**WEBSITE SECURITY -**

**BY KRITI JAIN AND SUNAYANA VEMPATI**

**PACKAGES TO BE INSTALLED :**

* MySQL via

1. sudo apt-get update
2. Sudo apt-get install mysql-server
3. Sudo mysql\_secure\_installation utility

* Node via

1. Sudo apt install nodejs

**Security during installation:**

1. For the secure installation of mysql use:

→ sudo mysql\_secure\_installation utility

This should prompt the user to define mysql root password and other security related options like removing remote access to the root user and setting the root password.

1. The root user has all the access rights.
2. Create user role in mysql to restrict the privileges given to user.

* CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';
* GRANT type\_of\_permission ON database\_name.table\_name TO ‘username’@'localhost’;

Note: type\_of\_permissions given: INSERT and SELECT

**Pages:**

1. Signup – A user can signup by providing first name, last name, email id, username, date of birth and password

2. Login – A user can login by providing correct credentials

3. Profile – A user will have option to upload pictures and view all of his/her pictures

**Technology Used:**

Backend: Node.js

Database: mysql

Framework: Express

Template engine: Ejs

**Authentication:**

Inside the views (Frontend), there is home page i.e. index.ejs which will have options to sign up and login.

**Signup:**

On clicking signup,

1. User will be taken to signup page where all the user details needs to be entered.
2. The form is validated to check for strong passwords and invalid inputs.
3. After signing up, passport.authenticate() is invoked which then invokes req.login to login new registered user.
4. Email is checked against existing emails in database to see if the user already exists.
5. The password is encrypted using bycrypt (bycrypt-nodejs module is used to achieve this) and then user details are stored into database. Passport sets up the session, serializes the user for the session.

**Login:**

1. On clicking login, user will be redirected to the login page where user needs to provide username and password to access the account.
2. Once the user provides the details, first database is checked if given user exists. If yes, then password validation is done.
3. If the entered details are correct, user will be redirected to upload image page, and if not then user will be redirected to the home page.

**Backup:**

Automatic daily backups are setup for mysql database into S3 bucket in case of the data is lost or modified by malicious users.

**Logout:**

When log out is clicked, passport will deserialize the session, removes user property and clears the session.

**Security Practices followed:**

* **Prevent DOS**: Enabled firewall to limit requests and encapsulated all the routes in a catch block.
* **Prevent SQL injection attack:** Using parameterized queries in SQL.

* **Prevent Cross Site Scripting:** Validating input on client side.
* **Patch vulnerable dependencies:** Using npm-audit to track any vulnerable dependency and patch it.
* **Bcrypt for hashing:** Used Bcrypt module for slow hashing which is not vulnerable to dictionary and brute force attacks.
* **Strong Password Policy:** Adapted techniques to ensure user inputted password is strong by using techniques like password should contain minimum length, caps and small letters, numbers and shouldn’t be in the list of frequently used passwords.Installed password-validator package.

* **Prevent Coding Vulnerabilities:** Used eslint-plugin for checking if any vulnerabilities exist like invoking child module, eval (to prevent running JavaScript code during runtime) etc. exists while coding.

* **Blacklisting JWT Tokens:** When using Passport.js module for authentication and serialization, it grants tokens. So, when some malicious activity is exposed, there is no way to take back the tokens. This can be handled by validating a token on every request with blacklisted tokens.

* **Creating User when running Node.js and SQL server:** Since root user hold complete privileges, created another user for running Node.js and SQL server with limited- only required permissions.

* **Hiding error details from clients:** Using express, as it hides the error details from the user, as they might hold sensitive information.

* **Modify session settings:** If cookies are sent over insecure communication, attacker can make use of these for sessions and finding out the framework. So, changed session settings instead of using default settings.
* **Safe redirects:** Validating user inputted links.
* **SSL Certificate:** Encrypts the data transmitted to site for creating secure link between user browser and website.