



**BSc (Hons) in Information Technology**  
**Year 4 – Semester 2, 2022**

**Secure Software Development – SE4030**  
**Assignment 2**

**2022\_REG\_69**

<b>Student ID</b>	<b>Student Name</b>
IT19216874	Senevirathne R.Y.
IT19362854	Abeygunawardana S.L.
IT19217796	Ratnayake M.A.A.H.
IT14113246	Athukorala A. D. R. P.

## Table of Contents

Introduction.....	3
Message Authentication.....	3
Server Verification.....	3
Threat Modeling (Microsoft Threat Modeling Tool).....	5
Identified Mitigations (Microsoft Threat Modeling Tool).....	6
Summary of Threat Model.....	11
Test Plan.....	12
Test Results .....	13
Codebase .....	17

# Introduction

In this assignment, a web application was developed for ABC company that allows its staff to save messages and upload files. The client app acts as a simple interface for login and providing messages or files it runs over an insecure network. The server app acts as a data repository, and it stores the uploaded messages and files by the client app. There are 3 main user roles in this system such as administrator, worker, and manager.

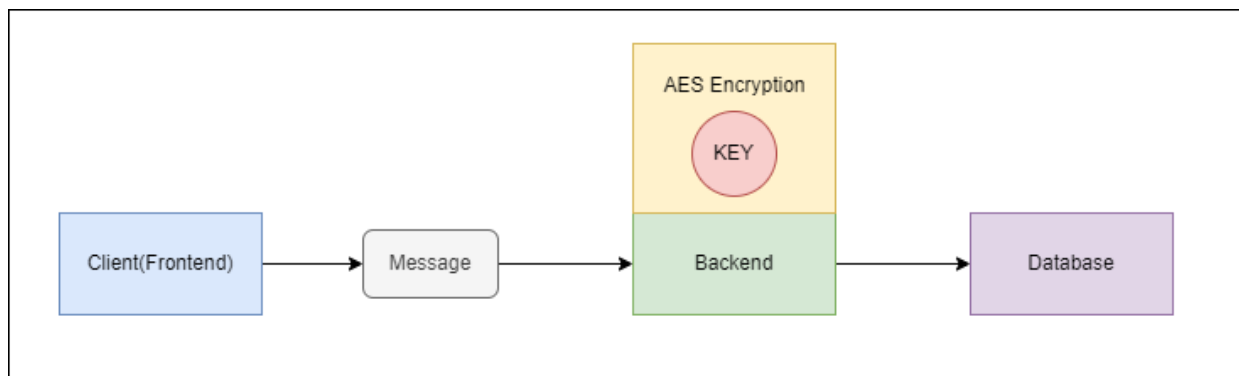
- The administrator can create accounts for the staff.
- Workers can only save messages.
- Managers can save messages and upload files.

In order to ensure the security of this web application, the below security requirements were considered.

- Confidentiality
- Integrity
- Message authentication
- Server identification

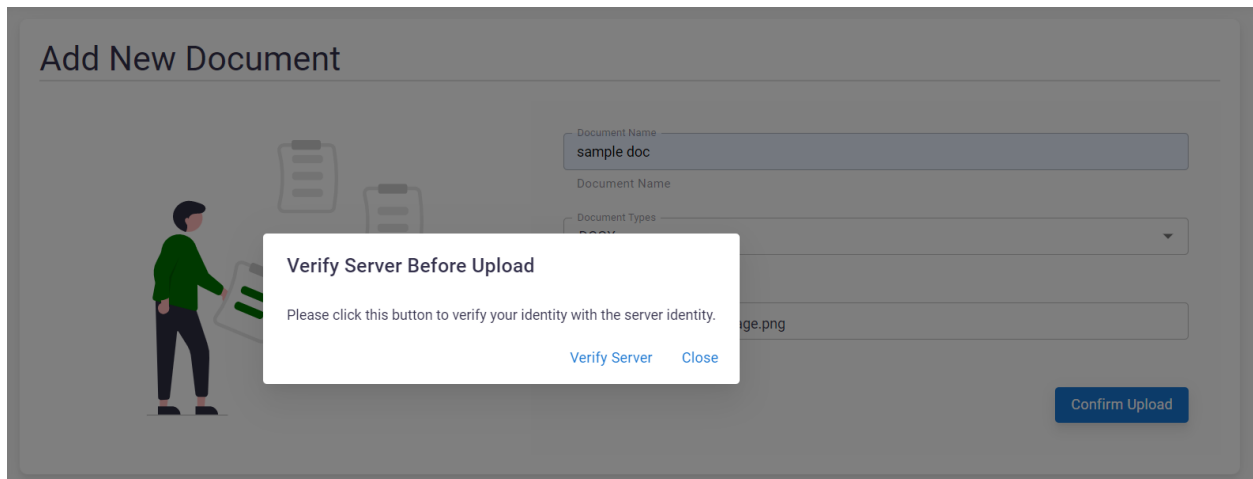
## Message Authentication

All messages sent by the user are encrypted in the back and saved to the database. Only the recipient and the sender can view the message content because the encryption algorithm uses a secret key unique to both users. The below diagram shows the flow of message encryption.



## Server Verification

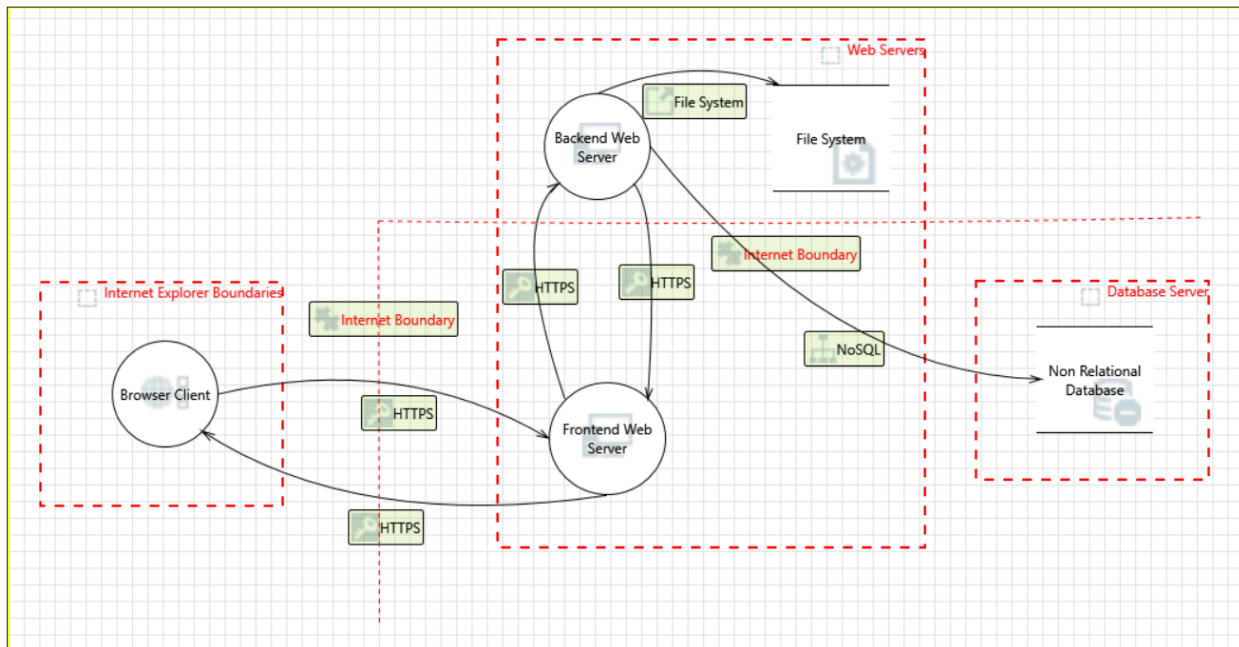
User can verify the server before upload any files to the server.



The screenshot shows a web interface for adding a new document. The main form is titled "Add New Document" and contains several input fields: "Document Name" (with the value "sample doc"), "Document Name" (empty), "Document Types" (a dropdown menu), and "Image" (with the value "image.png"). A "Confirm Upload" button is located at the bottom right. A modal dialog box is overlaid on the form, titled "Verify Server Before Upload". The dialog contains the text "Please click this button to verify your identity with the server identity." and two buttons: "Verify Server" and "Close". An illustration of a person in a green shirt and black pants is standing on the left side of the form.

# Threat Modeling (Microsoft Threat Modeling Tool)

Threat modeling is a process by which potential threats, such as structural vulnerabilities or the absence of appropriate safeguards, can be identified and enumerated, and countermeasures prioritized. For this assignment, Microsoft Threat Modeling tool was used. The Threat Modeling Tool is a core element of the Microsoft Security Development Lifecycle (SDL). It allows software architects to identify and mitigate potential security issues early when they are relatively easy and cost-effective to resolve. As a result, it greatly reduces the total cost of development.



## Identified Mitigations (Microsoft Threat Modeling Tool)

### 1. Spoofing of Destination Data Store File System [State: Mitigation Implemented] [Priority: High]

Category: Spoofing

Description: File System may be spoofed by an attacker, and this may lead to data being written to the attacker's target instead of File System. Consider using a standard authentication mechanism to identify the destination data store.

Justification: The file system can only be accessed by an authorized person that has a JWT authorization token and role-based authentication is applied to identify the user type.

### 2. Potential Excessive Resource Consumption for Backend Web Server or File System [State: Mitigation Implemented] [Priority: High]

Category: Denial Of Service

Description: Does Backend Web Server or File System take explicit steps to control resource consumption? Resource consumption attacks can be hard to deal with, and there are times that it makes sense to let the OS do the job. Be careful that your resource requests don't deadlock, and that they do timeout.

Justification: Because the file system can only be accessed by authorized users, the possibility of a DDOS attack is extremely low.

### 3. Spoofing the Browser Client Process [State: Mitigation Implemented] [Priority: High]

Category: Spoofing

Description: Browser Client may be spoofed by an attacker, and this may lead to unauthorized access to Frontend Web Server. Consider using a standard authentication mechanism to identify the source process.

Justification: JWT Authorization/Authorization

### 4. Cross Site Scripting [State: Mitigation Implemented] [Priority: High]

Category: Tampering

Description: The web server 'Frontend Web Server' could be a subject to a cross-site scripting attack because it does not sanitize untrusted input.

Justification: Frontend Server-Side Script Encryption

**5. Potential Data Repudiation by Frontend Web Server [State: Mitigation Implemented] [Priority: High]**

Category: Repudiation

Description: Frontend Web Server claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.

Justification: Node JS backend server logs.

**6. Data Flow HTTPS Is Potentially Interrupted [State: Not Applicable] [Priority: High]**

Category: Denial Of Service

Description: An external agent interrupts data flowing across a trust boundary in either direction.

Justification: JWT Authorization/Authorization

**7. Elevation Using Impersonation [State: Mitigation Implemented] [Priority: High]**

Category: Elevation Of Privilege

Description: Frontend Web Server may be able to impersonate the context of Browser Client to gain additional privilege.

Justification: JWT authentication and react protected routes with role-based authentication.

**8. Elevation by Changing the Execution Flow in Frontend Web Server [State: Mitigation Implemented] [Priority: High]**

Category: Elevation Of Privilege

Description: An attacker may pass data into Frontend Web Server to change the flow of program execution within Frontend Web Server to the attacker's choosing.

Justification: JWT authentication, Redux state implemented authentication, React protected routes with role-based authentication.

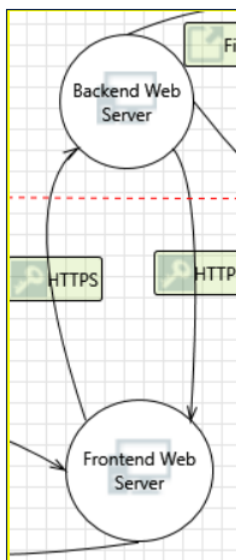
## 9. Cross Site Request Forgery [State: Mitigation Implemented] [Priority: High]

Category: Elevation Of Privilege

Description: Cross-site request forgery (CSRF or XSRF) is a type of attack in which an attacker forces a user's browser to make a forged request to a vulnerable site by exploiting an existing trust relationship between the browser and the vulnerable web site. In a simple scenario, a user is logged in to web site A using a cookie as a credential. The user browses to web site B. Web site B returns a page with a hidden form that posts to web site A. Since the browser will carry the user's cookie to web site A, web site B now can take any action on web site A, for example, adding an admin to an account. The attack can be used to exploit any requests that the browser automatically authenticates, e.g. by session cookie, integrated authentication, IP whitelisting. The attack can be carried out in many ways such as by luring the victim to a site under control of the attacker, getting the user to click a link in a phishing email, or hacking a reputable web site that the victim will visit. The issue can only be resolved on the server side by requiring that all authenticated state-changing requests include an additional piece of secret payload (canary or CSRF token) which is known only to the legitimate web site and the browser, and which is protected in transit through SSL/TLS. See the Forgery Protection property on the flow stencil for a list of mitigations.

Justification: All requests are protected with Self Signed Certificate. JWT token expired within a limited time and the user needs to validate or login again to validate the authenticity of the user. Furthermore, in the backend server role-based authentication is used to identify the user type.

Interaction: HTTPS





#### **10. Cross Site Scripting [State: Mitigation Implemented] [Priority: High]**

Category: Tampering

Description: The web server 'Backend Web Server' could be a subject to a cross-site scripting attack because it does not sanitize untrusted input.

Justification: JWT Authorization/Authorization

#### **11. Elevation Using Impersonation [State: Mitigation Implemented] [Priority: High]**

Category: Elevation Of Privilege

Description: Backend Web Server may be able to impersonate the context of Frontend Web Server in order to gain additional privilege.

Justification: React protected routes and Redux state-based token and role management.

#### **12. Potential Data Repudiation by Backend Web Server [State: Mitigation Implemented] [Priority: High]**

Category: Repudiation

Description: Backend Web Server claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.

Justification: JWT Authorization/Authorization

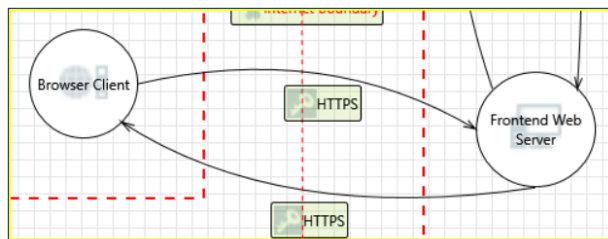
#### **13. Elevation by Changing the Execution Flow in Frontend Web Server [State: Mitigation Implemented] [Priority: High]**

Category: Elevation Of Privilege

Description: An attacker may pass data into Frontend Web Server in order to change the flow of program execution within Frontend Web Server to the attacker's choosing.

Justification: The frontend server is protected with SSL/TLS certificate. All the data passed into the frontend will be validated from the frontend by a JWT token generated from the backend and if the user is not recognized as valid immediately user will be redirected to the login.

Interaction: HTTPS



#### **14. Spoofing of Destination Data Store Non-Relational Database [State: Mitigation Implemented] [Priority: High]**

Category: Spoofing

Description: Non-Relational Database may be spoofed by an attacker and this may lead to data being written to the attacker's target instead of Non Relational Database. Consider using a standard authentication mechanism to identify the destination data store.

Justification: To ensure data integrity, the MongoDB and NodeJS backends were linked via a token string with BSON-type encrypted data format. Every database call will be validated and encrypted.

#### **15. Data Store Denies Non-Relational Database Potentially Writing Data [State: Not Applicable] [Priority: High]**

Category: Repudiation

Description: Non-Relational Database claims that it did not write data received from an entity on the other side of the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.

Justification: MongoDB and NodeJS backends were linked via a token string with BSON-type encrypted data format.

#### **16. Potential Excessive Resource Consumption for Backend Web Server or Non-Relational Database [State: Needs Investigation] [Priority: High]**

Category: Denial Of Service

Description: Does Backend Web Server or Non-Relational Database take explicit steps to control resource consumption? Resource consumption attacks can be hard to deal with, and

there are times that it makes sense to let the OS do the job. Be careful that your resource requests don't deadlock, and that they do timeout.

Justification: <no mitigation provided>

#### **17. Data Flow NoSQL Is Potentially Interrupted [State: Mitigation Implemented] [Priority: High]**

Category: Denial Of Service

Description: An external agent interrupts data flowing across a trust boundary in either direction.

Justification: MongoDB and the backend server (NodeJS) are connected with a BSON-type encrypted stream, but this need to be considered more complex way.

#### **18. Data Store Inaccessible [State: Needs Investigation] [Priority: High]**

Category: Denial Of Service

Description: An external agent prevents access to a data store on the other side of the trust boundary.

Justification: <no mitigation provided>

## **Summary of Threat Model**

### Threat Model Summary:

Not Started	0
Not Applicable	14
Needs Investigation	2
Mitigation Implemented	18
Total	34
Total Migrated	0

# Testing

## Test Plan

### Auth Middleware

1. Protect – Check the JWT token and user exists (Authenticated Any User)
2. Admin – Check the User Role is admin
3. Worker – Check the user role is worker
4. Manager – Check the user role is manager
5. None – Public Access

API Endpoints	Functionality	Method	Middleware
/api/users/login	User login	Post	None
/api/users	Admin Create User	Post	Protect, Admin
/api/users/	Admin get all users	Get	Protect, Admin
/api/users/:id	Get single user data	Get	Protect
api/users/token/: token	Validate user token	Get	None
api/users/all	Get all users for send message	Get	Protect
api/messages	Create new message	Post	Protect
api/messages/sent/:id	Get all sent messages by user id	Get	Protect
api/messages/received/:id	Get all received messages by user id	Get	Protect
api/documents	Upload document	Post	Protect, Admin, Manager
api/documents	View all documents	Get	Protect, Admin, Manager
api/documents/download/:docID/:token	Download document	Get	Protect, Admin, Manager

## Implementation

## Implementation

```

22 describe("Post /api/users/", () => {
23   it("should admin create user", async () => {
24     const res = await instance.post(
25       API_Endpoint + "users/",
26       {
27         name: "Test User",
28         username: "test",
29         email: "test@gmail.com",
30         password: "111111",
31         type: "Worker",
32       },
33       {
34         headers: {
35           Authorization:
36             "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYzMmRjZjgyZGVkMWVlbnJhLnNmCjNNSiIsImhhdCI6MTY2ODEyMzMyZWwiLCJ0eXAiOiJKdWwifQ==",
37         },
38       }
39     );
40     expect(res.status).toBe(201);
41   });
42 });
43
44 describe("GET /api/users/", () => {
45   it("should admin get all users", async () => {
46     const res = await instance.get(API_Endpoint + "users/", {
47       headers: {
48         Authorization:
49           "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYzMmRjZjgyZGVkMWVlbnJhLnNmCjNNSiIsImhhdCI6MTY2ODEyMzMyZWwiLCJ0eXAiOiJKdWwifQ==",
50       },
51     });
52     expect(res.status).toBe(200);
53   });
54 });

```





## Results

```
> backend@1.0.0 test D:\SLIIT\Year 4 Semester 2 - 2022\Secure Software Development - SE4030\All Practicals\SSD Assignment 2\Code\Backend
> jest

PASS test/api.test.js (5.015 s)
  POST /api/users/login
    ✓ should login user (513 ms)
  POST /api/users
    ✓ should admin create user (726 ms)
  GET /api/users/
    ✓ should admin get all users (309 ms)
  GET /api/users/:id
    ✓ should auth user get user data by id (295 ms)
  GET /api/users/:token
    ✓ should any user can validate token (155 ms)
  GET /api/users/all
    ✓ should auth user get all users for messaging (298 ms)
  GET /api/messages/sent/:id
    ✓ should worker get sent messages (469 ms)
  GET /api/messages/received/:id
    ✓ should worker get received messages (477 ms)
  POST /api/messages/
    ✓ should worker send message (305 ms)
  GET /api/documents
    ✓ should manager get all documents (460 ms)
  GET /api/documents/download/:id/:token
    ✓ should manager view document (336 ms)

Test Suites: 1 passed, 1 total
Tests: 11 passed, 11 total
Snapshots: 0 total
Time: 5.167 s
Ran all test suites.
```



## Codebase

Frontend: [https://github.com/Sandun01/SSD\\_Assignmnet2\\_Frontend.git](https://github.com/Sandun01/SSD_Assignmnet2_Frontend.git)

Backend: <https://github.com/rashini123/SSD-Assignment2.git>