Cybersecurity Internship Report

Intern Name: Ehmaan Shafqat

Project Title: Strengthening Security Measures for a Web Application

Submitted to: Faizan Khan

Date: 26 June, 2025

Week 5: Ethical Hacking & Exploiting Vulnerabilities

Project Overview

Goal:

- Perform ethical hacking on a test web application.
- Identify and exploit vulnerabilities (SQLi, CSRF).
- Implement security patches to mitigate risks.

Tools Used:

- **Kali Linux** (Penetration Testing Toolkit)
- **SQLMap** (Automated SQL Injection Detection)
- **Burp Suite** (CSRF Testing)
- **Node.js + csurf** (CSRF Protection)

Task 1: Ethical Hacking Basics

Conducted Reconnaissance

Tool: Kali Linux (`nmap`, `dirb`, `nikto`)

Target: Test Web Application (`http://testapp.local`)

Findings:

1. **Open Ports:**
   ```
   nmap -sV testapp.local
   ```
   - **Port 80 (HTTP)** – Apache 2.4.29
   - **Port 3306 (MySQL)** – Exposed (Potential SQLi risk)
2. **Directory Enumeration:**
   ```
   dirb http://testapp.local
   ```
   - `/admin` – Exposed admin panel (No brute-force protection)

- `/login` – Vulnerable to credential stuffing

3. **Nikto Scan:**

```
nikto -h http://testapp.local
```

- **Outdated Software** (Apache 2.4.29 has known CVEs)

Task 2: SQL Injection (SQLi) Exploitation & Fix

Exploitation with SQLMap

Command:

```
sqlmap -u "http://testapp.local/login?user=admin&pass=123" --risk=3 --level=5
```

Vulnerabilities Found:

- **Boolean-Based SQLi** (Bypass login using `admin' -- `)
- **Error-Based SQLi** (Extracted DB name: `app_db`)

Mitigation: Prepared Statements

Before (Vulnerable Code):

```
app.post('/login', (req, res) => {
  const { user, pass } = req.body;
  db.query(`SELECT * FROM users WHERE username='${user}' AND password='${pass}'`);
});
```

After (Fixed Code):

```
app.post('/login', (req, res) => {
  const { user, pass } = req.body;
  db.query('SELECT * FROM users WHERE username=? AND password=?', [user, pass]);
});
```

Verification:

- Retested with SQLMap → **No SQLi detected.**

Task 3: CSRF Exploitation & Protection

CSRF Attack Simulation (Burp Suite)

1. Captured a legitimate POST request:

```
POST /change-email HTTP/1.1
Host: testapp.local
Cookie: sessionid=xyz123
Content-Type: application/x-www-form-urlencoded
new_email=attacker@evil.com
```

2. Created a malicious HTML page:

```
<form action="http://testapp.local/change-email" method="POST">
  <input type="hidden" name="new_email" value="hacker@evil.com">
</form>
<script>document.forms[0].submit();</script>
```

3. **Result:** CSRF attack successful (email changed without user consent).

Mitigation: CSRF Tokens (csurf)

Implementation:

```
const csrf = require('csurf');
const csrfProtection = csrf({ cookie: true });

app.post('/change-email', csrfProtection, (req, res) => {
  // Requires valid CSRF token
});
```

Frontend Integration:

```
<form action="/change-email" method="POST">
  <input type="hidden" name="_csrf" value="{{csrfToken}}">
  <input type="email" name="new_email">
</form>
```

Verification:

- Burp Suite retest → **CSRF blocked (403 Forbidden).**

Deliverables Checklist

Task	Status	Details
Reconnaissance	Done	Found exposed admin panel & MySQL.
SQL Injection Exploit	Done	Extracted DB name via SQLMap.
SQLi Fix	Done	Implemented prepared statements.
CSRF Exploit	Done	Changed email via malicious HTML.
CSRF Protection	Done	Added `csurf` middleware.

Updated GitHub Repository

Files Modified:

```
├── app.js                # Added CSRF & SQLi fixes
├── /security-testing/
│   ├── sqlmap_scan.txt   # SQLMap results
│   ├── csrf_poc.html     # Malicious CSRF test file
└── README.md             # Updated security docs
```

Key Commits:

- `feat: Add CSRF protection using csurf`
- `fix: Replace raw SQL queries with prepared statements`

1. **Critical Vulnerabilities Found:**
   - Unfiltered user input → **SQLi**.
   - Missing anti-CSRF tokens → **Account takeover risk**.
2. **Improvements Made:**
   - **Parameterized queries** for SQLi prevention.
   - **CSRF tokens** for state-changing requests.

Conclusion

This project successfully:

Exposed critical vulnerabilities (SQLi, CSRF) in a test environment.

Patched security flaws using industry-standard practices.

Documented ethical hacking process for future reference.