Web Application Security Assessment

Internship Task: Security Assessment of Web Application

Intern Name: Rabia Khawaja

Date: 22nd April 2025

-----------------------------------------------------------------------------------------

Week 5: **Ethical Hacking & Exploiting Vulnerabilities**

**Goal**:Learn Ethical Hacking Techniques, exploit vulnerabilities in a test environment, and enhance security.

Tasks:

1. **Ethical Hacking basics:**

**Use Kali Linux (or any penetration testing tool)**

**Perform reconnaissance on a test web application.**

**What we need to do**

Open your **Kali Linux terminal**.

Type this to check your IP:

Ifconfig

If you have a local web app running (like your Node.js login system), note its port (e.g., localhost:3000).

Write this command

nmap -sV -p 3000 127.0.0.1

Use nikto to find vulnerabilities

nikto -h http://3000 127.0.0.1

1. **SQL Injection & Exploitation:  
   Use SQLMAP to identify SQL injection vulnerabilities**

**Apply prepared statements to mitigate SQL injections.**

**What we need to do**

SQL Injection (SQLI) is when someone tricks a website into running bad SQL code. We have to find and fix these.

· Make sure your Node app has an input field vulnerable to SQL injection.

· Run your server:

In Terminal write

sqlmap -u "http://127.0.0.1:3000/login?username=test&password=test" --batch --risk=3 --level=5

**Now moving towards Prepared Statements.**

When a user inputs something like:

username = ' OR 1=1 --

A **vulnerable SQL query** might become:

SELECT \* FROM users WHERE username = '' OR 1=1 --' AND password = ''

**This tricks the database into logging in anyone, even without valid credentials.**

**Prepared statements fix this by treating user inputs as data, NOT code.  
So even if someone writes an injection like ' OR 1=1, it’ll be stored as input — not executed.**

**What we need to do,**

### ****1. Install SQLite3 for Node.js****

Run in terminal inside your project folder:

npm install sqlite3

### ****2. Create a New Database File****

In your project folder, create a file called:

users.db

It’ll store our login data.

### ****3. Add This to Your**** server.js ****(Updated with Prepared Statements)****

Replace your old /login route with this secure version:

const express = require('express');const winston = require('winston');const sqlite3 = require('sqlite3').verbose();

const app = express();const PORT = process.env.PORT || 3000;

// Logger setupconst logger = winston.createLogger({

level: 'info',

format: winston.format.combine(

winston.format.timestamp(),

winston.format.printf(({ timestamp, level, message }) => {

return `${timestamp} [${level.toUpperCase()}]: ${message}`;

})

),

transports: [

new winston.transports.Console(),

new winston.transports.File({ filename: 'security.log' })

],

});

// Database connectionconst db = new sqlite3.Database('./users.db', (err) => {

if (err) {

return console.error(err.message);

}

console.log('Connected to SQLite database');

});

// Create table if not exists

db.run(`

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

username TEXT,

password TEXT

)

`);

// 🛡️ Secure login with prepared statement

app.get('/login', (req, res) => {

const { username, password } = req.query;

const sql = 'SELECT \* FROM users WHERE username = ? AND password = ?';

db.get(sql, [username, password], (err, row) => {

if (err) {

logger.error(`DB Error: ${err.message}`);

return res.status(500).send('Internal Server Error');

}

if (row) {

logger.info(`Successful login for user: ${username}`);

res.send('Login successful');

} else {

logger.warn(`Failed login attempt for user: ${username}`);

res.send(' Invalid credentials');

}

});

});

// Start server

app.listen(PORT, () => {

logger.info(`🚀 Server running on port ${PORT}`);

});

### ****4. Add a Test User to the Database****

**Temporarily add this route in your file (then delete it later):**

app.get('/add-user', (req, res) => {

const { username, password } = req.query;

const sql = 'INSERT INTO users (username, password) VALUES '

db.run(sql, [username, password], (err) => {

if (err) {

return res.send('Error adding user');

}

res.send('User added');

});

});

Then go to browser:

http://127.0.0.1:3000/add-user?username=test&password=test

### ****5. Test Login (Normal)****

Now write

http://127.0.0.1:3000/login?username=test&password=test

You should see:

Login successful

### ****6. Test Login (Injection Attempt)****

Now try this malicious one:

http://127.0.0.1:3000/login?username=test' OR '1'='1&password=test

Because of **prepared statements**, it will fail :

Invalid credentials

1. **Cross-Site Request Forgery (CSRF) Protection:**

**Implement CSRF tokens using csurf in Node.js**

**Test CSRF vulnerabilities using Burp Suite.**

**What we need to do**

**CSRF** is being tricked into clicking a hidden button while your session is still active.

#### What To Do:

* Install csurf:

npm install csurf

Add this to your server.js:

const csrf = require('csurf');const cookieParser = require('cookie-parser');

app.use(cookieParser());

app.use(csrf({ cookie: true }));

cookieParser() This allows your app to **read cookies** sent by the browser.

csrf({ cookie: true }) This sets up CSRF protection using a **token stored in a cookie**.

Every request that changes data (like POST, PUT, DELETE) now requires a valid CSRF token.

**Now Create a route in your server to send token.**

**This will allow frontend to ask the backend for the CSRF token, and then use it in forms or API calls.  
The route will be**

app.get('/csrf-token', (req, res) => {

res.json({ csrfToken: req.csrfToken() });

});

**\*Testin with Burp Suite to make sure only real users of your site can perform actions.**

· Your server **gives** the browser a secret token. The browser **sends it back** on important actions. If it’s missing or wrong = request gets rejected.

· **Open your app** in a browser and **login** (if needed).

· Open **Burp Suite**, enable the **proxy**, and intercept a request to a protected route (like a form submission).

· In the **Intercept** tab, look for the CSRF token (it might be in headers, cookies, or request body).

· **Modify it** (e.g. change 1 letter or remove it entirely).

· **Forward** the request..