Project Report



# Index

[Index 2](#_27yxas7g3svt)

[1. Introduction 5](#_p2edqmyo4yas)

[1.1. Project Info 5](#_uldm77htz4v1)

[2. System definition 7](#_a7vi9tfexpjj)

[2.1. System Architecture 7](#_92rcsyhy3lh)

[2.2. System Components 7](#_4g3ion9hueeb)

[2.3. Restrictions 8](#_vzorxbuzcqf0)

[3. Requirement 9](#_lmfmlc2kpait)

[3.1. Customer Requirement 9](#_37dane8ntujy)

[3.2. Functional Requirements Decomposition 9](#_presu3bofor3)

[3.3. Non-functional requirements Decomposition 11](#_p8swhcfjzw22)

[4. Threat Modeling 12](#_ww3km747twpy)

[4.1. DFD 12](#_ajyx8ercfxzi)

[4.2. PnG 13](#_vx549q6bx1bu)

[5. Risk Analysis 15](#_uyeg44vfh681)

[5.1. Derived Threats from Model 15](#_awum5fs6l7k6)

[5.2. Supply-Chain Risk Management Plan 15](#_uo4yhrgdxqb4)

[6. Risk Assessment 20](#_xt07p2f5pspq)

[7. Implementation 23](#_uqq3y3vypj1x)

[7.1. Login Interface 23](#_a72cac4kk15v)

[7.1.1. Requirements 23](#_jadu2iurlnzw)

[7.1.2. User Authentication Server 23](#_d145z1lxpky)

[7.1.3. Multi-Factor Authentication 27](#_f4q2o4djn60r)

[7.2. Secure Logger 28](#_5krtppzf68o6)

[7.2.1. Functionality requirements 28](#_bwreh271t5qo)

[7.2.2. Rationale 28](#_jkyb8uoqrqmd)

[7.2.3. Design 29](#_3hggnxrf0aus)

[7.2.4. Limitations & Considerations 29](#_wl58zf72ox3p)

[7.2.5. Next Steps 30](#_stydn750j2d2)

[7.3. Secure Recording data file 30](#_w01zge3zldei)

[7.3.1. Functionality requirements 30](#_lp8gtv4xhcgg)

[7.3.2. Rationale 30](#_me0tzi75ux21)

[7.3.3. Design 31](#_vvpqszun12ed)

[7.3.4. Limitations & Considerations 31](#_b5ypkd5rp2a2)

[7.3.5. Next Steps 31](#_un855tpsdqub)

[7.4. TLS protocol 32](#_d2s8vh236qw5)

[7.4.1. Display (log-in) and DB server 32](#_sq2mnideu0zd)

[7.4.2. PI and Display App 33](#_450ppyswx2gy)

[8. Security Review 36](#_7xd60jkmbc7j)

[8.1. Not mitigated risks 36](#_yaijchlrpeid)

[8.2. Inspection of Mitigations 36](#_robs0bcycgk)

[8.2.1. Mitigated risks 36](#_56rokgoarzsd)

[8.2.2. Data transfer View 36](#_hfe5ufrpsm75)

[8.2.3. Attack surface view 39](#_ptwggtvkik84)

[8.2.4. Data Storage View 39](#_ncyr32n48sl6)

# Introduction

## Project Info

Team name: AimHigh

Project name: The Flight Agent System  
Team member :   
 - Phuong Tuan Pham  
 - Haesung Kim  
 - Jihye Ahn  
 - Jongseok Jung  
 - Junwoo Mun  
 - Yeonseok Kang

Project Term: 2025.05.21 ~ 2025.06.06

Roles and Schedule

| **Owner** | **Job List** | **5/21** | **5/22** | **5/23** | **5/24** | **5/25** | **5/26** | **5/27** | **5/28** | **5/29** | **5/30** | **5/31** | **6/1** | **6/2** | **6/3** | **6/4** | **6/5** | **6/6** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [Jihye Ahn](mailto:jihyea@andrew.cmu.edu) | Set Pi running environment | O | O |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| [Junwoo Mun](mailto:junwoom@andrew.cmu.edu)  [Haesung Kim](mailto:haesung2@andrew.cmu.edu)  [Phuong Tuan Pham](mailto:ppham@andrew.cmu.edu) | Setup Display app environment | O | O | O | O | O |  |  |  |  |  |  |  |  |  |  |  |  |
| [Yeonseok Kang](mailto:yeonseok@andrew.cmu.edu)  [Jongseok Jung](mailto:jongseoj@andrew.cmu.edu) | Setup Big Query | O | O | O | O | O | O | O | O |  |  |  |  |  |  |  |  |  |
| [Haesung Kim](mailto:haesung2@andrew.cmu.edu) | Define System Architecture |  |  |  |  |  |  | O | O |  |  |  |  |  |  |  |  |  |
| [Haesung Kim](mailto:haesung2@andrew.cmu.edu) | Requirement engineering |  |  |  |  |  |  | O | O |  |  |  |  |  |  |  |  |  |
| [Jihye Ahn](mailto:jihyea@andrew.cmu.edu) | Develop Login UI |  |  |  |  |  |  | O | O | O | O |  |  |  |  |  |  |  |
| [Haesung Kim](mailto:haesung2@andrew.cmu.edu) | Design System structure |  |  |  |  |  |  |  | O | O |  |  |  |  |  |  |  |  |
| [Junwoo Mun](mailto:junwoom@andrew.cmu.edu) | Threat Modeling |  |  |  |  |  |  |  | O | O |  |  |  |  |  |  |  |  |
| [Junwoo Mun](mailto:junwoom@andrew.cmu.edu) | Analyze and Assess Threats |  |  |  |  |  |  |  |  | O | O |  |  |  |  |  |  |  |
| [Junwoo Mun](mailto:junwoom@andrew.cmu.edu) | Define Risk and find mitigation |  |  |  |  |  |  |  |  | O | O |  |  |  |  |  |  |  |
| [Phuong Tuan Pham](mailto:ppham@andrew.cmu.edu) | Implement CSV encrypt system |  |  |  |  |  |  |  |  |  | O | O | O | O | O |  |  |  |
| [Phuong Tuan Pham](mailto:ppham@andrew.cmu.edu) | Implement logging system |  |  |  |  |  |  |  |  | O | O | O | O | O | O |  |  |  |
| [Jihye Ahn](mailto:jihyea@andrew.cmu.edu) | Implement Login system |  |  |  |  |  |  |  |  |  | O | O | O | O | O |  |  |  |
| [Yeonseok Kang](mailto:yeonseok@andrew.cmu.edu)  [Jongseok Jung](mailto:jongseoj@andrew.cmu.edu)  [Junwoo Mun](mailto:junwoom@andrew.cmu.edu) | Apply TLS protocol |  |  |  |  |  |  |  |  | O | O | O | O | O | O |  |  |  |
| [Haesung Kim](mailto:haesung2@andrew.cmu.edu) | Implement 2-factor auth |  |  |  |  |  |  |  |  | O | O | O | O | O | O |  |  |  |
| [Junwoo Mun](mailto:junwoom@andrew.cmu.edu) | Prepare presentation |  |  |  |  |  |  |  |  |  |  |  |  |  | O | O | O | O |
| All | Write User Guide & TC & demo |  |  |  |  |  |  |  |  |  |  |  |  | O | O | O | O | O |
| All | Review |  |  |  |  |  |  |  |  |  |  |  |  |  | O | O | O | O |

# System definition

## System Architecture

The flight agent system is a system that collects radio signals from aircraft to compose a real-time global flight map.

The default system looks like below

|  |
| --- |

## System Components

1. Feeder  
    - Collect radio signals from aircraft and send them to the remote component  
    - **Raspberry Pi** is a feeder in our system.
2. Hub  
    - Receive data from feeders and compose real-time global flight map  
    - **ADS-B Hub** is a hub in our system.  
    - User can setup local custom hub
3. Display App  
    - Parse aircraft data and visualize it into 2D map, and provide user interface
4. Cloud database  
    - support file saving and file   
    - **Google Big Query** is a cloud database in our system

## Restrictions

* ADS-B Hub checks IP (or key) of feeders and only accepts data from registered feeders
* ADS-B Hub only provides data to accounts with active feeders.
* Big Query checks json key so that authorized users can access the database

# Requirement

## Customer Requirement

* Functional requirement :

1) Develop password-protecting extension

* Non-functional requirement :   
   1) must be secured, high-quality  
   2) error detection, recovery, and reporting

## Functional Requirements Decomposition

* Develop password-protecting extension

1. Login interface
   1. Create login screen
   2. Get User name
   3. Get Password
   4. Get Email for 2-way auth
   5. Confirm login scenario

<Sequence Diagram>

| Login System gets ID/Password from DisplayApp  Login System sends Query to DB  Login System decides success or fail from Query response  Display App sends Email for 2-way auth  DIsplay App verify 2-way auth |
| --- |

1. Login system with DBMS
   1. Create user-registration UI
   2. Register User with Username, password and Email
2. Multi-factor authentication
   1. Use Email verification for 2-factor authentication

## Non-functional requirements Decomposition

* Must be secured, high-quality

To make the flight system reliable, we put Integrity the most important.

1. No Spoofing
2. No Tampering

The next important point was confidentiality which is a new requirement.

1. No Data revealing

* Error detection, recovery, and report

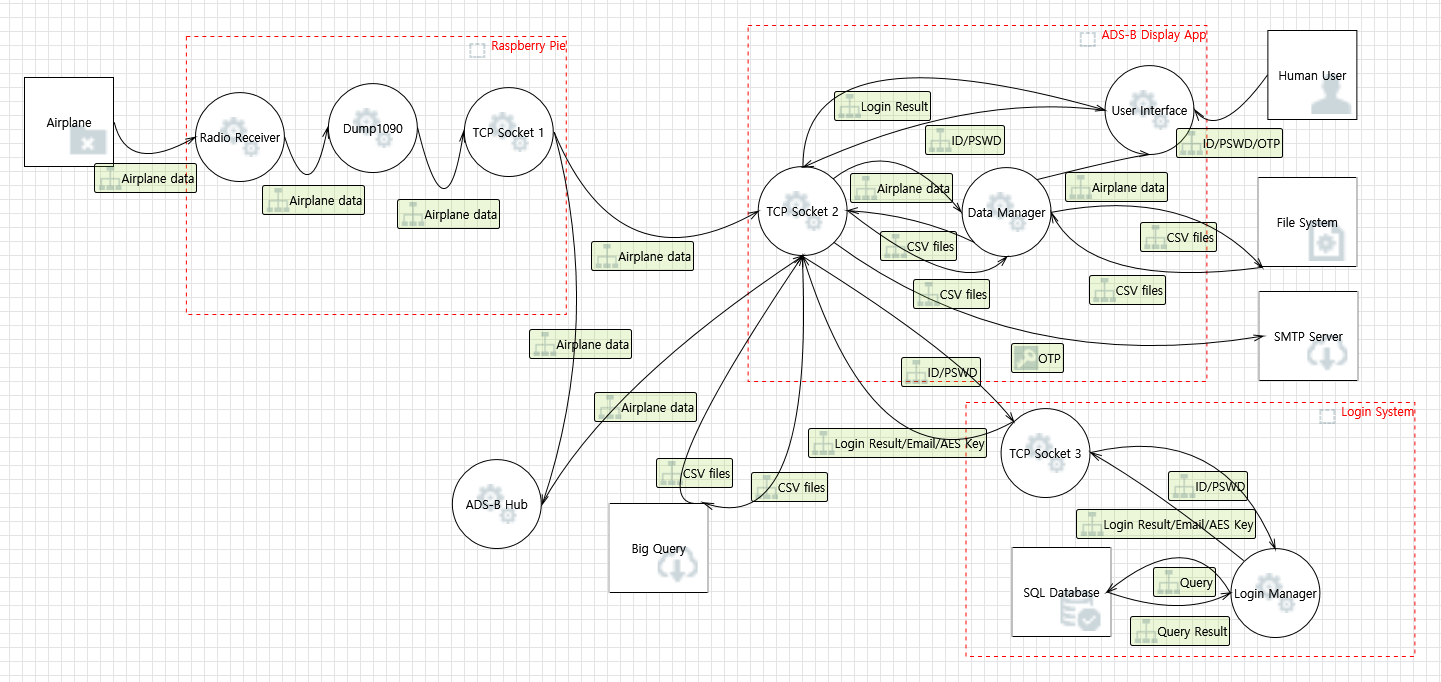
To detect errors, we found that a log saving system is needed.

1. Needs Log

# Threat Modeling

## DFD

With detailed requirements, we composed the system, and the structure of the system can be shown in DFD like below



With the diagram, we could identify the attack surfaces and assets

<Attack Surface>

| Attack surface | Owner | Detail | Usage |
| --- | --- | --- | --- |
| TCP Socket | Raspberry Pi | TCP port 30001 | Send feeding data to hub |
| TCP Socket | Raspberry Pi | TCP port 30002 | Not used |
| TCP Socket | Raspberry Pi | TCP port 30003 | Not used |
| TCP Socket | Display App | TCP port 5002 | Receive data from Pi |
| TCP Socket | Display App | TCP port 30002 | Receive data from hub |
| TCP socket | Display App | TCP port 587 | SMTP |
| Https connection | Display App | HTTPS port 443 | HTTPS |
| Https connection | Login System | HTTPS port 443 | HTTPS |

<Assets>

| Data | Sender | Receiver | Method |
| --- | --- | --- | --- |
| Airplane Data | Airplane | Radio Receiver | Radio |
| Airplane Data | Raspberry Pi | ADS-B Hub | TCP |
| Airplane Data | Raspberry Pi | Display App | TCP |
| Airplane Data | ADS-B Hub | Display App | TCP |
| CSV file | Display App | File System | IPC |
| CSV file | File System | Display App | IPC |
| CSV file | Big Query | Display App | TCP |
| CSV file | Display App | Big Query | TCP |
| Queries | Login System | DBMS | TCP |
| Query response | DBMS | Login System | TCP |
| ID/Password | Display App | Login System | https |
| Key for AES | Display App | Login System | https |
| Email | Login System | Display App | https |
| OTP | Login System | Email Server | SMTP |

## PnG

We also used PnG method for threat modeling and derived some threats

| Persona | Terrorist | Malicious retired man |
| --- | --- | --- |
| Motivations | Make airplane change course for hijacking. | Malicious intentions towards the control system company for unfair dismissal. |
| Goals | Change airplane to change course to avoid collision with a fake airplane. | Deterioration of the reliability and reputation of the control system. |
| Skills | Can send multiple virtual airplane radio data to ADS-B hub. | Can change ADS-B Hub's broadcasting data to be randomized. |
| Misuse case | A terrorist can set up multiple feeders and send fake information to the hub. | A retired man can ruin data that ADS-B Hub is providing to discredit the company. |
| Mitigation | Hub needs to validate the data feeder and check the data is reliable | Display Application should verify flight data's feasibility in application level |

# Vulnerability Analysis

## Derived Threats from Model

With the DFD, we used Microsoft Threat Modeling Tool to elicit threats.

|  |
| --- |

Among **88 threats**, **21 mitigated**, **36 not applicable**, and **31 need to be investigated, threats** are detected.

We also found 2 threats from PnG method, so total **90 threats** were found in our system.

## Supply-Chain Risk Management Plan

* List of Supply Chain

We defined components of the supply chain.

Hardware elements and 3rd party programs are already fixed in this project, so we focused on libraries that we can choose.

| **#** | **Name & version** | **Type** | **Category** | **Selected reason** |
| --- | --- | --- | --- | --- |
| 1 | Software Defined Radio | Hardware | 3rd party | By requirements |
| 2 | Raspberry Pi 5 | Hardware | 3rd party | By requirements |
| 3 | dump1090 | Linux Application | opensource | By requirements |
| 4 | ASD-B Client | Linux Application | opensource | By requirements |
| 5 | ADS-B-Display | Window Application | Reused software | By requirements |
| 6 | BigQuery | Cloud database system | COTS | By requirements |
| 7 | WinCrypt | Encryption Library | opensource | Selected by A-SQUARE |
| 8 | Openssl | Cryptography / TLS library | opensource | Selected by A-SQUARE |
| 9 | Gmail | SMTP Service | COTS | Selected by A-SQUARE |
| 10 | Database | SQLcipher | opensource | Selected by A-SQUARE |
| 11 | Flask | Web Framework | opensource | Selected by A-SQUARE |

* Comparison based on A-SQUARE

1. COTS Software
   1. SMTP service  
      **Evaluate**

* Gmail:
* Official Google-maintained and supported
* Free account
* Secure software development methods are applied
* One of the biggest Email Clients in the world
* Outlook
* Official Microsoft-maintained and supported
* Free account
* Secure software development methods are applied
* Less popular than Gmail

**Decision**

We chose Gmail because Gmail has more documentation to follow and is easy to set up.

1. **Open Source Software**
   1. Database  
      **Evaluate**

* SQLite
* SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured SQL database engine.
* Official document
* Large community with active developers, widely trusted by developers
* SQLcipher
* a standalone fork of the SQLite database library that adds 256-bit AES encryption of database files and other security features.
* Active developers, widely trusted by developers.
* Official document

**Decision**

We chose SQLcipher because SQLcipher provides encryption and signing algorithm that we needed. SQLCipher is superior from the perspectives of security and risk mitigation.

* 1. Web-framework  
     **Evaluate**
* Flask
* Large community with active developers, widely trusted by developers
* Ideal for lightweight APIs or single-page apps
* Django
* Large community with active developers, widely trusted by developers
* Slightly larger global adoption for full-stack sites

**Decision**

We chose Flask because Flask is more lightweight and allows direct control of DB. Also, Django is powerful, but has excessive configuration restrictions for secure extended DBs.

* 1. Cryptography / TLS library  
     **Evaluate**
* OpenSSL
* Large community with active developers, widely trusted by developers
* Well-documented
* Sufficient feedback from users
* LibreSSL
* Small community with fewer active developers
* insufficient documentation
* Limited feedback users

**Decision**

We choose OpenSSL because it is more widely used and has various use cases, and sufficient experience from other developers. And OpenSSL looks much more fast-interactive with CVE.

* 1. Encryption library

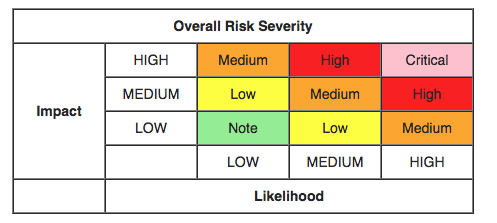
**Evaluate**

* WinCrypt
* Based on MSDN
* Less documentation
* Specialized in windows application
* Built-in to Windows apps
* Few CVE
* Included in the Windows authentication ecosystem
* Libsodium
* Brief and intrinsic documentation
* Adoption in next-generation security-oriented services
* Good feedback
* Several CVEs exist, attack response is fast

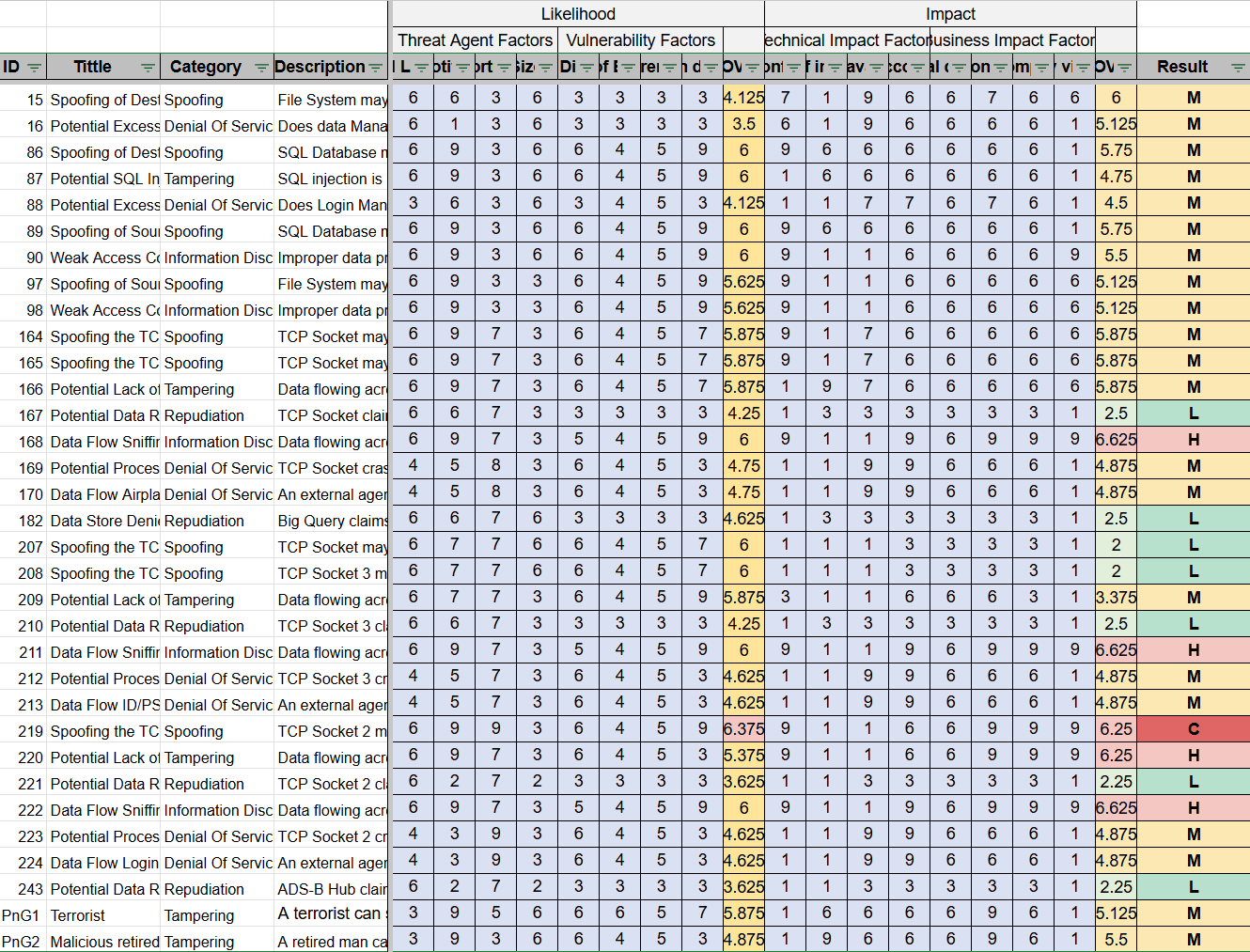
**Decision**

We choose WinCrypt because it is more stable and compatible with Windows applications. And also verified by Microsoft.

# Risk Assessment

From the threat we defined, OWASP risk assessment framework to rank the threat and select risks.  
<OWASP risk rating methodology>  


The threats we discovered are rated as **4 high, 1 critical, 21** **medium and 7 low**.



So We considered a total of **26 threats as risks**: critical, high, and medium severity threats.

Considering time and resources, we designed and implemented mitigation measures for **critical** and **high** scored risks first, and some **medium** scored risks.

|  | |
| --- | --- |
| ID (score) | 164(M), 165(M), 166(M), 168(H) |
| Category | Spoofing, tampering |
| Mitigation method | TLS with 1-way authentication using server-side certificates |

|  | |
| --- | --- |
| ID (score) | 209(M), 211(H), 219(C), 220(H), 222(H) |
| Category | Information Disclosure, Tampering, Spoofing |
| Mitigation method | TLS with 1-way authentication using server-side certificates |

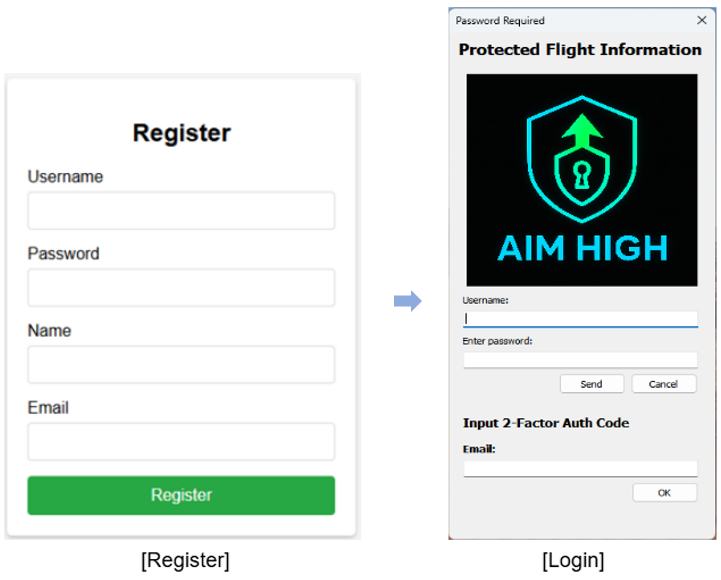
|  | |
| --- | --- |
| ID (score) | 15(M), 97(M), 98(M) |
| Category | Spoofing, Information Disclosure |
| Score | M |
| Mitigation method | Encrypt Data before saving it in the file system. And also save hash of the data to guarantee its integrity |

# 

# 

# Implementation

## Login Interface



### Requirements

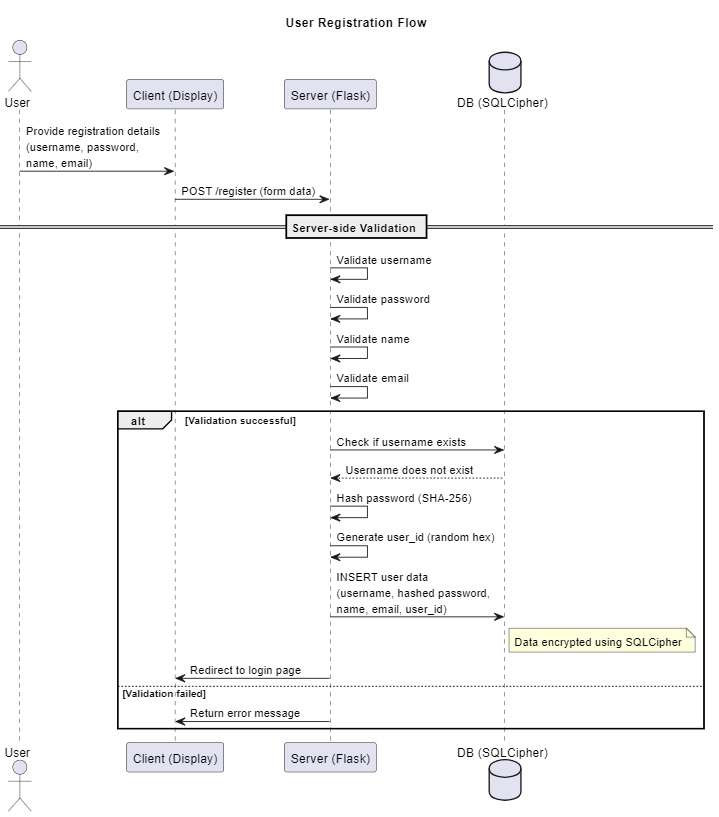
The primary objective is to ensure the confidentiality, integrity, and availability of critical data, particularly in the context of flight monitoring and related telemetry. To enforce access control, a password authentication mechanism will be implemented for monitored flights. Users will be required to authenticate prior to accessing or viewing flight information, thereby preventing unauthorized data exposure. The authentication process will be integrated into the GUI and securely linked with the backend system.

### User Authentication Server

* Objective

-To protect user credentials by using SQLCipher’s AES-256 encryption, even if the database file is leaked.  
-To securely store hashed and salted passwords in an encrypted SQLite database, reducing the risk of credential theft.  
-To maintain the integrity and confidentiality of authentication data by applying full-database encryption and key-based access control.

* Design(Registration)



* User Input Validation  
  The user submits registration details (username, password, name, email) via the /register endpoint. The server validates the input fields for correctness (e.g., username length, password complexity, email format).
* Password Hashing

The server hashes the user's password using SHA-256 to ensure secure storage. A unique user\_id is generated using a random 32-byte hex value.

* Database Check

The server queries the database to check if the username already exists.

If the username is found, the server aborts the process with a conflict error.

* Data Storage

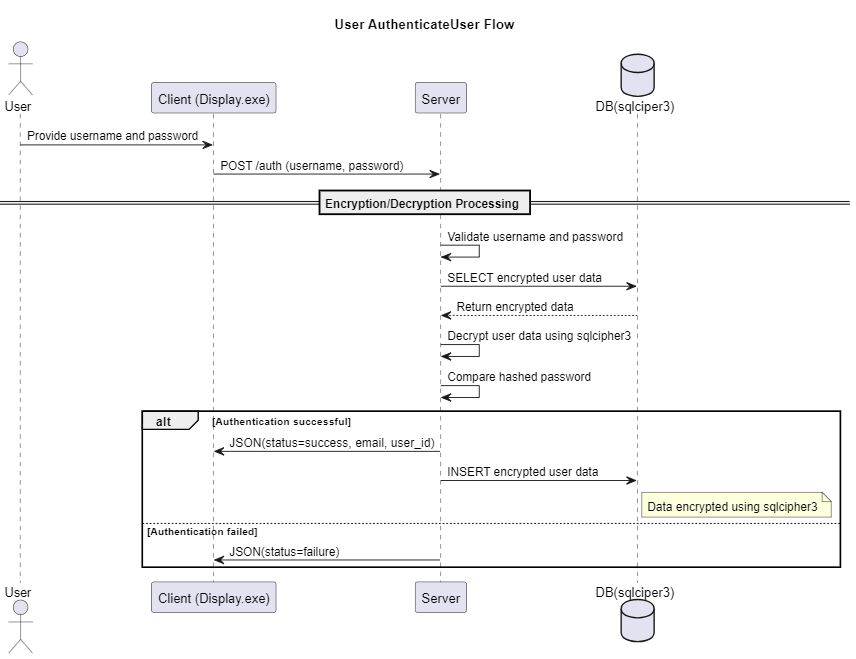
The server inserts the validated and hashed user data (username, hashed password, name, email, user\_id) into the SQLCipher-encrypted database. The data is securely stored using AES-256 encryption.

* Response

Upon successful registration, the user is redirected to the login page.

If validation fails, an appropriate error message is returned.

* Design(User Authentication)



* User Authentication Request  
   The client application sends a POST request containing the user's credentials (username and password) to the authentication server.
* Encrypted Data Retrieval from Database The server queries the SQLite database (secured with SQLCipher) to retrieve the encrypted user data associated with the provided username.
* Decryption and Credential Verification The server decrypts the retrieved user data using SQLCipher and compares the hashed input password against the stored password hash to validate the credentials.
* Secure Data Storage Upon Success Upon successful authentication, essential user attributes (e.g., email, user ID) are stored or updated in the database in encrypted form using SQLCipher.

### Multi-Factor Authentication

To strengthen user authentication and safeguard sensitive data, we have   
 implemented **Multi-Factor Authentication (MFA)** in our Flight Agent System.   
 MFA provides an additional layer of security beyond traditional password-based   
 authentication, ensuring that access is granted only to verified users who   
 possess both password and OTP via email.

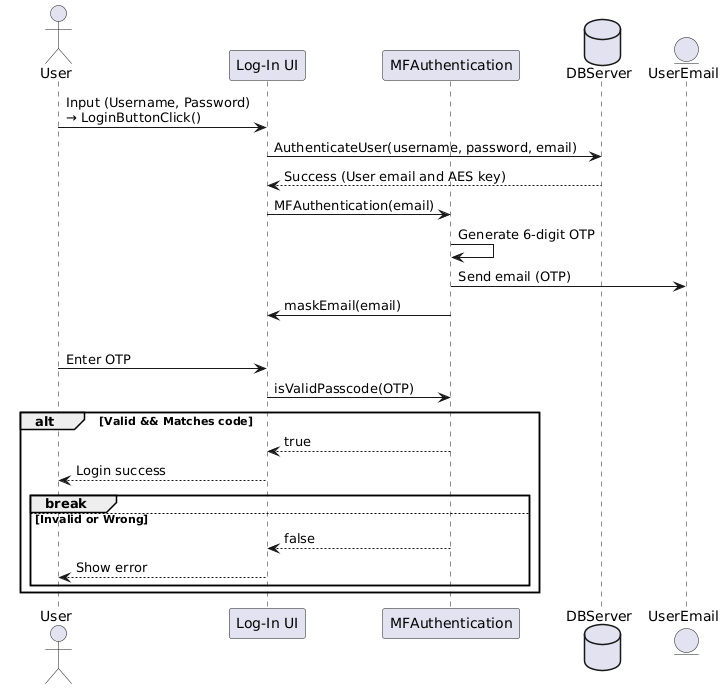
* + - * Objective

- Prevent unauthorized access due to compromised passwords

- Mitigate risks related to phishing, brute-force, and credential stuffing attacks

- Comply with modern security best practices and compliance standards

* + - * Design



* + - * Authentication Workflow
        + Credential Verification

- The user enters their username and password.

- These credentials are sent via HTTPS to a backend authentication server.

- If valid, the server responds with a registered email and a temporary AES encryption key.

* + - * OTP Delivery

- A 6-digit one-time passcode (OTP) is generated on the client.

- The OTP is sent to the user’s registered email using a secure SMTP connection (TLS-enabled).

* + - * + OTP Verification

- The user enters the received OTP.

- The client validates the OTP format and compares it with the one previously generated.

- Upon success, access to the system is granted.

* + - * + Security Considerations

- The OTP is time-based and expires shortly after issuance

## Secure Logger

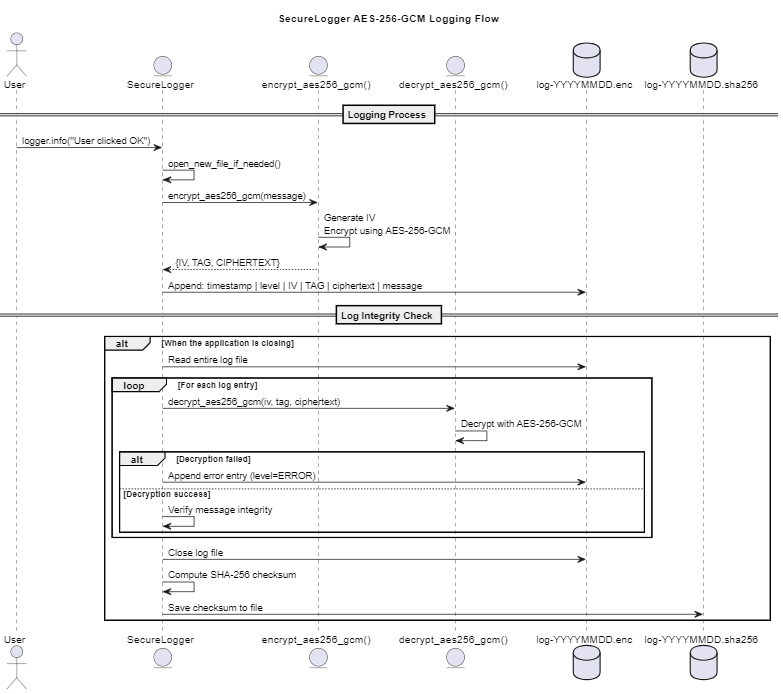
### Functionality requirements

* Secure Logger feature secures the log file content, preventing the log file from being modified.
* The log file will be encrypted line by line to make sure the line cannot be modified.
* The Log file’s checksum is calculated and saved checksum file.
* The User-specific encryption keys are saved on the Login Server
* The AES-256-GCM algorithm is used to encrypt and decrypt the log content.
* The SHA-256 checksum will be computed and stored.

### Rationale

* The log file is required to support non-repudiation.
* All log files and their contents must be encrypted to protect against tampering and ensure data integrity.
* The AES-256-GCM algorithm is chosen to ensure both confidentiality and integrity of the log data.
* The SHA-256 checksum will be computed and stored to ensure the integrity of the log file.
* Log message available in the log file and is not encrypted to make sure the log is human-readable.

### Design



### Limitations & Considerations

Currently, the message content is saved with the log file. Depending on the system concern, the log will or will not be human-readable.

### Next Steps

If the log content should not be human-readable, another software/logic is required to decrypt the log content.

## Secure Recording data file

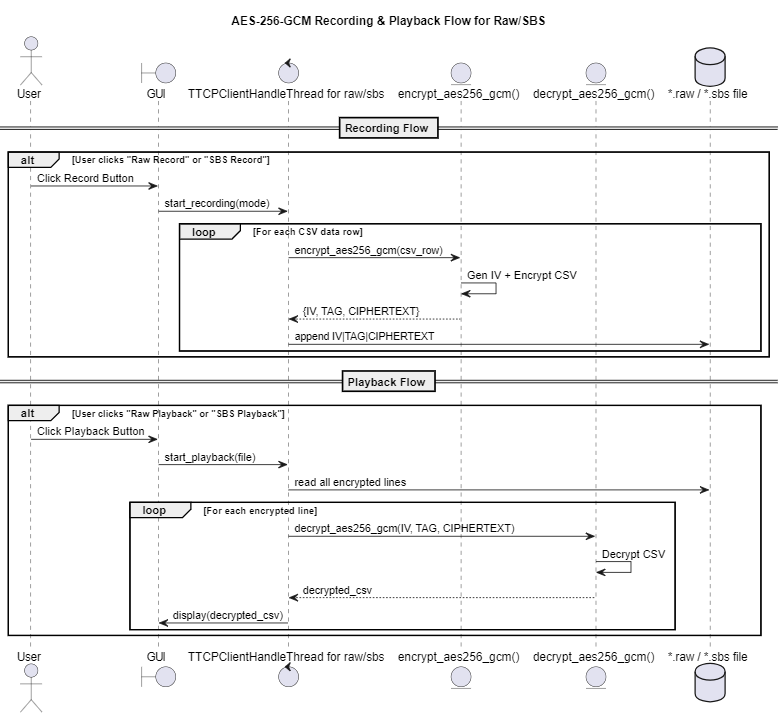
### Functionality requirements

* Secure Recording data file feature secures the recording data file
* The User-specific encryption keys are saved on the Login Server
* The AES-256-GCM algorithm is used to encrypt and decrypt the log content

### Rationale

* All recording data must be encrypted to protect against tampering and ensure data integrity.
* The AES-256-GCM algorithm is chosen to ensure both confidentiality and integrity of the data.

### Design



### Limitations & Considerations

N/A

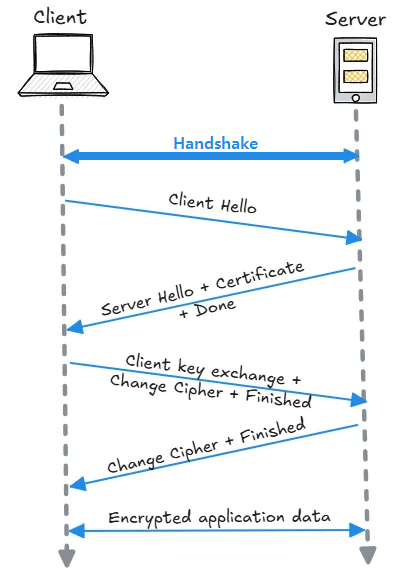
### Next Steps

The log file on BigQuery should be encrypted too.

## 

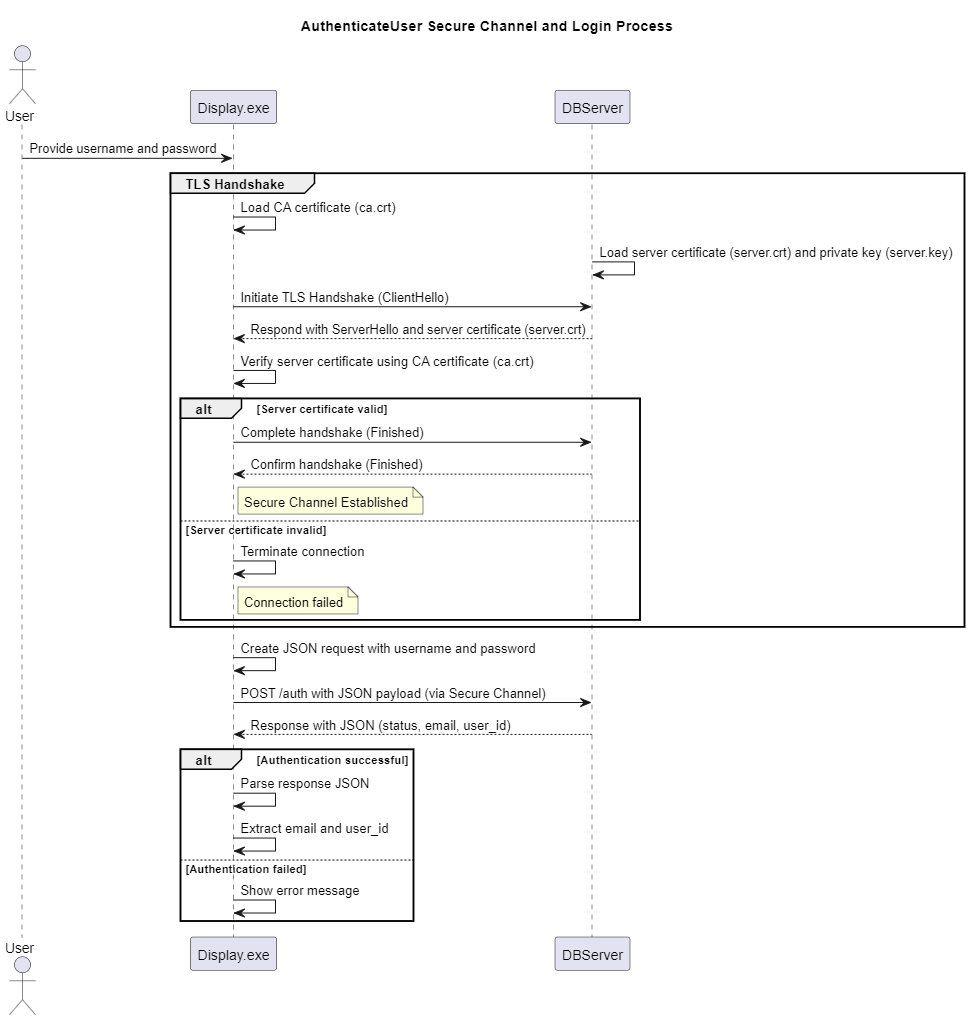
## TLS protocol

In the flight agent system, the TLS (Transport Layer Security) protocol is employed to establish a secure communication channel for the safe transmission of sensitive data, including flight information and personal authentication credentials. TLS ensures data confidentiality, integrity, and authentication between communicating entities over an untrusted network.



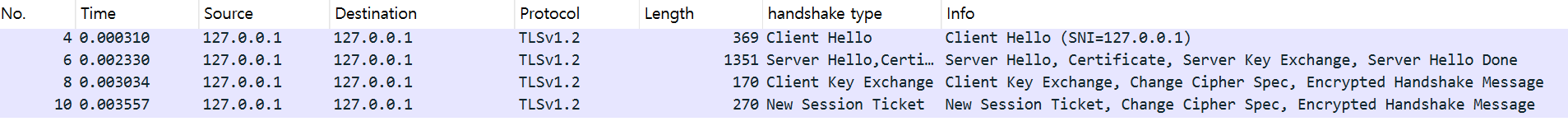
### Display (log-in) and DB server

* + - * Requirement  
        All user authentication data exchanged between the login application (client) and the authentication server must be transmitted over a secure channel.
      * Mitigation Strategy  
        The client application (Display) transmits user login credentials to the server (DB Manager) over a secure, encrypted channel utilizing the HTTPS protocol. A secure communication path is initially established via a TLS handshake(TLSv1\_2), ensuring confidentiality and integrity of the connection. Upon successful negotiation of the TLS session, authentication data is transmitted securely through the established channel.
      * Implementation

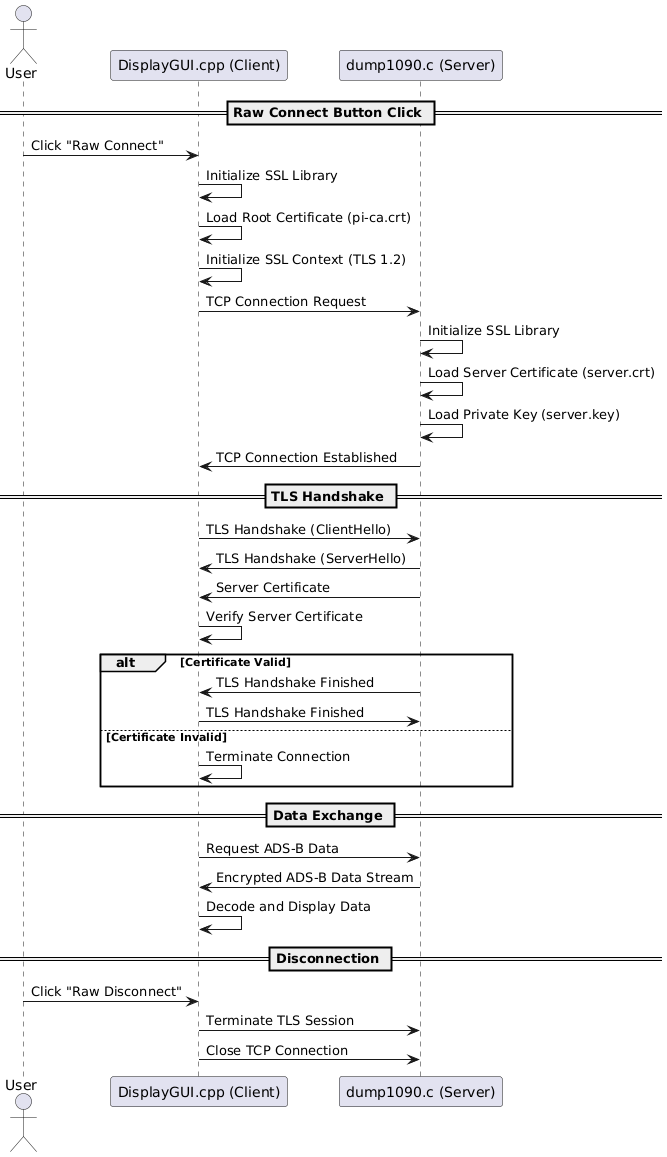


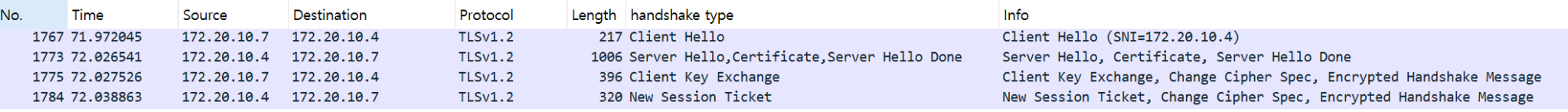
* + - * Inspection

By Wireshark tool, we could see handshake process between client and server



### PI and Display App

* + - * Requirement  
        The communication channel between the Raspberry PI and the Display App shall ensure the secure transmission of flight-related data. The system must guarantee data confidentiality, integrity, and authenticity during transit to prevent unauthorized access, tampering, or spoofing. The architecture shall support secure message validation and enforce strong identity assurance mechanisms between the endpoints.
      * Mitigation Strategy  
        To prevent spoofing attacks on the TCP socket and ensure that only authorized processes can establish connections, the system must implement TLS (Transport Layer Security) for all communications involving airplane data. The TLS layer shall authenticate the remote peer via X.509 certificate validation and establish a secure encrypted channel to maintain data confidentiality and integrity.
      * Implementation  
        
      * Inspection  
        By Wireshark tool, we could see handshake process between client and server



# Security Review

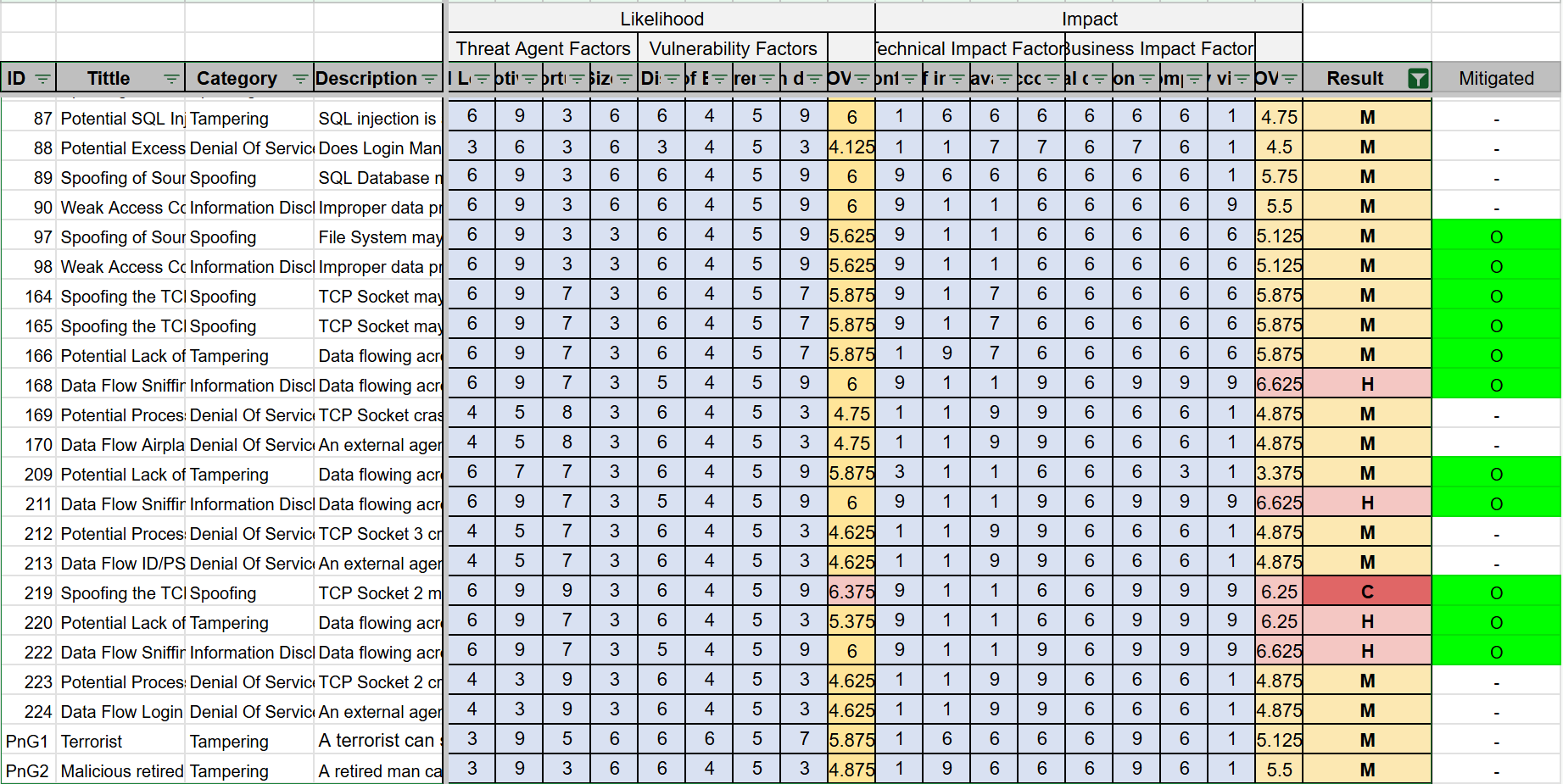
## Not mitigated risks

* We considered Confidentiality and Integrity more than Availability. So we didn’t focus on mitigating availability risks like “Denial of service”.  
  We could add IP address detection and blocking logic, or add timeout spec, log-based DoS detecting logic to defend against DoS.
* We focused on attack surface so data communication in trust boundary is not prioritized target for mitigation
* We could add data anomaly detection logic so that we could defense data corruption from ADS-B Hub or Pi (risk derived from PnG)

## Inspection of Mitigations

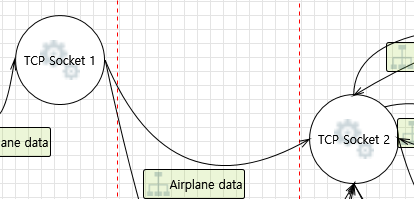
### Mitigated risks

**11 out of 26 risks** are mitigated. All the Critical and High risks are mitigated.



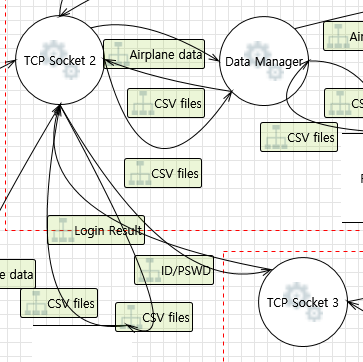
### Data transfer View

1) Data between Pi and Display Application



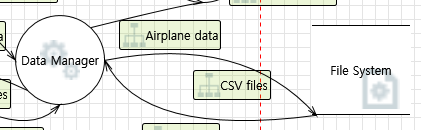
| Risks | Spoofing | Tampering | Repudiation | Information Disclosure | Denial of Service | Elevation of Privilege |
| --- | --- | --- | --- | --- | --- | --- |
| Status | Mitigated | Mitigated | Mitigated | Mitigated | - | - |
| Method | Authentication with Certificate | Authentication with Certificate | Logging | Channel Encrypting | - | - |

2) Data between Display Application and Login System



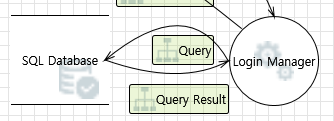
| Risks | Spoofing | Tampering | Repudiation | Information Disclosure | Denial of Service | Elevation of Privilege |
| --- | --- | --- | --- | --- | --- | --- |
| Status | Mitigated | Mitigated | Mitigated | Mitigated | - | - |
| Method | Authentication with Certificate | Authentication with Certificate | Logging | Channel Encrypting | - | - |

3) Data between Display App and File System



| Risks | Spoofing | Tampering | Repudiation | Information Disclosure | Denial of Service | Elevation of Privilege |
| --- | --- | --- | --- | --- | --- | --- |
| Status | Mitigated | Mitigated | Mitigated | Mitigated | - | - |
| Method | Authentication | Authentication | Logging | Channel Encrypting | - | - |

4) Data between DBMS and Login System



| Risks | Spoofing | Tampering | Repudiation | Information Disclosure | Denial of Service | Elevation of Privilege |
| --- | --- | --- | --- | --- | --- | --- |
| Status | Mitigated | Mitigated | - | Mitigated | - | - |
| Method | Authentication | Authentication | - | Channel Encrypting | - | - |

### Attack surface view

| Attack surface | Owner | Detail | Result |
| --- | --- | --- | --- |
| TCP Socket | Raspberry Pi | TCP port 30001 | 1-way authentication TLS applied |
| TCP Socket | Display App | TCP port 5002 | Not Applicable (ADS-B hub only supports raw socket connection) |
| TCP Socket | Display App | TCP port 30002 | 1-way authentication TLS applied |
| TCP socket | Display App | TCP port 587 | Use SMTP protocol. safe |
| Https connection | Display App | HTTPS port 443 | Use https protocol. safe |
| Https connection | Login System | HTTPS port 443 | Use https protocol. safe |

### Data Storage View

| Data | Stored | Storage | Confidentiality | Integrity |
| --- | --- | --- | --- | --- |
| Airplane Data | As CSV | In window file system | Encrypted in AES  (AES-GCM) | Use MAC to verify (AES-GCM) |
| ID/Password | As DB | In DB | Encrypted in AES (AES-GCM) | Use MAC to verify (AES-GCM) |
| Email | As DB | In DB | Encrypted in AES (AES-GCM) | Use MAC to verify (AES-GCM) |
| AES key | As DB | In DB | Encrypted in AES (AES-GCM) | Use MAC to verify (AES-GCM) |

## 