Security Report | AWASP

Executive Summary

This report provides an overview of the security measures implemented in [Your Application Name], a full-stack application designed to serve law firms in managing clients, cases, and attorneys. The report covers various aspects of security, focusing on vulnerabilities identified by OWASP and corresponding mitigation strategies.

Application Overview

LawLink is built using Java Spring Boot for the backend, utilizing JWT tokens for authentication and JPA for database interaction. The front end is developed with React and TypeScript.

OWASP Top 10 Vulnerabilities

1. Injection

- **Mitigation**: The use of JPA (Java Persistence API) with Hibernate ORM (Object-Relational Mapping) provides protection against SQL injection attacks by automatically escaping and parameterizing input when executing queries against the database. This prevents malicious input from being interpreted as SQL commands.

  Additionally, by leveraging JPQL (Java Persistence Query Language) or Criteria API for querying databases instead of native SQL queries, the risk of SQL injection is further mitigated, as JPQL queries are type-safe and do not involve direct string concatenation.

- **Implementation Details**:
  - Entity classes are annotated with JPA annotations such as `@Entity`, `@Table`, `@Column`, etc., to define the object-relational mapping.
  - Criteria API or JPQL queries are used for database operations instead of native SQL queries wherever possible.
  - Input validation is performed at the application layer to ensure that user-supplied data conforms to expected formats and ranges before being passed to JPA queries.

```java
@Entity
@Table(name = "client")
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class ClientEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "firstName")
    private String firstName;

    @Column(name = "lastName")
    private String lastName;
```

2. Broken Authentication

- **Mitigation**:
    - **Password Hashing**: User passwords undergo secure hashing using industry-standard algorithms like bcrypt or SHA-256, rendering them resistant to brute-force attacks and unauthorized decryption.

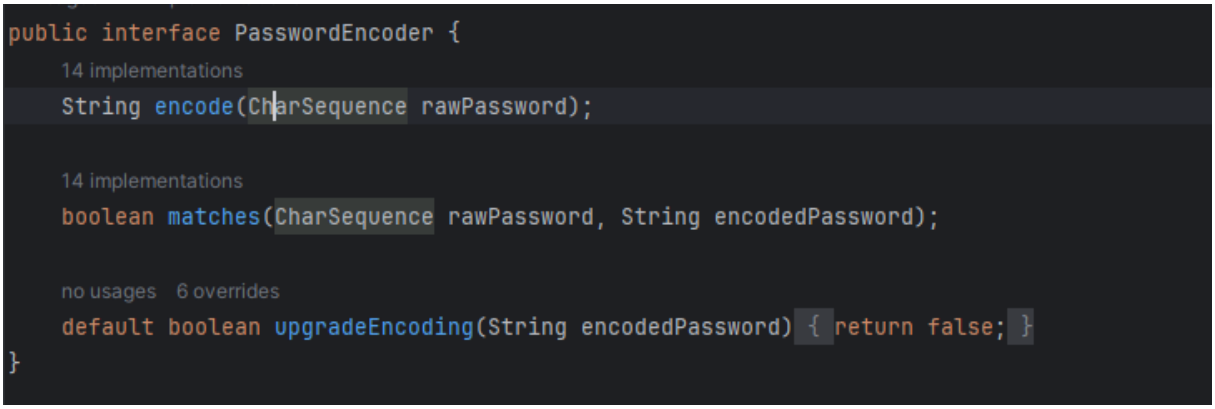
```java
if (!matchesPassword(loginRequest.getPassword(), user.getPassword())) {
    throw new InvalidUserException("Invalid username or password");
}

String accessToken = generateAccessToken(user);
RefreshTokenEntity refreshToken = generateRefreshToken(user);

user.setRefreshTokenId(refreshToken);
userRepository.save(user);

return LoginResponse.builder()
        .accessToken(accessToken)
        .refreshToken(refreshToken.getToken())
```

    - **Token-based Authentication**: Authentication tokens, such as JSON Web Tokens (JWT), are utilized to validate user identity securely. Tokens are encrypted and signed to deter tampering and forgery attempts.



3. Sensitive Data Exposure

- **Mitigation**: All passwords are being encrypted by the SHA-256 Algorithm. To sensitive data such as names, phone numbers, emails, and others, the access is being controlled by something called “Broken Access Control”. Only the person whose account is, and an authorized employee can access such data.

5. Broken Access Control

- **Mitigation**: Role-based access control has been implemented so that only authorized faces can access sensitive functionalities and data.

6. Security Misconfiguration

- **Mitigation**: Follow secure configuration guidelines for all components of the application, including web servers, databases, and frameworks. Regularly scan for and remediate misconfigurations using automated tools.

7. Cross-Site Scripting (XSS)

- **Mitigation**: The application implements input validation on both the client and server, sides. On the client side it is done by simple HTML validation and on the server side it is done by inbuild annotations which do not allow any data different from what is wanted to be processed.

```
<input
    type="text"
    name="lastName"
    className="input-text"
    placeholder="Last Name"
    value={formData.lastName}
    onChange={handleChange}
/>
<input
    type="email"
    name="email"
    className="input-text"
    placeholder="Email"
    value={formData.email}
    onChange={handleChange}
/>
```

9. Using Components with Known Vulnerabilities

- **Mitigation**: Regular updates and patches all third-party libraries and components used in the application. Monitoring security advisories and subscriptions to vulnerable databases to stay informed about known vulnerabilities.

10. Insufficient Logging & Monitoring

- **Mitigation**: In **LawLink**, logging mechanisms are implemented to track security-relevant events and facilitate incident response. However, there is room for improvement in enhancing the comprehensiveness and effectiveness of logging and monitoring capabilities.
- **Implementation Details**:
    - **Logging**: The application logs security-relevant events, such as authentication failures, access control violations, and critical application errors, to dedicated log files. Log entries include timestamps, user identifiers, and contextual information to aid in analysis and troubleshooting.

Conclusion

LawLink Client Software is equipped with measures to address the OWASP Top 10 vulnerabilities, mitigating risks and ensuring the security of sensitive data for law firms. However, security is an ongoing process, and continuous vigilance is necessary to adapt to emerging threats and maintain a strong security posture.

Specific attention should be given to **XML External Entities** & **Insecure Deserialization.** Since none of those issues is being addressed in the application this needs to be implemented as soon as possible.