**Web Application Security Report**

**Project Name:** Basic Login System

**GitHub Repository:** Basic Login System

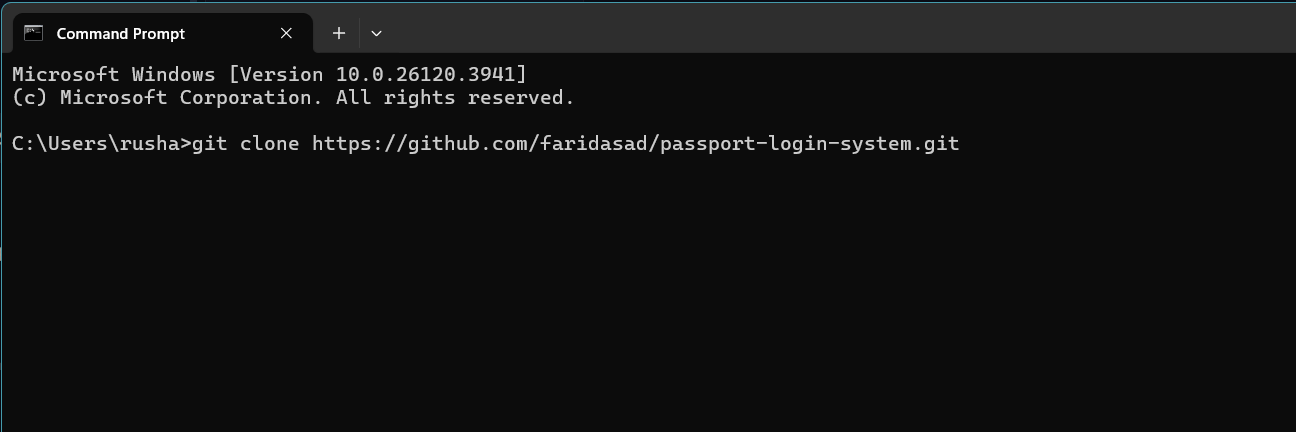
**Author:** Rushali

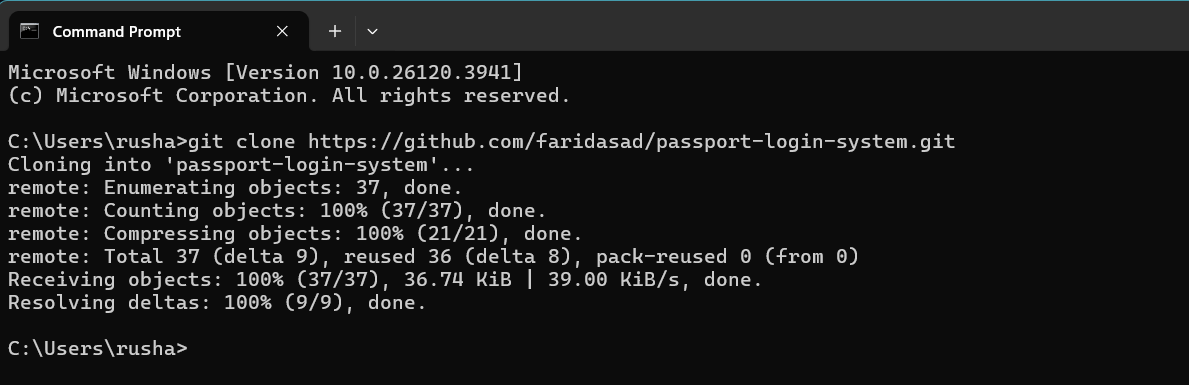
**Date:** *5-May’25*

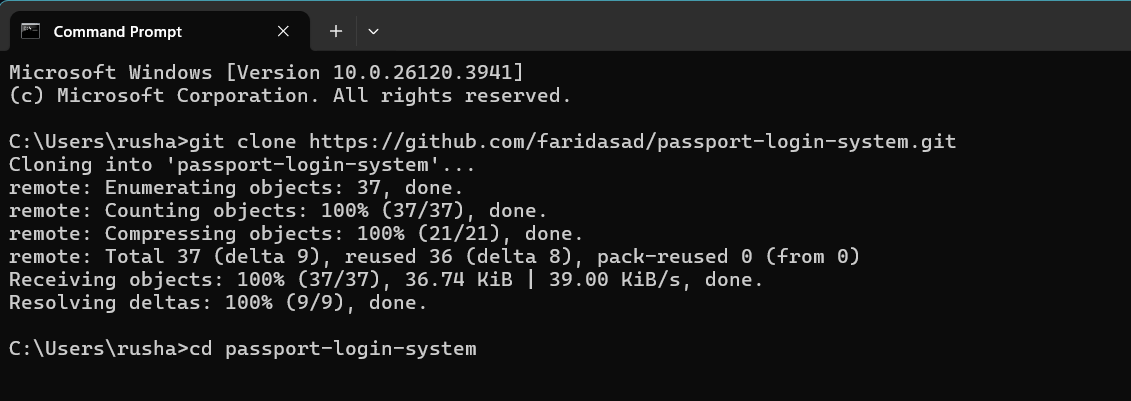
**Week 1: Security Assessment**

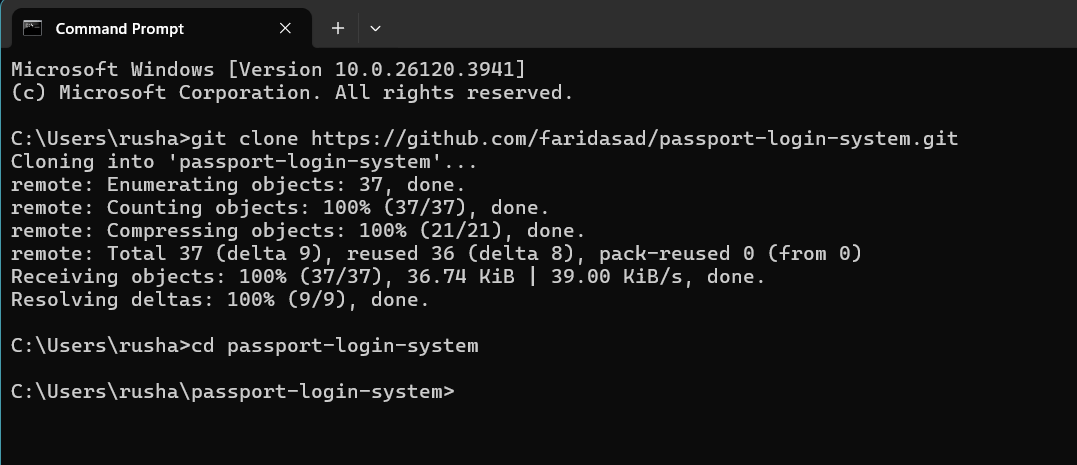
**Understanding the Application**

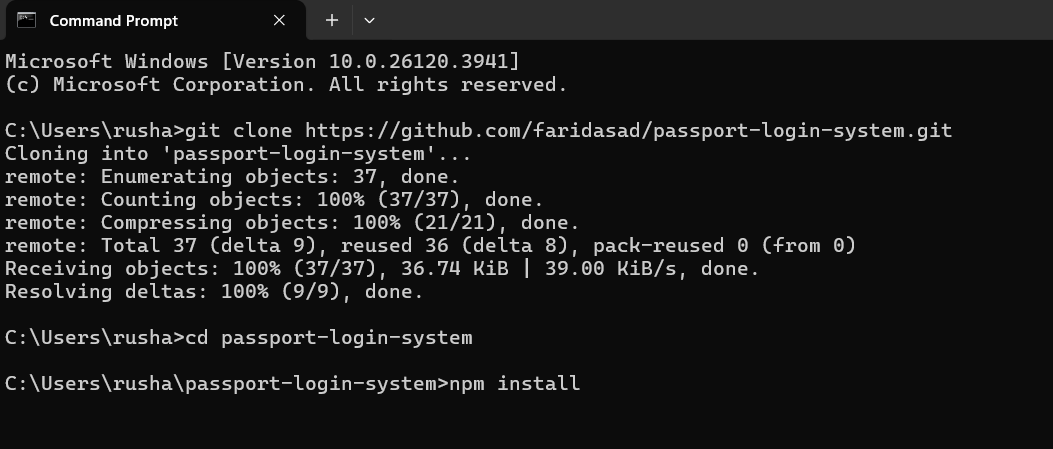
During the first week, a **basic Node.js login system** was selected from GitHub and cloned into the local environment for analysis. Dependencies were installed using:

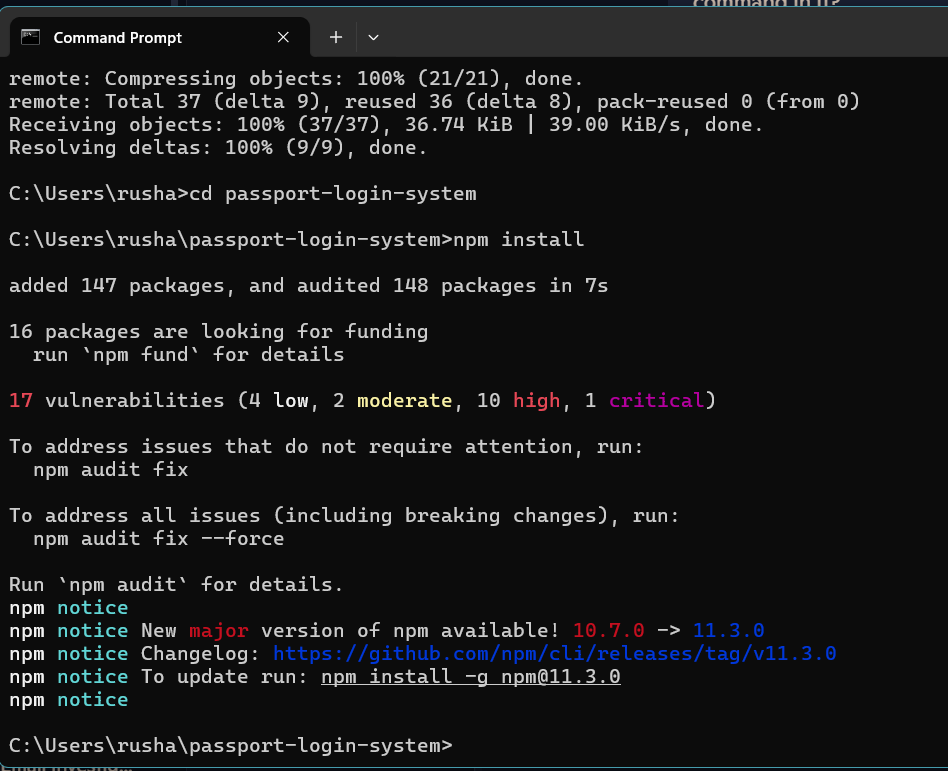


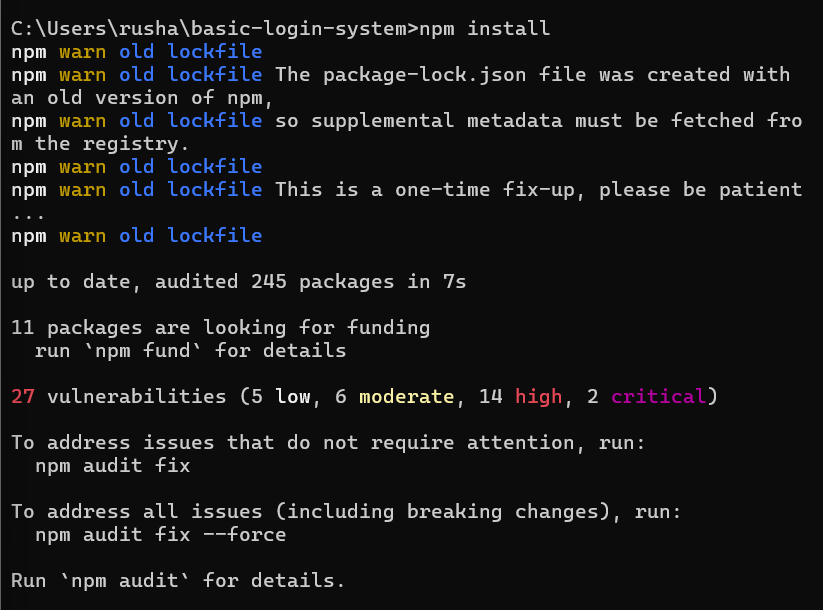


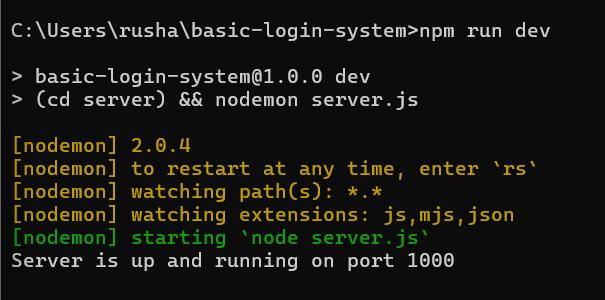


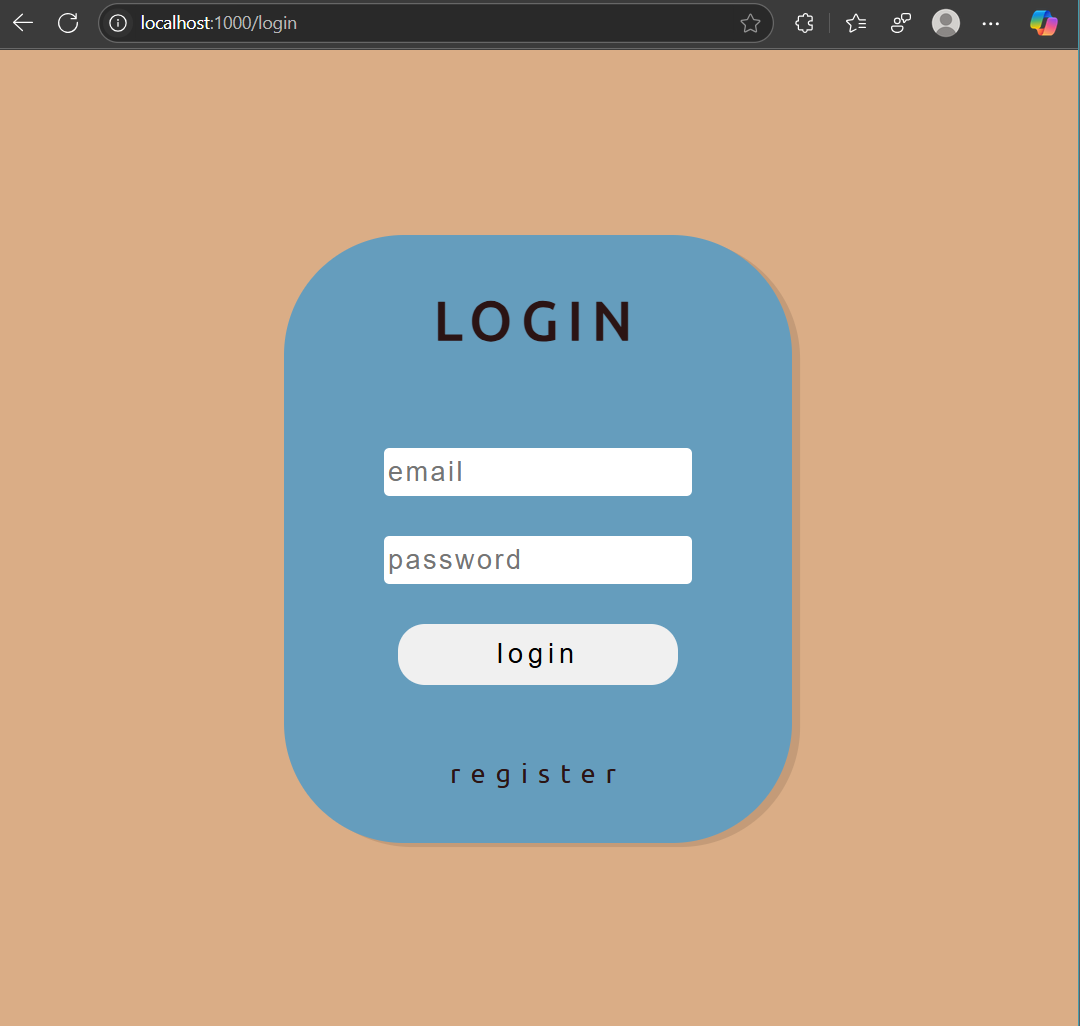






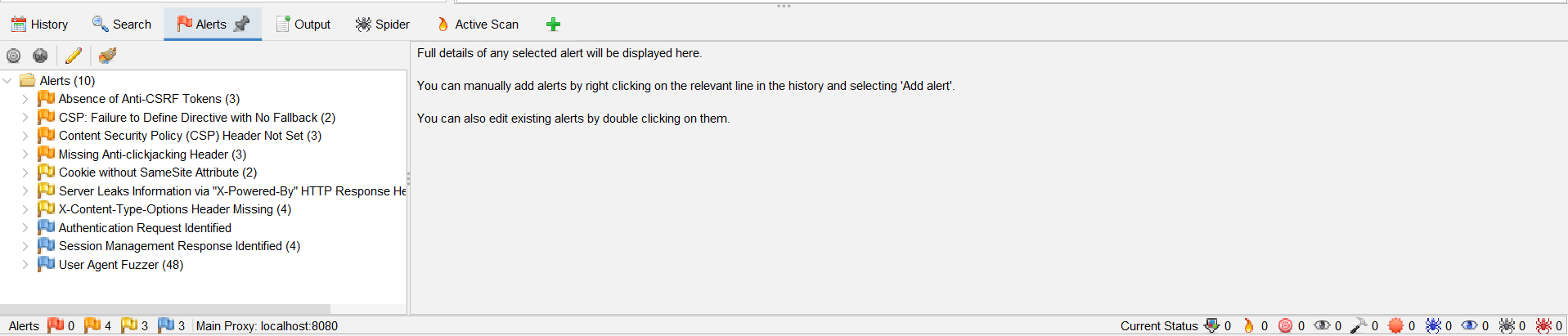


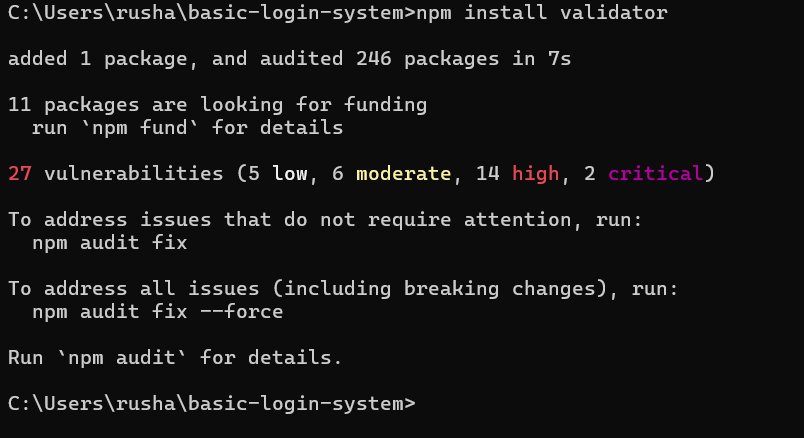


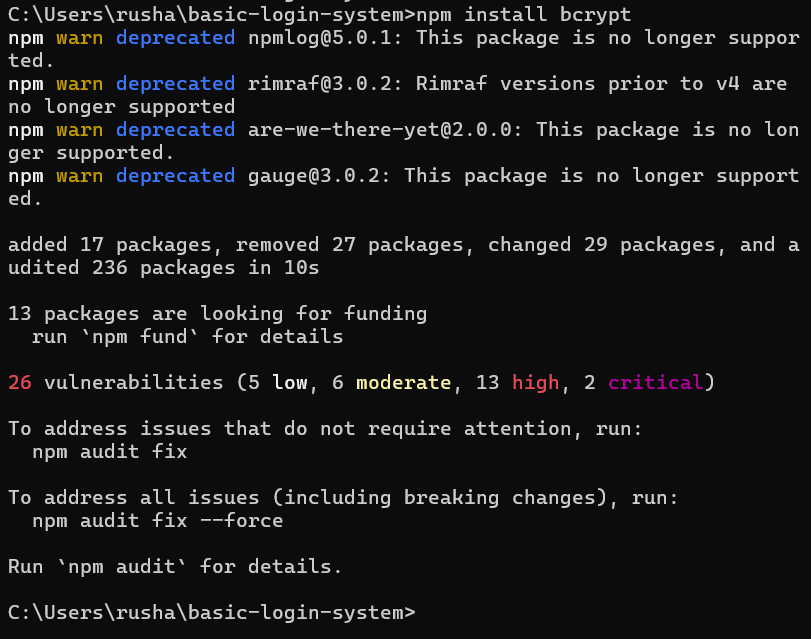


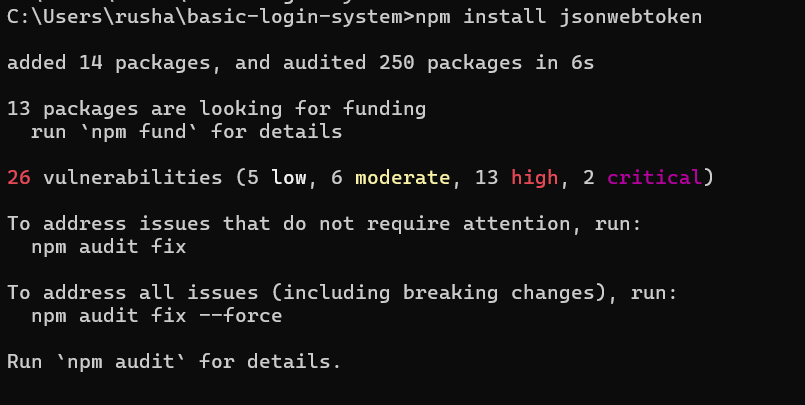


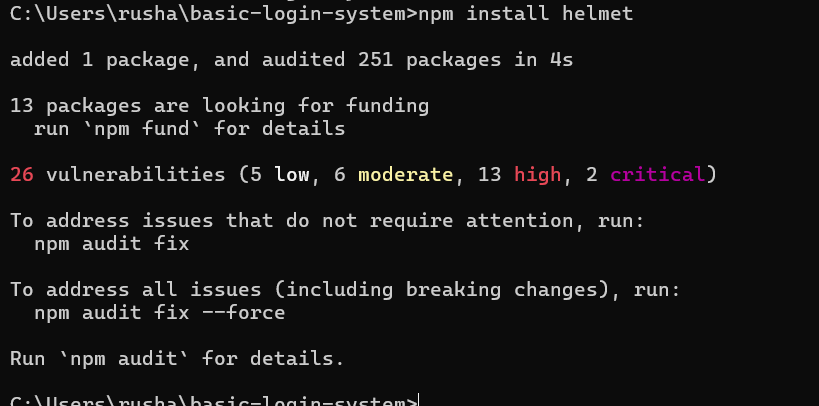


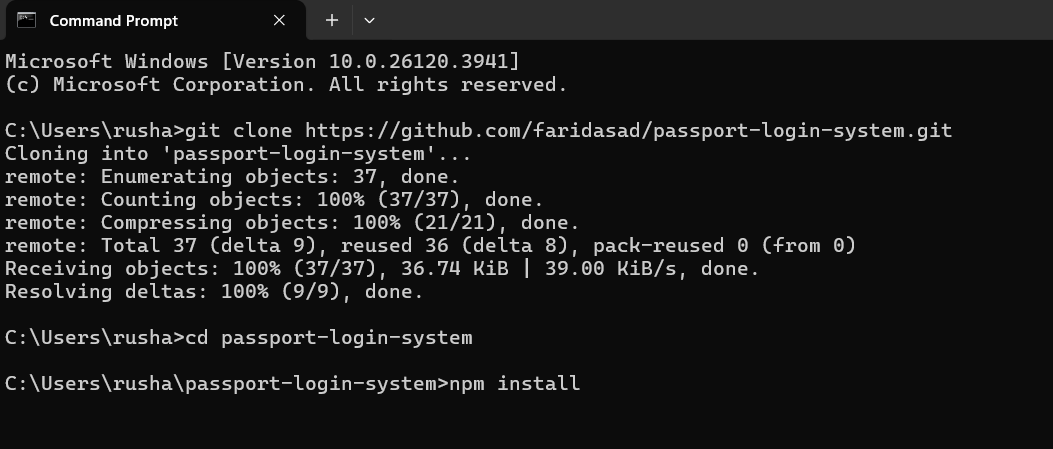












The application was launched using:

bash

npm start

Upon exploring the login, signup, and profile features via http://localhost:3000, vulnerabilities were examined.

**Basic Vulnerability Assessment**

Three primary security checks were conducted:

* **OWASP ZAP Scan:** Automated scan detected **XSS vulnerabilities and missing security headers**.
* **Browser Developer Tools for XSS Testing:** Injecting the following script into a form field led to a popup:

html

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

* **SQL Injection Attempt:** Entering the query:

sql

admin' OR '1'='1

bypassed authentication, confirming **SQL Injection vulnerability**.

**Documenting Findings**

Based on the assessments, the following vulnerabilities were recorded:

Recommendations included **input validation, password hashing, token authentication, secure HTTP headers, and logging mechanisms** to improve security.

**Week 2: Implementing Security Measures**

**Sanitizing & Validating Inputs**

To eliminate XSS and SQL Injection risks, **Validator.js** was installed:

bash

npm install validator

Sanitization rules were applied in the registration route:

js

const validator = require('validator');

app.post('/register', async (req, res) => {

try {

if (!validator.isEmail(req.body.email)) throw 'Invalid email format';

if (!validator.isLength(req.body.password, { min: 6 })) throw 'Password must be at least 6 characters long';

if (!validator.isAlphanumeric(req.body.username)) throw 'Username must only contain letters and numbers';

const hashedPassword = await bcrypt.hash(req.body.password, 10);

users.push({

id: Date.now().toString(),

email: req.body.email,

username: req.body.username,

password: hashedPassword

});

res.redirect('/login');

} catch (e) {

res.render('register', { errors: e });

}

});

**Password Hashing with bcrypt**

To prevent password theft, **bcrypt.js** was installed:

bash

npm install bcrypt

Passwords were now hashed before being stored:

js

const bcrypt = require('bcrypt');

const saltRounds = 10;

const hashedPassword = await bcrypt.hash(req.body.password, saltRounds);

During login, bcrypt verifies password authenticity:

js

const match = await bcrypt.compare(req.body.password, user.password);

if (!match) return res.status(400).send("Incorrect password");

📌 **Screenshots to paste here:**

* **Code snippet showing password hashing before storage.**
* **Database entries after implementation (hashed passwords).**

**Token-Based Authentication (JWT)**

**JWT authentication** was introduced to protect user sessions:

bash

npm install jsonwebtoken

A **JWT token** is generated upon login:

js

const jwt = require('jsonwebtoken');

const token = jwt.sign(

{ id: user.id, email: user.email },

process.env.JWT\_SECRET,

{ expiresIn: '1h' }

);

res.json({ token });

Middleware ensures tokens are required for protected routes:

js

function authenticateToken(req, res, next) {

const token = req.headers['authorization']?.split(' ')[1];

if (!token) return res.status(401).json({ message: "Access denied. No token provided." });

try {

req.user = jwt.verify(token, process.env.JWT\_SECRET);

next();

} catch (err) {

res.status(403).json({ message: "Invalid or expired token." });

}

}

**Secure HTTP Headers with Helmet.js**

**Helmet.js** was installed to prevent security threats:

bash

npm install helmet

Helmet middleware added to **server.js**:

js

const helmet = require('helmet');

app.use(helmet());

**Week 3: Advanced Security and Final Reporting**

**Penetration Testing After Security Fixes**

The same security tests were re-run after improvements:

* **OWASP ZAP showed fewer warnings.**
* **XSS script execution was blocked due to sanitization.**
* **SQL Injection attempts returned authentication failures.**

📌 **Screenshots to paste here:**

* **OWASP ZAP scan results post-fixes.**
* **Failed XSS and SQL Injection attempts.**

**Logging with Winston**

To track login attempts and potential security risks, **Winston.js** was installed:

bash

npm install winston

Logging implemented in server.js:

js

const winston = require('winston');

const logger = winston.createLogger({

level: 'info',

format: winston.format.combine(

winston.format.timestamp(),

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

),

transports: [

new winston.transports.Console(),

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

]

});

logger.info('Application started');

**Conclusion**

After implementing robust security features such as **input validation, password hashing, JWT authentication, HTTP security headers, and server logging**, the Basic Login System is now **significantly more secure**. This project demonstrates how common web vulnerabilities can be mitigated using industry-standard security practices.