

Name: Hajra Zafar
DHC-Id: 490

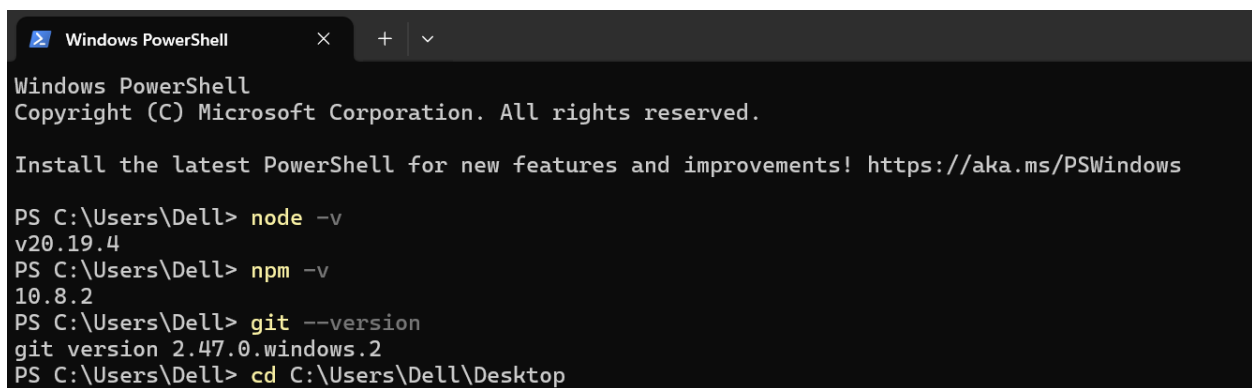
WEEK 1:

Security Assessment

OWASP Zap:

Automated scan performed using OWASP ZAP. Report attached in repo under /zap-report

Understand the Application:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Dell> node -v
v20.19.4
PS C:\Users\Dell> npm -v
10.8.2
PS C:\Users\Dell> git --version
git version 2.47.0.windows.2
PS C:\Users\Dell> cd C:\Users\Dell\Desktop
```

Built a Simple App:

```

PS C:\Users\Dell\Desktop\user-management> npm init -y
Wrote to C:\Users\Dell\Desktop\user-management\package.json:

{
  "name": "user-management",
  "version": "1.0.0",
  "description": "## Setup ``bash npm init -y npm install express body-parser ejs node index.js ``",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

PS C:\Users\Dell\Desktop\user-management> npm install express body-parser ejs

added 75 packages, and audited 76 packages in 46s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\Dell\Desktop\user-management> node index.js
App running at http://localhost:3000

```

Progress so far:

1. **Environment Setup Done**
 - At first I checked **Node.js, npm, Git** → all installed correctly.
2. **Created a Lightweight Test App**
 - Instead of using heavy apps like Juice Shop (which gave errors), we built a **simple User Management App** (signup & login).
 - Extracted it → Installed dependencies (express, body-parser, ejs).
3. **App is Running**
 - i started it with node index.js.
 - Confirmed output:
 - App running at <http://localhost:3000>
 - That means the backend server is live.
4. **Currently at:**
 - Visiting <http://localhost:3000/> shows Cannot GET / → normal, because we didn't make a homepage.
 - The **working routes** are:
 - /signup → for registration
 - /login → for login

Where We stand in Internship Tasks:

- we now have a **mock vulnerable app** running on localhost.

- Next steps are to **perform attacks/tests** (XSS, SQLi, headers, weak password storage)

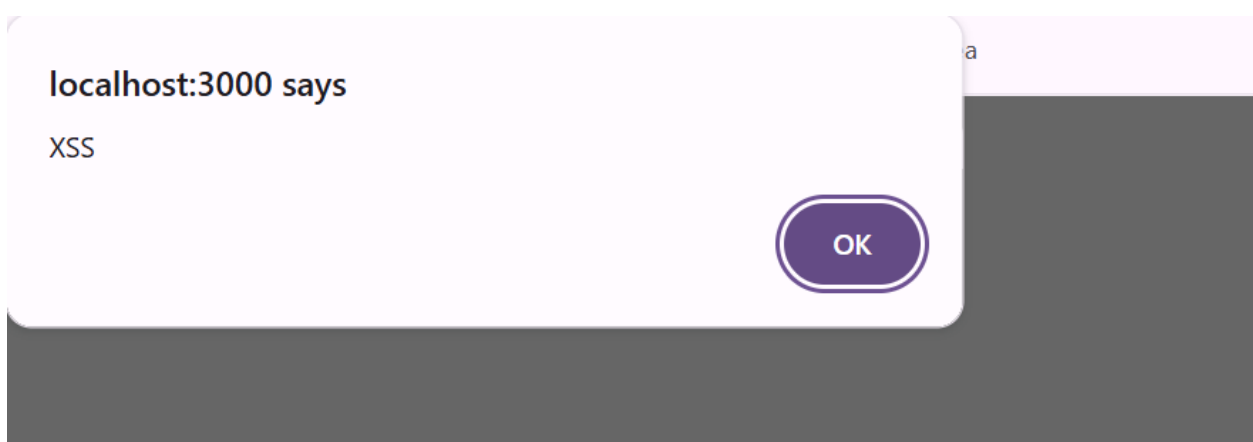
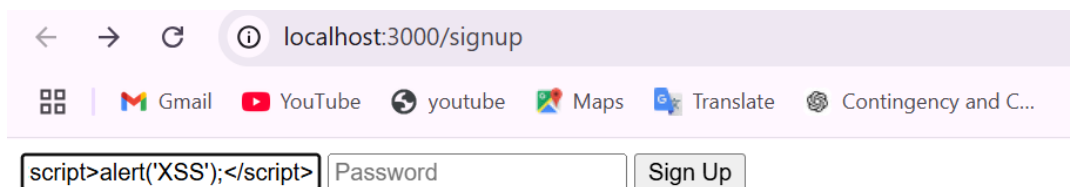
Perform Basic Vulnerability Assessment

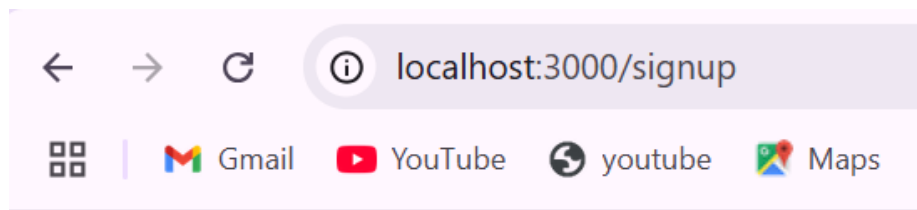
I. XSS Test (Cross-Site Scripting)

- Go to <http://localhost:3000/signup>
- In username field enter:

Enter :

```
<script>alert('XSS');</script>
```



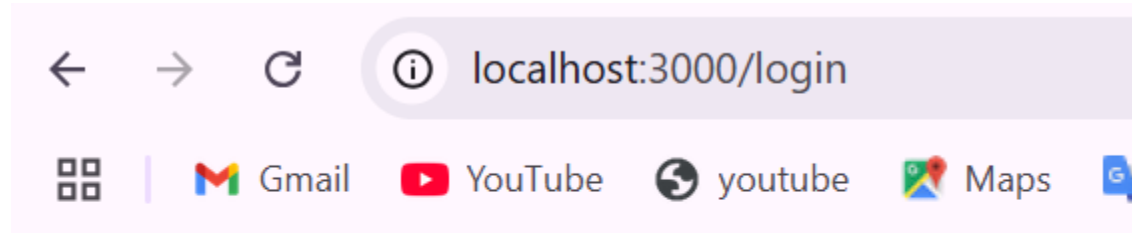


User registered:

2. SQL Injection Test :

SQL Injection Risk

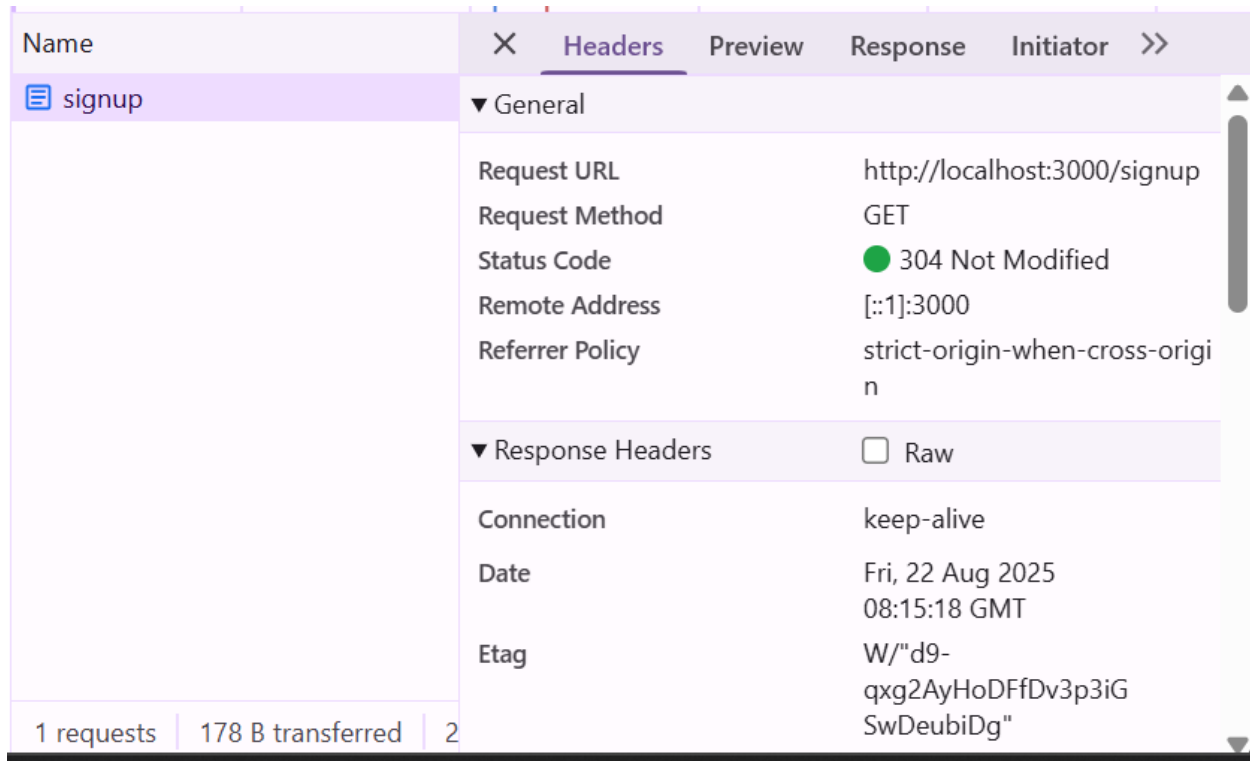
Tried payload admin' OR '1'='1 on login form. Authentication bypass did not succeed because no SQL database is connected. However, since user inputs are directly passed into authentication logic without sanitization, the application would be vulnerable if backed by SQL. Recommendation: always sanitize inputs and use parameterized queries.



Invalid credentials

3.Security Misconfiguration evidence:

Click f12 → then network tab → ctrl R to reload → All → header → signup click → header → response header



Name	Headers	Preview	Response	Initiator	>>
signup	<div><div>▼ General</div><div>Request URL: http://localhost:3000/signup Request Method: GET Status Code: 304 Not Modified Remote Address: [::1]:3000 Referrer Policy: strict-origin-when-cross-origin</div><div>▼ Response Headers <input type="checkbox"/> Raw</div><div>Connection: keep-alive Date: Fri, 22 Aug 2025 08:15:18 GMT Etag: W/"d9-qxg2AyHoDFfDv3p3iGSwDeubiDg"</div></div>				

1 requests | 178 B transferred | 2

▼ Response Headers ☒ Raw

```
HTTP/1.1 304 Not Modified
X-Powered-By: Express
ETag: W/"d9-qxg2AyHoDFfDv3p3iGSwDeubiDg"
Date: Fri, 22 Aug 2025 08:15:18 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

Risk: The app does not include important headers like Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options, or X-Frame-Options. Without these, the app is vulnerable to XSS, clickjacking, MIME-sniffing, and downgrade attacks.

Recommendation: Use the helmet middleware in Express to automatically add these headers:

```
const helmet = require("helmet");
```

```
app.use(helmet());
```

4 .Weak Password Storage

- Open the index.js file.
- We'll Look at the signup route code.

That proves passwords are stored in **plaintext** → which is a vulnerability.

```
// Signup page
app.get("/signup", (req, res) => {
  res.render("signup");
});

app.post("/signup", (req, res) => {
  users.push({ username: req.body.username, password: req.body.password });
  res.send("User registered: " + req.body.username);
});
```

Recommendations:

- Sanitize/validate inputs
- Use bcrypt for password hashing
- Add Helmet for security headers
- Validate forms with validator.js

What does sanitize/validate inputs mean?

When a user types something into a form (username, email, password, etc.), app should **not trust it directly**.

- **Validation** = Check that the input is in the correct format.
 - Example: email must look like name@email.com
 - Example: password must be at least 8 characters long
- **Sanitization** = Clean the input so it can't break the system or inject malicious code.
 - Example: if someone types `<script>alert('XSS')</script>` → the app should **strip out** `<script>` before saving/using it.

Solution → Hash passwords with bcrypt

bcrypt is a library that converts a plaintext password into a secure hash before storing it.

Example:

```
const bcrypt = require("bcrypt");

// In signup route
const hashedPassword = await bcrypt.hash(req.body.password, 10); // 10 = salt rounds
users.push({ username: req.body.username, password: hashedPassword });
```

Now, instead of storing "mypassword123", our database will store something like:

```
$2b$10$D9hVZZ8i8xBtCak5eMlslO4mftLdf..sa57G1uXbbTxMBQld6M6dW
```

This is useless to an attacker without cracking.

What is Helmet?

Helmet is a Node.js middleware for Express apps.

It **adds important HTTP security headers automatically** to protect our app from common attacks.

By default, our app is missing headers like:

- Content-Security-Policy → stops injection of malicious scripts.
- X-Content-Type-Options → stops MIME-type sniffing.
- X-Frame-Options → prevents clickjacking.
- Strict-Transport-Security → enforces HTTPS.

We saw this in **misconfiguration findings** → those headers were absent.

Validate forms with validator.js

“Use validator library for input validation”, it means:

- Reject scripts / SQL-like input.

- Accept only proper values (like a real email, or strong password).

Conclusion:

The Week 1 assessment successfully identified multiple security issues in the User Management System, including Cross-Site Scripting (XSS), missing security headers, and weak password storage. While the SQL Injection attempt did not succeed due to the absence of a database, unsanitized inputs pose a significant risk in a real-world scenario. Implementing the recommended fixes such as input validation, bcrypt hashing, and Helmet.js will significantly strengthen the application's security posture.