Farah Anany 19103787

## Vulnerabilities Displayed in the Code

1. Identification and Authentication Failure (7) and Cryptographic Failure (2)

The password should be hashed and not stored in plain text to protect the data. Also, there is no Regex that enable strong password policies.

A screenshot of a computer

Description automatically generated

A login screen with blue text

Description automatically generated

Vulnerability: if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {

$user\_id = $\_POST['user\_id'];

$password = $\_POST['password'];

$sql = "SELECT \* FROM users WHERE user\_id='$user\_id' AND password='$password'";

$result = $conn->query($sql);

if ($result->num\_rows > 0) {

$\_SESSION['user\_id'] = $user['user\_id'];

$\_SESSION['role'] = $user['role'];

header("Location: home.php");

} else {

echo "Invalid ID or Password!";

}

}

Sample Mitigation(HASHING): $password = password\_hash($\_POST['password'], PASSWORD\_BCRYPT);

1. Security Misconfiguration (5)

Displaying errors back to the user can give the attacker information about the system.

Vulnerability: error\_reporting(E\_ALL); ini\_set('display\_errors', 1);

Sample Mitigation: ini\_set('display\_errors', 0);

1. Identification and Authentication Failure (7)

The session’s ID doesn’t change. Session management should be intact.

Vulnerability: session\_start();

Sample Mitigation: session\_regenerate\_id(true);

1. SQL Injection(3)

The search item in the admin page can allow the user to query which is very dangerous and makes user input anything from deleting to dumping data.

Vulnerability: if ($\_SERVER["REQUEST\_METHOD"] == "POST" && isset($\_POST['search'])) {

$search\_term = $\_POST['search\_term'];

$search\_type = $\_POST['search\_type'];

if ($search\_type === 'subject') {

$sql\_search\_subject = "SELECT u.user\_id, u.first\_name, u.last\_name, e.grade

FROM users u

JOIN enrollments e ON u.user\_id = e.user\_id

WHERE e.course\_id = '$search\_term'";

$search\_result = $conn->query($sql\_search\_subject);

} elseif ($search\_type === 'user\_id') {

$sql\_search\_user = "SELECT \* FROM users WHERE user\_id = '$search\_term'";

$search\_result = $conn->query($sql\_search\_user);

}

}

Sample Mitigation: $search\_term = $conn->real\_escape\_string($\_POST['search\_term']);This excludes any special characters to prevent SQL injections like OR '1'='1'.

1. Cross-Site Scripting (XSS) (Injection (3))

The username is sent back when the web application says Welcome, {user\_id}.

This can be extremely dangerous if the user\_id is a malicious code.

Vulnerability: <h2>Welcome, <?php echo $user\_id; ?></h2>

Sample Mitigation: Enable Regex or exclude certain characters.

1. Broken Access Control(1)

Viewing the course Id doesn’t have any form of access control. A checking of whether the person is supposed to see the details of course must occur

$course\_id = $\_GET['course\_id'];

Sample Mitigation:

session\_start();

if (!isset($\_SESSION['user\_id'])) {

header('Location: login.php');

exit();

}

$course\_id = $\_GET['course\_id'];

$user\_id = $\_SESSION['user\_id'];

// Ensure the user has permission to view the course

$query = "SELECT \* FROM courses WHERE course\_id = ? AND user\_id = ?";

$stmt = $conn->prepare($query);

$stmt->bind\_param("ii", $course\_id, $user\_id);

$stmt->execute();

$result = $stmt->get\_result();

if ($result->num\_rows == 0) {

echo 'You do not have permission to view this course.';

exit();

}