Security Report

TrendTrack Design Co.

Date	Jan 13, 2024
Author	Ivet Kalcheva
Status	Completed (Version 1.0)

Version History

Version	Date	Author(s)	Changes	State
0.5	05/01/25	Ivet Kalcheva	Added the security risk table	Draft
1.0	13/01/25	Ivet Kalcheva	Added reasoning	Completed

Distribution

Version	Date	Receivers
0.5	08/01/25	Frank Coenen and Bart Rabeling
1.0	17/01/25	Frank Coenen and Bart Rabeling

Contents

Introduction

Document Purpose

This document provides an analysis of potential security risks in the **TrendTrack** application based on the **OWASP Top 10 (2021)**. Each risk is evaluated with respect to its **Likelihood**, **Impact**, **Risk Level**, and the **Actions Planned** to mitigate it. The aim is to ensure that the TrendTrack platform adheres to best practices in application security, reducing vulnerabilities and protecting sensitive user data, ensuring a secure and seamless user experience.

Security Risk Analysis Table

Security Risk	Likelihood	Impact	Risk Level	Actions Planned
Broken Access Control	Low	High	Critical	Checks to ensure users can only access data, based on user roles and their i.
Injection (SQL Injection)	Low	High	High	Use parameterized queries and ORM frameworks to prevent SQL injection.
Sensitive Data Exposure	Medium	High	High	Encrypt sensitive data (e.g., passwords) and ensure HTTPS is used for data transmission.
Broken Authentication	Medium	High	High	Implement token expiration, and secure authentication methods like JWT.
Function Level Access Control	Medium	Medium	Medium	Ensure all endpoint actions are restricted based on user roles and the data being accessed.
Software and Data Integrity Failures	Medium	High	High	Validate data.
Insecure Design	Low	High	Medium	Validate data.
Using Components with Known Vulnerabilities	Medium	High	High	Regularly update libraries (no action taken).
Insufficient Logging and Monitoring	Low	Medium	Medium	Logging and monitoring for security-related events.
Server-Side Request Forgery (SSRF)	Low	High	Medium	Restrict external HTTP requests (no action taken).

Reasoning

Broken Access Control

Access control ensures that users can only access data within their permitted scope, protecting sensitive information from unauthorised access. I have implemented access control based on user roles (e.g., ADMIN and CLIENT) and identity-based access checks. The current access control restricts access to data, based on the role and specific identity of the user, attempting to access the resources.

Examples:

```
@Override
public void validateUser(Long userId, String token) {
    AccessToken accessToken = accessTokenDecoder.decode(token);
    Long loggedInUserId = accessToken.getUserId();
    List<String> loggedInUserRoles = new ArrayList<>(accessToken.getRoles());

    boolean isAdmin = loggedInUserRoles.contains("ADMIN");

    if (!Objects.equals(userId, loggedInUserId) && !isAdmin) {
        throw AuthException.noAccess();
    }
}
```

```
@PutMapping("{id}")
@RolesAllowed({"ADMIN", "CLIENT"})
public ResponseEntity<Void> updateUser(@PathVariable Long id,
                                       @RequestBody @Valid UpdateUserRequest request,
                                       @RequestHeader("Authorization") String
authorizationHeader) {
    String token = authorizationHeader.replace("Bearer ", "");
    authService.validateUser(id, token);

    request.setId(id);
    userService.updateUser(request);

    return ResponseEntity.noContent().build();
}
```

Injection (SQL Injection)

Using JPA's repository and query mechanism ensures that the input is sanitized and properly handled without exposing the database to injection attacks.

Sensitive Data Exposure

Sensitive data exposure happens when sensitive information, such as passwords, credit card numbers or personal identifiers is stored or transmitted insecurely. To protect sensitive data, all passwords are encrypted using a strong encryption algorithm (e.g., Bcrypt in this case) and stored in the database in their encrypted form. Additionally, we enforce HTTPS for all communications to ensure that sensitive data is transmitted securely over the network.

Broken Authentication

Broken authentication allows attackers to impersonate users by exploiting weak authentication mechanisms. To mitigate this risk, we implement token-based authentication methods such as JWT. Tokens have an expiration time, reducing the risk of token reuse.

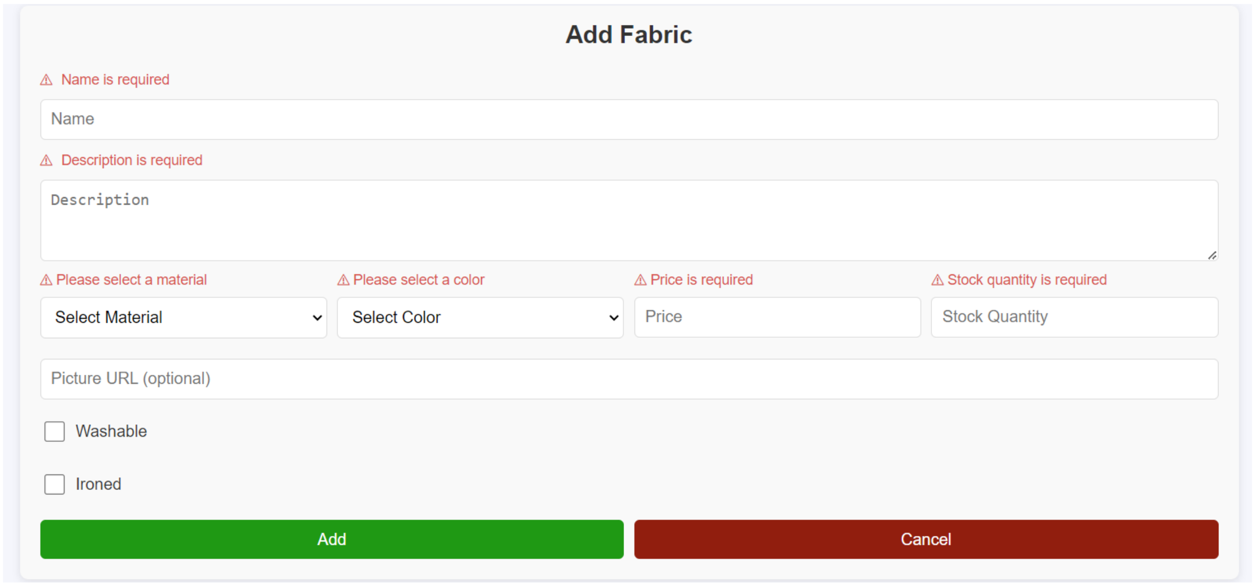
Function Level Access Control

Function-level access control ensures that users can only access specific application functionalities based on their roles and permissions. By restricting access to sensitive endpoints to authorized roles only, we reduce the risk of unauthorized users performing actions outside their scope. This is particularly important for endpoints that modify or delete data.

```
<Routes>
  {/* unauth */}
  <Route path="/" element={<HomePage />} />
  <Route path="/authentication" element={<AuthenticationPage />} />
  <Route path="/catalogue" element={<CataloguePage />} />
  {/* admin */}
  <Route path="/fabric-management" element={<ProtectedRoute element={<FabricManagementPage />}
requiredRole="ADMIN"/>} />
   <Route path="/user-management" element={<ProtectedRoute element={<UserManagementPage />}
requiredRole="ADMIN"/>} />
   <Route path="/statistics" element={<ProtectedRoute element={<StatisticsPage />} requiredRole="ADMIN"/>} />
   <Route path="/orders" element={<ProtectedRoute element={<OrderPage />} requiredRole="ADMIN"/>} />
   {/* client */}
   <Route path="/cart" element={<ProtectedRoute element={<CartPage />} requiredRole="CLIENT"/>} />
   <Route path="/my-orders" element={<ProtectedRoute element={<UserOrdersPage />} requiredRole="CLIENT"/>} />
   {/* auth */}
   <Route path="/my-account" element={<ProtectedRoute element={<PersonalInfoPage />} requiredRole="CLIENT,ADMIN"
/>} />
</Routes>
```

Software and Data Integrity Failures and Insecure Design

This is prevented by using validation on both front (react hook forms) and backend to ensure that no other validation is going through.



```
@NotNull
@DecimalMin("0.0")
@Digits(integer = 10, fraction = 2)
private double price;
```

```
@NotNull
@PositiveOrZero
private int stock;

@Pattern(regexp = "^(https?://.*\\.(png|jpg|jpeg|svg|gif))?$")
private String pictureUrl;
```

Using Components with Known Vulnerabilities

Using outdated or vulnerable libraries and components increases the risk of security breaches. Regularly updating libraries and staying informed about known vulnerabilities are essential practices to mitigate this risk. Knowing this, the issue is acknowledged, but no solutions are implemented for this project.

Insufficient Logging and Monitoring

Without sufficient logging and monitoring, detecting and responding to security incidents becomes difficult. We implement centralized logging to track security-related events and behaviours. By monitoring unusual activity, such as failed login and register attempts, we can detect and respond to potential attacks in real-time.

```java
public LoginResponse login(LoginRequest loginRequest) {
    UserEntity user = userRepository.findByUsername(loginRequest.getUsername())
            .orElseThrow(AuthException::invalidCredentials);

    if (!isPasswordValid(loginRequest.getPassword(), user.getPassword())) {
        logger.warn("Failed login attempt for username: {}", loginRequest.getUsername());
        throw AuthException.invalidCredentials();
    }

    String accessToken = generateAccessToken(user);
    logger.info("User {} logged in successfully.", loginRequest.getUsername());

    return LoginResponse.builder()
            .accessToken(accessToken)
            .build();
}
```

Server-Side Request Forgery (SSRF)

SSRF attacks occur when an attacker sends crafted requests from the server to internal services, potentially accessing unauthorised resources. To prevent SSRF, you must restrict external HTTP requests and validate input URLs before making any requests. This ensures that the application cannot be used to target internal systems or perform unwanted actions. Knowing this, the issue is acknowledged, but no solutions are implemented for this project.

Conclusion

Based on this analysis, the **TrendTrack** platform has identified and mitigated several critical security risks through secure development practices, such as **backend validation**, **encryption** and **secure access control**. However, certain risks, such as **Injection** and **Broken Authentication**, require continued attention and regular updates to ensure ongoing protection. By addressing these vulnerabilities proactively, the platform aims to provide a secure environment for its users and safeguard sensitive data.