Secure Programming Project Document

Paavo Kemppainen

Contents

General description

The software is a Single Page Web application (SPA) done with React as the frontend client and Node.js as the backend server. React was chosen as the frontend technology as it is quite popular still and I didin't have previous experience with it so this was a good chance to learn it. Node.js was chosen as the backend technology as it has wide range of middlewares which can be used to implement features quickly and most often the middleware are more secure and tested what single projects could achieve by doing them by themselves. The database accessed by the backend is done with PostreSQL and node's 'pg' middleware as they work well together and we can do some security testing with SQL databases.

The purpose of the software is to be a secure base for future applications. At this point only offers the functionalities such as register user, login user, and show user profile information. These features by themselves offer a good base for possible security vulnerabilities which can then be found with manual and automated testing which is the second goal of this project.

Structure of the Program

Frontend

React development can be done in two different components. First is the earlier type of class/object component based development and then there is the latest addition of functional component development. Both of them have their own advantages and for various reasons I ended up trying both of them as different components could be done in either of the styles. The React App starts with the App.js which contains the base of the client application. Different components can be imported and used which makes React so flexible to use. The components done and used in this project are Login, UserForm, and UserProfile. (Other components and files are by default in the generated/imported base)

Backend

Node.js works as our API for the React client. The app.js is the base for the server and all the middlewares that are used before the routing are implemented here. The request is processed in the route that was a match. If the request needs to get, update, or insert some data to the database, it is done in the database module whose functions the routes call. The data that is saved is user information such as last name, first name, email, and the password hash with the salt.

Secure Programming Solutions

HTTPS: The application works through HTTPS which is an encrypted communication channel. This provides server authentication and data integrity. The sertificate used in at the moment is not applied via any Public Key Infrastructure (PKI) Certificate Authority (CA) but is self signed. This process would be done in the real version of the sofware.

Passport: Node.js middleware that is used for user authentication with different authentication strategies. This software utilizes the local strategy which entails that the user account is only used in this application and the account details are saved in the application's server. Other options could be Facebook, Twitter or other authentication via the OAuth2 authorisation protocol where the user would use some other services account.

Bcrypt: Node.js middleware that enables easy and effortless password hashing and storage format. This middleware is able to hash the password securely using the Blowfish block cipher. These hashes are then saved to the database instead of plaintext/encrypted passwords. When a login attempt is made the given password is hashed again and compared to the hash in the database.

PG: Node.js middleware used to do queries to the PostgreSQL database. It offers parametrized queries which then prevent SQL injection attacks.

Access Control: Users that are logged in cannot see data that is not theirs. This can be seen in the profile page and its backend implementation. The user is first checked to be authenticated, the data row for the authenticated user is fetched from the database and the data is returned.

Local storage / cookie clearance: When the user logouts the session is destroyed, the cookie is session cookie token is cleared from the response which tells the browser to delete the cookie as well. The client also clears it's local storage so that the UI doesn't think it is logged in.

Testing

Technologies/Tools

For the backend server code testing, Jest framework was chosen as it offers easy and fast setup for Javascript testing. For library vulnerabilities, Retire.js as a Grunt plugin was used with npm's own 'npm audit' command.

The API testing is done with the Jest and Supertest. Supertest allows us to use Jest to make HTTP requests to test the API.

Done tests

SQL injection: SQL injection can be and is tested with the Jest framework as the only place the database is accessed is through the userDB component.

Access control / Login: Login credentials are used for profile pages. Manual access control checking is done. Supertest has capabilities to keep the session but at the moment there is some error with the passport and supertest compatibility/configuration so no automated tests work here.

Library vulnerabilities: The test is run in command line 'grunt retire'. The code base doesn't have vulnerabilites. With 'npm audit' there are numerous package dependency vulnerabilites with 1 being critical, 22 high, 7 moderate, and 15 low, but these are mostly from the deveDependencies which are used for the testing process. With ‘npm audit --production’ the vulnerabilities narrow down to 3 low and one critical.

ESLint was used for brief static code analysis, but there were not any major discoveries. Some unused variables, but that’s about it. There is an ESLint security plugin which could be used to further test with own or imported rules.

Further Development

If server goes down after login the UI doesn’t know that the sessions are lost. (Local storage must be cleared)

Remember me optionality. (Now remebers always if the server doesn’t go down or the user doesn’t log out)

Refactoring and making code more reusable.

Actual application business features.

Better error handling and detection.

More regular and security testing.

Go through all the OWASP and find proper technologies / solutions to protect against them

There are lots of material for safe Node.js development that gather the some of the usable tools such as https://github.com/lirantal/awesome-nodejs-security. This could be used as a reference for further security testing.

Profile information updating needs some work.

Dockerize the project.

Known Vulnerabilities

As automated tests are not working for access control, these could be vulnerable.

There is one critical vulnerability in the module dependencies within the production modules (doesn’t include modules used for testing). This is with package ‘constantinople’ and allows ‘Sandbox Bypass Leading to Arbitrary Code Execution’.