Secured Software Engineering
(COMSM0164)
Lab Exercise: Securing Real-Time APIs (Part 1 – WebSockets)

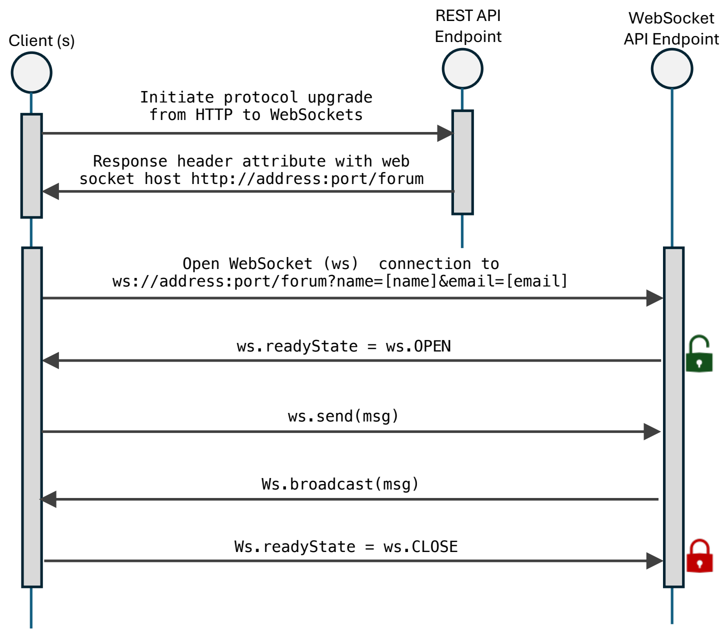
Prerequisite:

(1) Visual Studio Code (VS Code)
(2) A GitHub account
(3) Some knowledge of JavaScript and Node.js for API functional implementation.
(4) The client-side in this exercise also uses JavaScript, HTML and CSS. The latter two are optional prerequisite which enables a user-friendly and interactive web interface design.

Overview

In this exercise, you will examine the implementation of a real-time API as an alternative to RESTful APIs. You will be able to identify scenarios where *near-instantaneous* data exchange between software APIs is essential and the relevant secured software design approach. In part 1 of this exercise, you will achieve this objective by implementing a real-time discussion forum called **SSE-Forum**. The resulting real-time implementation is shown in the sequence diagram below. This includes the implementation of a client that initiates an upgrade from HTTP to WebSockets protocol via a REST API to a WebSocket endpoint. Once the upgrade is achieved, continuous messaging between the client and WebSocket endpoint is enabled until one of the parties triggers a close event. You will implement this sequence of data and information controls by following steps 1-8.

SSE-Forum will be initially insecure and likely to compromise the functional requirements of the system. As the next step, you will explore the security requirements associated with exposing real-time APIs. Using SSE-Forum to achieve additional challenge tasks, you will suggest potential security flaws in the design. In part 2 of this exercise, you will implement possible mechanisms that can be used to satisfy identified security flaws.


Client (s)
REST API Endpoint
WebSocket API Endpoint
Initiate protocol upgrade from HTTP to WebSockets
Response header attribute with web socket host http://address:port/forum
Open WebSocket (ws) connection to ws://address:port/forum?name=[name]&email=[email]
ws.readyState = ws.OPEN
ws.send(msg)
Ws.broadcast(msg)
Ws.readyState = ws.CLOSE

SSE-Forum Information Flow Design

Step 1: Install Node.js and NPM

You can download the latest LTS version from https://nodejs.org/en. Follow the prompts in the Node.js Installer and customize the defaults, if necessary. When you're done, you should have installed Node.js, as well as NPM (Node Package Manager).

Verify the installation by running the following commands in your terminal:

```
node -v
npm -v
```

If the versions of Node.js and NPM show up, then your installation was successful.

Step 2: Create a new project folder

Next, create a new folder for the project by running the following command in your terminal:

```
mkdir ws_api
```

To navigate to the project enter the command:

```
cd ws_api
```

Step 3: Initialize a new Node.js application

Open the folder created in step 2 on VS Code. To initialise the project, navigate to the folder on VS Code terminal and run the following command:

```
npm init
```

You will be prompted to enter the project name, description, and GitHub repository. You can accept the defaults by pressing Enter/Return, or customise them. Open the generated `package.json` file and carry out the following refactoring tasks:

1. Modify the `main` field to `server.js`. This field is a property that holds the primary entry point to the system.
2. The `scripts` property is a dictionary containing script commands that are run at various times in the lifecycle of your package. The key is the lifecycle event, and the value is the command to run at that point. Examples of lifecycle events can be found at https://docs.npmjs.com/cli/v9/using-npm/scripts. Add a `start` lifecycle event and set the value to `node server.js`. This indicated that when the command npm `start` is executed on the terminal, then node.js will execute `server.js`.

At this point, the resulting `package.json` is as follows:

```
{
  "name": "ws_api",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {

  }
}
```

Step 4: Install Dependencies

SSE-Forum utilises four external dependencies. These are:

`Dotenv` - A zero-dependency module that loads environment variables from a .env file into node.js process. This storing configuration in the environment separate from code is based on the Twelve-Factor App methodology (https://12factor.net/config). This means that system configurations such as credentials, database handles, and canonical hostnames are not stored as constants in the code.

`express` – A back-end web application framework for building RESTful APIs with Node.js.

`ip` – A library for validating and manipulating IPv4 and IPv6 addresses in JavaScript

`WS` – A library for creating WebSocket servers for Node.js. Note that WebSockets are natively supported in all modern browsers. Hence, SSE-Forum utilises WS on the server and then the browser's WebSocket API on the client.

Run the following commands from the `ws_api` directory to install these dependencies:

```
npm install dotenv
npm install ip
npm install express
npm install ws
```

Step 5: Setup Config Environment

In this step, you will implement the template necessary to separate configuration from code. In the `ws_api` directory, create a file named `.env -` VS Code automatically detects `env.` files as configuration files. Given that these files may contain sensitive attributes, they must not be checked into revision control.

Step 6: Implement SSE-Forum API Server

The server API component that is responsible for ensuring a persistent information flow channel between the client and server is implemented in this step. The exchange of messages between the two parties is based on the WebSockets protocol. To achieve this step, copy server.js from the exercise folder into the `ws_api` folder. Take some time to study and understand the functionality implemented in this component. The key points are as follows:

- The first set of statements imports the necessary modules and initialises a variable called `clients` of type Map. This variable holds references to active SSE-Forum members.

- The `start` function exposes the API server to clients by executing the `initWS` and `sendWSEndpoint` functions

- The `initWS` function implements the websockets protocol to persist the information flow channel between the client and server. These include listening to message requests from clients, broadcasting received messages to all clients and closing web socket close events. Note that the function creates a WebSocket server and listens to clients' requests via a port number retrieved through a reference to the environment (`process.env.PORT_WS`). To set up this reference, add the following line in your `.env` file:

  ```
  PORT_WS=8888
  ```

  Note that this approach has the advantage of allowing you to change the port number to which the SSE-Forum API listens without changing its code. In this example, the WebSocket is exposed to clients via [server_ip_address]:8888

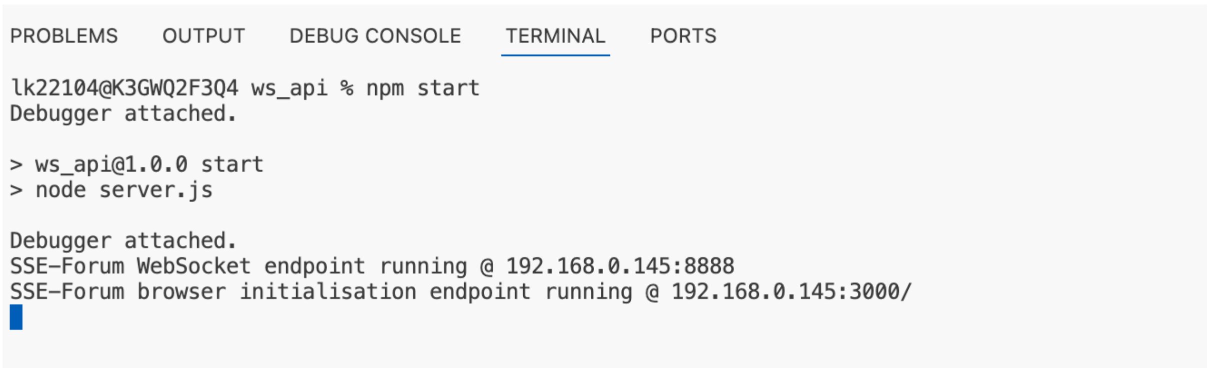
- A direct connection to SSE-Forum API by typing [server_ip_address]:8888 on your browser would result in a "Upgrade Required" error. This error indicates that the connection between the client and server API is currently based on HTTP and needs to be upgraded to WebSockets. This upgrade can only be programmatically achieved through the native WebSocket implementation in browsers. The `sendWSEndpoint` function runs an express server that sends the SSE-Forum WebSocket endpoint as a response header attribute to the client browser. This is retrieved and used by the browser to natively send a WebSocket upgrade request to SSE-Forum API.
  The express server listens to clients' requests via a port number retrieved through a reference to the environment (`process.env.PORT_INIT`) and exposes a the

  ```
  PORT_INIT=3000
  ```

"/" endpoint via RESTful API call. To set up this reference, add one more line below to your `.env` file:
In this example, the express server listens to clients' requests using the address [server_ip_address]:3000

- Finally, the `broadcastMessage` and uuidv4 `uuidv4` functions are utility functions that `initWS` uses to broadcast received messages to all clients and generate a unique identifier for a user, respectively.

By completing this step, you have implemented the SSE-Forum API server, with the ability to listen to WebSockets requests and respond in real time via a persistent information flow channel. Run the `npm start` as follows on your terminal for SSE-Forum to begin listening to clients' requests.



```
lk22104@K3GWQ2F3Q4 ws_api % npm start
Debugger attached.

> ws_api@1.0.0 start
> node server.js

Debugger attached.
SSE-Forum WebSocket endpoint running @ 192.168.0.145:8888
SSE-Forum browser initialisation endpoint running @ 192.168.0.145:3000/
```

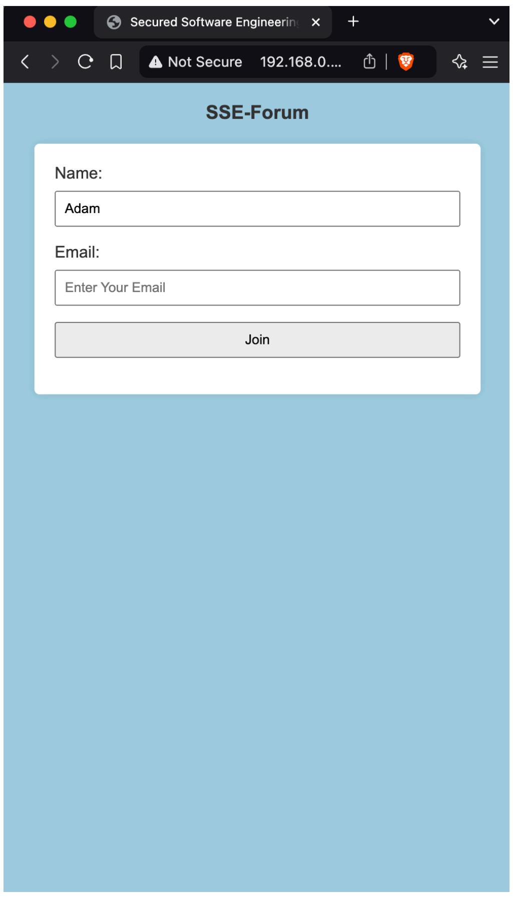
Step 7: Implement SSE-Forum Client

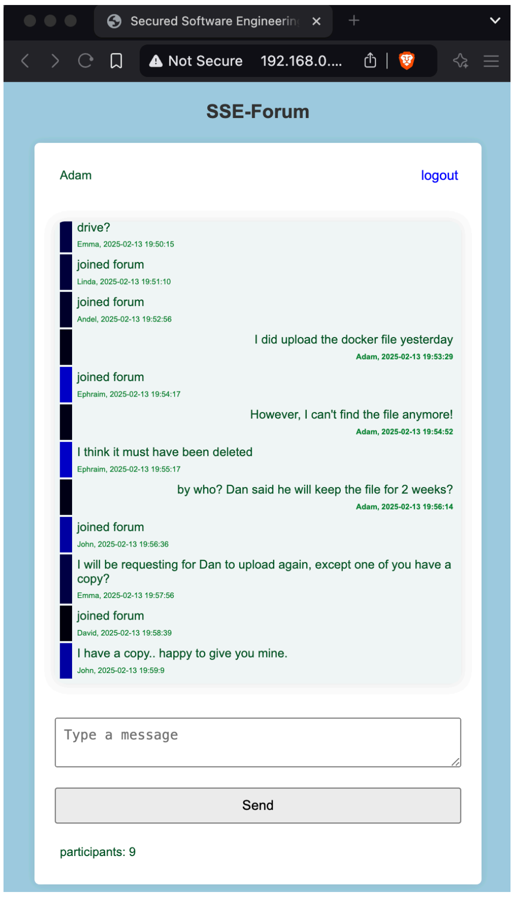
SSE-Forum client is a web browser that requests a WebSocket endpoint from the REST API endpoint created by `sendWSEndpoint` function. A successful request includes a REST response that embeds the WebSocket endpoint in its header. The client then uses the browser's native WebSocket to initiate a protocol upgrade from HTTP to WebSockets using the received endpoint. Once the connection is open by the WebSocket API, messages are passed between both the client and server until one of the parties initiates a close event. To achieve this step:

(1) Copy the index.html file into the `ws_api` directory. Take some time to study and understand the functionality implemented in this file. The key functions are `initWSConnection,` which is used to orchestrate the opening of a web socket connection, listening to broadcast messages and sending messages. The `connectToServer` is a helper function that gets the name and email from the user. It then uses native browser WebSocket to request a persistent connection to the server API WebSocket endpoint.

(2) Copy the `public/css` folder into the `ws_api` directory. This folder contains a `style.css` file that enhances formatting in index.html.

By completing this step, you have implemented the SSE-Forum client. To make a client request, input [server_ip_address]:3000 on your web browser. This achieves the first step in the sequence diagram of making a REST API request to initiate a protocol upgrade from HTTP to WebSockets. A successful execution will result in a browser page requesting for the name and email of a user. This is shown in the left figure below:





drive?
Emma, 2025-02-13 19:50:15

joined forum
Linda, 2025-02-13 19:51:10

joined forum
Andel, 2025-02-13 19:52:56

I did upload the docker file yesterday
Adam, 2025-02-13 19:53:29

joined forum
Ephraim, 2025-02-13 19:54:17

However, I can't find the file anymore!
Adam, 2025-02-13 19:54:52

I think it must have been deleted
Ephraim, 2025-02-13 19:55:17

by who? Dan said he will keep the file for 2 weeks?
Adam, 2025-02-13 19:56:14

joined forum
John, 2025-02-13 19:56:36

I will be requesting for Dan to upload again, except one of you have a copy?
Emma, 2025-02-13 19:57:56

joined forum
David, 2025-02-13 19:58:39

I have a copy.. happy to give you mine.
John, 2025-02-13 19:59:9

When a user provides a name and email and clicks the Join button, a persistent WebSocket connection is opened, allowing the user to discuss with other participants in the forum. The right figure above is a screenshot of Adam's browser interface and shows the discussions he is having with other participants in the forum.

Step 8: Push project to GitHub repository

Log in to your GitHub account and create a repository called ws-api. Make sure that the repository is marked as private. To keep it simple, do not include a .gitignore, readme.md or license file. We will rather add a .gitidnore file locally on VS Code before pushing to GitHub.

Create a file called .gitignore and enter the following into the file:

```
node_modules
.env
```

This will ensure that dependencies in node_modules directory and environment variables containing secrets are not committed to the git repository.

Finally, execute the following git commands on the terminal to push the project to GitHub repository:

```
git init
git add .
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/[GitHubID]/ws-api.git
git push -u origin main
```

Challenge Task:

Take some time to review SSE-Forum – both the browser client and server API implementation. Run the program as indicated in steps 6 and 7, and test with multiple participants. Also, inspect the HTTP header attributes (To do so on Chrome browser, right-click, then select Inspect and press the network tab). Particularly, observe how we have leveraged HTTP cookies to share WebSocket the endpoint and session ID with the client. Following on, attempt the following two challenge tasks.

1. Consider a redesign of SSE-Forum purely based on REST API requests from clients and responses by the server. Discuss in no more than 300 words the tradeoffs (risks and/or benefits) associated with this alternative design.

2. As browsers impose further restrictions on third-party cookies, developers should start to look at ways to reduce their reliance on them. Propose an alternative design that does not use cookies

3. Based on your implementation of steps 1-8 above, identify a minimum of three security flaws in the system.

4. Refactor SSE-Forum to use WebSocketsSecure (wss) instead of websockets

5. Which security flaws in (3) has wss mitigated.

6. In no more than 300 words, discuss the severity of the security flaws identified and how they can be mitigated.

Deliverables

Submission should be made electronically as a PDF containing all deliverables via Blackboard for Exercise X. The PDF should also contain a maximum of 250 words of reflection on the exercise to help keep a record of your learning. Be sure to include a link to your GitHub repository.

IMPORTANT NOTES:

- Check that you've invited the course lecturer to your GitHub repository and that an acceptance of the invitation was received and acknowledged.
- Also, ensure that one of the TAs or the course lecturer has reviewed your solutions before the end of the lab session.