']) < 8)
        {
        die("Username or password too short");
    }
    $password = password_hash($_POST['password'],
       PASSWORD_DEFAULT);
    $stmt = $pdo->prepare("INSERT INTO users (username, email,
       password) VALUES (?, ?, ?)");
    $stmt->execute([$username, $email, $password]);
    header("Location: login.php");
}
?>
```

5 Findings

- **SQL Injection**: No vulnerabilities found due to PDO prepared statements in login.php and register.php.
- **CSRF**: Initial lack of CSRF tokens made forms vulnerable. Mitigated by adding token generation and validation.
- **Input Validation**: Added to prevent malicious inputs and ensure data integrity.
- **Screenshots**:

ot of Nmap scan results

rming no vulnerabilities

ability before mitigation

6 Conclusion

The ethical hacking process identified and mitigated CSRF vulnerabilities by implementing tokens and validated the robustness of SQL injection protections. Input validation was added to enhance security. All changes were committed to the GitHub repository at `https://github.com/abdullah-akram/home-rental`.