Secured Software Engineering
(COMSM0164)

Lab Exercise: API Authentication Schemes

Part 1: Basic, Digest and Bearer Authentication

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.
(5) Postman/Rest Client

Overview

This exercise builds on the previous exercise on Implementing HTTPS APIs using SSL certificates. We will extend the implementation of E-Manager (an event management system) to explore different API authentication schemes and understand the level of security they provide, typical usage scenario, specific considerations and their limitations.

Step 1: Basic Authentication Scheme

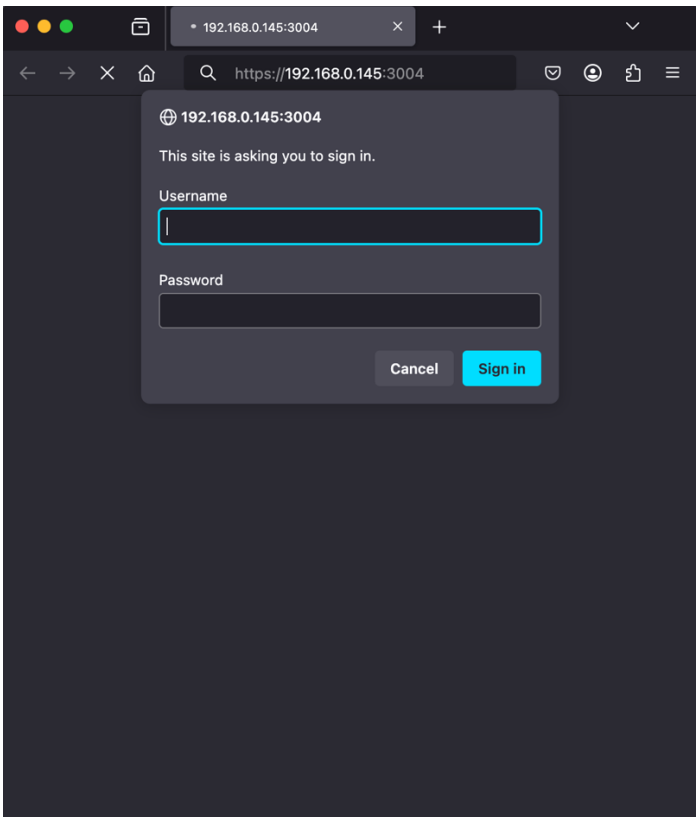
This is an HTTP authentication method where a user provides their username and password. This is encoded in base64 format in the request header, allowing a server to verify the user's identity before granting access to a resource. To achieve this behaviour, the server API configures its response header to advertise `basic` authentication scheme as the HTTP authentication methods (or challenges) that should be used to gain access to a specific resource. Correspondingly, the authorisation request header on the client (browser) is set to `basic` authentication scheme with username and password credentials.

To implement a basic authentication scheme for E-Manager, copy `eserver-ssl-basic.js` from the exercise folder into the `ws_events` folder. **Take some time to explore the additional functionality implemented in this component.** The file contains two main refactoring on the API server. First is the `app.use` statement that mandates the express server to use the `authentication` function to verify users' identity. The second is the `authentication` function itself, which implements the basic authentication scheme behaviour.

Add `username=admin` and `password=admin` as two new environment variables to your `.env` file. Also, add a new script property to `package.json` as follows:

```
{
  "name": "ws_events",
  "version": "1.0.0",
  "main": "eserver.js",
  "scripts": {
    "start": "node eserver.js",
    "start-ssl": "node eserver-ssl.js",
    "start-ssl-basic": "node eserver-ssl-basic.js"
  },
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "dotenv": "^16.4.7",
    "express": "^4.21.2",
    "ip": "^2.0.1"
  }
}
```

Finally, to run the newly implemented features, execute the command `npm run start-ssl-basic` on your terminal for E-Manager to begin listening to clients' requests. To make a client request, input https://[server_ip_address]:3004 on your web browser. A successful outcome will first generate self-signed certificate warnings. Bypass this warning, and the browser will pop up a username and password request, as shown below. Log in using the credentials username=admin and password=admin.



Challenge Task 1

Assuming you desire an alternative behaviour for E-Manager where user authentication is necessary only when adding a new event or deleting an existing one – i.e authentication is not required to view events E-Manager. Implement your modifications to the `eserver-ssl-basic.js file` and save it as `eserver-ssl-basic_v2.js` in the `ws_events` folder.

Step 2: Digest Authentication Scheme

This HTTP authentication method offers a more secure approach to handling credentials compared to basic authentication by ensuring that user credentials are not sent in clear text. In achieving this behaviour, the server API sets its WWW-Authenticate header to advertise `digest` authentication scheme as the HTTP authentication method (or challenges). The authentication logic involves a series of steps including an initial client request, server challenge, client response and server verification.

To implement this digest authentication scheme for E-Manager, copy `eserver-ssl-digest.js` from the exercise folder into the `ws_events` folder. This code sets up express server with digest authentication. **Take some time to explore the additional functionality implemented in this component.** The file contains two main refactoring on the API server. Similar to basic scheme, the `app.use` statement mandates the express server to use the `authentication` function to verify users' identity. The `authentication` function handles the authentication logic. which implements the basic authentication scheme behaviour. The `generateNonce` function creates a unique nonce for each authentication challenge, and the `parseDigestAuth` function parses the digest authorization header.
Users are stored in the `users` object which reads its values from the process environment. To create the three users utilised in the code, add the following key-value attributes to .env file:

```
username1=user1
password1=password1
username2=user2
password2=password2
username3=user3
password3=password3
```

The code depends on `crypto` package to implement the authentication logic, install this additional dependency by running `npm install crypto.` Add another script property to `package.json` as follows:

```
"scripts": {
    "start": "node eserver.js",
    "start-ssl": "node eserver-ssl.js",
    "start-ssl-basic": "node eserver-ssl-basic.js",
    "start-ssl-digest": "node eserver-ssl-digest.js"
  },
```

Run the digest authentication by executing the command `npm run start-ssl-digest` on your terminal for E-Manager to begin listening to clients' requests. Finally, initiate a browser client request to the address https://[server_ip_address]:3004 and log in using any of the credentials from the .env file.

Challenge Task 2

Assume you've received complaints from users that the performance of E-Manager is slow, and you observed that this is due to the digest authentication scheme operating over HTTPS.

1. Modify `eserver-ssl-digest.js` authentication to operate without SSL/TLS protocol.
2. Discuss the compromise you've made as a result of the new design decision that E-Manager should operate over HTTP rather than HTTPS.

Step 3: Bearer Authentication Scheme

Bearer Authentication is an HTTP authentication scheme where a client sends a security token, called a "bearer token," in the Authorization header of a request to access protected resources. Implementing this authentication scheme in a Node.js application involves a server API using tokens to authenticate and authorise clients. For a client to obtain a token, it needs to be authenticated by an authentication server. This usually involves the client sending the user credentials (username and password) to an authentication endpoint that returns a token. Once the user's credentials are verified, the client is given a token, which is included in the `Authorization` header of every HTTP request that points towards a protected resource endpoint.

To implement this bearer authentication scheme for E-Manager server API, copy `server-SSL-bearer.js` from the exercise folder into the `ws_events` folder. This code sets up express server with bearer authentication. **Take some time to explore the additional functionality implemented in this component.** The key functionalities relate to creating a secret-key (or private key) using JSON Web Token (JWT). There is also an additional authentication endpoint called `\login` that checks a user's credentials and grants tokens to clients when their users are validated. Finally, `authenticateToken` function is a middleware that verifies tokens before access to a protected endpoint is granted. Also, note the event endpoints that have been protected and the corresponding valid credentials selected from the environment file.

Similar to step 2, the code depends on `crypto` package to implement the token generation logic, install this additional dependency by running `npm install crypto`. Also, add another script property to `package.json` and run the server accordingly.

Secondly, to implement E-manager client, copy `eindex-bearer.html` from the exercise folder into the `ws_events` folder. This file is a refactoring of `eindex.html` to support the client in asking users for credentials and subsequently sending a post request to the login endpoint on the server API. Once validated, it uses the returned token for every HTTP request to a protected resource endpoint. . Finally, initiate a

browser client request to the address https://[server_ip_address]:3003 and log in using the corresponding valid credentials selected from the .env file.

Challenge Task 3

You observed that a specific client is being used to attempt a denial-of-service attack on E-Manager based on the high frequency of requests being made.

1. Suggest two changes to `eserver-ssl-bearer.js` to mitigate the likelihood of this observed attack.
2. Implement the two suggestions you've made in `eserver-ssl-bearer_2.js` and `eserver-ssl-bearer_3.js` respectively.
3. Discuss the advantages, disadvantages and compromises you've made as a result of the new design decisions made to implement your two suggestions.