Lukuhaaste

Author: JP Paalassalo
Course:

Introduction

The goal of this project is to demonstrate secure programming concepts in MEAN stack application. The application is a checklist tool for 50 preset reading challenges, and the user can record reading achievements for each challenge. The achievements are shared to other app users.

In the context of this course, the following work was done:

- top-level threat analysis for the app including networking and programs
- identifying and prioritizing threats
    - implementing https for both frontend and backend
    - introducing user capabilities for frontend
    - implementing auth0 login and jwt sessions (frontend)
    - jwt tokens for backend
- existing demo was deployed to public service

Application architecture and deployment

The application consists of Angular single-page app frontend served by nginx, nodejs+express backend and Mongo database. These three are deployed in shared network, and firewall configuration allows external access to necessary frontend and backend ports.

```
@startuml
nwdiag {
  group docker {
    color = "#FFaaaa";
    MongoDB [shape=database];
    LukuhaasteBackend [shape=node];
    LukuhaasteFrontend [shape=node];
    xendocker;
  }
  internet [shape = cloud, description = "internet"];
  internet -- firewall;
  firewall [shape=node ];
  network DMZ {
      address = "10.x.x.x/24";
      firewall;
      MongoDB;
      LukuhaasteFrontend;
      LukuhaasteBackend;
  }

  network development {
    address = "10.y.y.y/24";
    firewall;
    xendocker [shape=node, description = "zen + docker\nplatform"];
    workstation [ shape=node ];
  }
@enduml
```

The selected architecture exposes database (and its admin access endpoint) unnecessarily to DMZ network. An alternative decision would be to place database to an internal network visible only to backend. The rationale for the selected architecture is that docker does not readily support connecting containers to multiple networks; it would be necessary to start the containers first and then attach the running backend container to other network. It would make system setup and maintenance more complex.

The database exposure vulnerability can be mitigated by limiting database access to backend and admin IP's only.

Threat analysis

Threat analysis was implemented using a threat modelling process where all system components and interfaces were studied, attack vectors were identified and each vulnerability was scored as risk (0-5) based on damage x probability.

Scope	Vulnerability	Risk	Mitigation
firewall	configuration error	2	Utilize external testing services
firewall	unauthorized access	2	Keep FW updated. Strong admin password. Restrict access from internal network too.
frontend	nginx vulnerabilities	4	Set up watchtower to rebuild frontend as nginx upgrades are available.
	docker API vulnerabilities		Keep upgrading platform. All public

docker platform	exposed to containers	1	containers must run with limited non-root privileges.
xen platform	virtualisation network stack has vulnerability	1	Keep xen up-to-date.
backend/frontend node platform	node library vulnerabilities	2	Check and fix vulnerabilities using snyk.
development	Development workstation gets infected and provides admin access to system	5	Isolate development environment to a separate workstation or on Qubes OS.
frontend code	Frontend provides unauthorised access to backend	1	(Frontend code is public and fully modifiable by attackers; not much can be done) Implement user authentication and authorization. Sanitize data sent to backend.
frontend code	Frontend identity is compromised enabling MITM attack and possibly stealing user credentials	2	Use proper certificates and force using HTTPS, use external authentication.
frontend code	Lost user credentials allow stealing the account	2	Enable multi-factor authentication and anomaly detection
backend code	unauthorised access	3	Implement user authentication and jwt sessions
backend code	injection attacks	3	Sanitize all incoming data
backend code	CORS	3	Implement. Set SameSite cookie attribute to Strict.

Authentication and authorization

In order to achieve all required functionality and security authentication and authorization is inplemented using external service. Auth0 was selected. In Lukuhaaste use case, the recommended setup is "Authorization Code Flow with PKCE" where PKCE stands for "proof Key for Code Exchange". [https://auth0.com/docs/flows/authorization-code-flow-with-proof-key-for-code-exchange-pkce] [https://auth0.com/docs/libraries/auth0-angular-spa]

```
@startuml
actor User
skinparam minClassWidth 120
participant "Lukuhaaste\n Angular App\n in Browser" as Browser
participant "Lukuhaaste\n backend API\n " as backend
participant "\nAuth0 Tenant\n" as auth
participant "Google \nauthentication API\n" as google
== Initialization ==
rnote over auth
 Tenant is configured:
 Client ID
 Allowed redirect URIs
endrnote


== Authentication ==
```

```
User -> Browser: Click "login"
rnote over Browser
 Generate Code
 Verifier and
 Code Challenge
endrnote
Browser --> auth: Authentication Code Request\n and Code challenge
auth --> Browser: Auth0 universal login\n HTML page
User --> Browser: Enter google credentials
Browser --> google: request access token
rnote over auth
 Generate one-time
 Authorization code
 and redirect to app
endrnote
auth --> Browser: Authorization Code
note over Browser
 Redirect with Authorization Code can be captured
 by malicious code. It is of no use without
 Code Verifier that proves only original requester
 gets Access Tokens.
endnote
Browser --> auth: Authorization Code\n and Code Verifier
rnote over auth
 Verify
endrnote
auth --> Browser: ID Token Access Token

== Access backend ==
Browser --> backend: GET/PUT/POST\n data with Access token
rnote over backend
 Validate tokens
 Verify token claims
endrnote
backend --> auth: verify token
auth --> backend: OK
backend --> Browser: Response
@enduml
```

CORS

@startuml
participant Browser
participant "Frontend\nlukuhaaste.prgramed.fi" as front
participant "Backend\napi.lukuhaaste.prgramed.fi" as back
Browser -> front ++ : GET index.js
front -> Browser -- : 200 OK Access-Control-Allow-Origin: lukuhaaste.prgramed.fi
activate Browser

note across #FFAAAA
Backend access vulnerability:
Backend does not know from what context the client makes requests.
The browser will automatically attach available access cookies to requests.
User can load code from malicious web site, unknowingly making requests
to vulnerable backend using user credentials.
end note

note across
Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism
that allows a server to indicate any other origins (domain, scheme, or port)
than its own from which a browser should permit loading of resources.
The mechanism relies on browser implementing CORS protocol.
end note

note across
A CORS preflight request is a CORS request
that checks to see if the CORS protocol is understood
and a server is aware using specific methods and headers.
A preflight request is automatically issued by a browser.
end note

Browser -> back ++ : OPTIONS /api/challenges \n
Origin: lukuhaaste.prgramed.fi
back -> Browser -- : 204 no-content \n
Access-Control-Allow-Origin: lukuhaaste.prgramed.fi \n
X-Frame-Options: SAMEORIGIN

note over Browser
Browser error if js was loaded from other origin
than indicated by Access-Control-Allow-Origin
end note

Browser -> back ++ : GET /api/v1/users

note over back
Request headers must match
those indicated in preflight reply
end note

back -> Browser -- : 200 OK \n
Access-Control-Allow-Origin: lukuhaaste.prgramed.fi
@enduml

ReadingChallenge

This project was generated with [Angular CLI (https://github.com/angular/angular-cli)](https://github.com/angular/angular-cli) version 11.2.0.

Development server

Run `ng serve` for a dev server. Navigate to `http://localhost:4200/`. The app will automatically reload if you change any of the source files.

Code scaffolding

Run `ng generate component component-name` to generate a new component. You can also use `ng generate directive|pipe|service|class|guard|interface|enum|module`.

Build

Run `ng build` to build the project. The build artifacts will be stored in the `dist/` directory. Use the `--prod` flag for a production build.

Running unit tests

Run `ng test` to execute the unit tests via [Karma (https://karma-runner.github.io)](https://karma-runner.github.io).

Running end-to-end tests

Run `ng e2e` to execute the end-to-end tests via [Protractor (http://www.protractortest.org/)](http://www.protractortest.org/).

Further help

To get more help on the Angular CLI use `ng help` or go check out the [Angular CLI Overview and Command Reference (https://angular.io/cli)](https://angular.io/cli) page.