Documentation: Web Security Fundamental Lab

This document provides a comprehensive overview of the security measures implemented in a Spring Security-based application. It includes an analysis of vulnerabilities found through an OWASP Dependency-Check Report, potential code vulnerabilities, threat modeling based on OWASP guidelines, and a discussion on web security fundamentals.

Security Measures Implemented

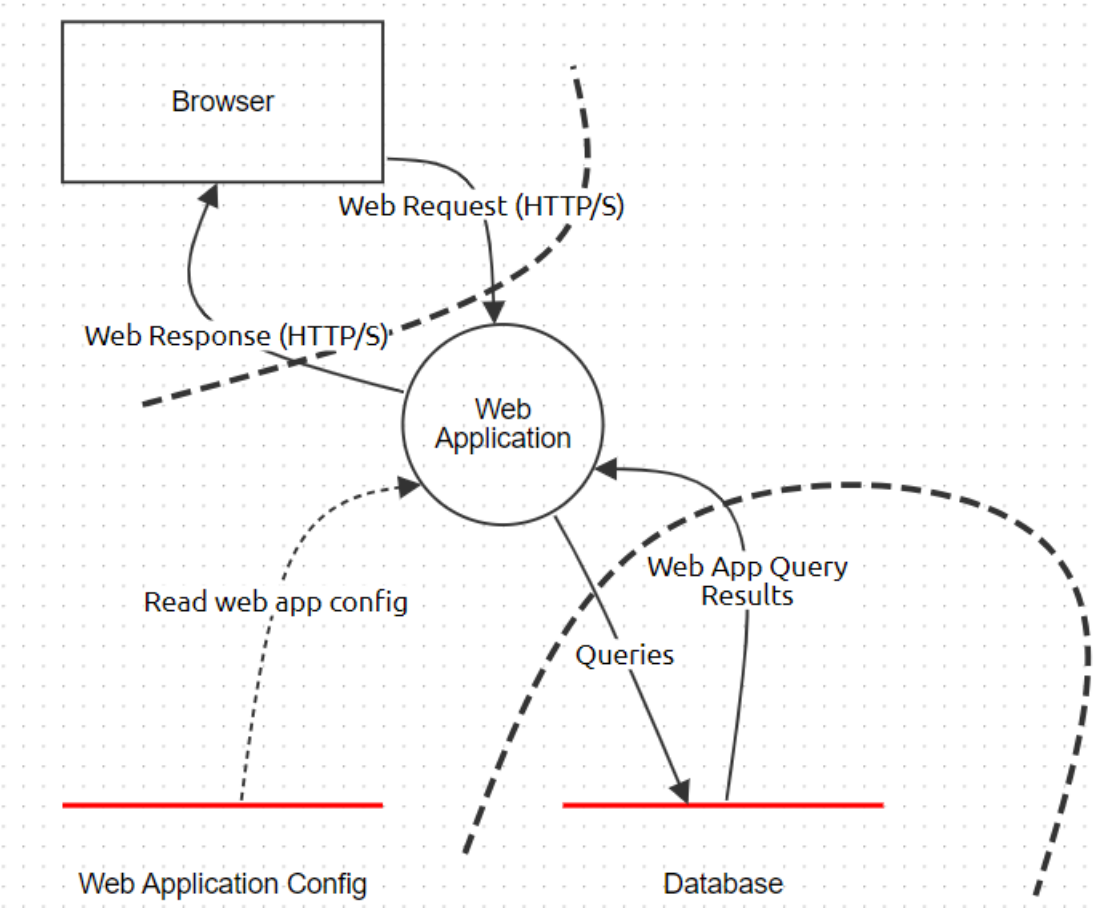
1. The application uses Basic Authentication with Spring Security where all endpoints are secured. Users must be authenticated to access any endpoint of the API.
2. Input Validation: We have implemented robust input validation mechanisms to ensure that only appropriately formatted data is allowed. This includes:
    - Content Validation: Before creating or updating notes, the content is validated against a set of rules to reject potentially harmful scripts, such as <script> tags, and other unwanted patterns using regular expressions. This helps prevent Cross-Site Scripting (XSS) attacks.
3. Output Encoding: To protect against XSS vulnerabilities, all output data, especially user-generated content, is encoded before being sent to the client. This ensures that any potentially dangerous characters are rendered harmless.
    - HTML Encoding: Using StringEscapeUtils.escapeHtml4(), we encode all HTML content, effectively preventing any HTML or JavaScript from executing in the user's browser.

Vulnerability Analysis Using OWASP Dependency-Check Report

1. Disabling CSRF protection exposes the application to CSRF attacks.
2. Spring Framework – Regular expression Denial of Service (ReDoS): Insufficient Regular Expression Complexity

Threat Modeling

The threat model follows the OWASP STRIDE methodology, identifying potential threats based on the application data flow.



Web Security Fundamentals

Web security is crucial in protecting both users and infrastructure from various threats. Here are my takeaways from the fundamental concepts:

a. Cross-Site Scripting (XSS)

XSS attacks allow attackers to inject malicious scripts into content that other users see. Protection strategies include validating input, encoding output, and using Content Security Policies (CSP).

b. SQL Injection

This attack exploits vulnerabilities in data-driven applications to execute malicious SQL statements. Defences include using parameterized queries, ORM frameworks, and rigorous validation.

c. Cross-Site Request Forgery (CSRF)

In CSRF attacks, unauthorized commands are transmitted from a user that the web application trusts. Mitigations include using anti-CSRF tokens and same-site cookies.

d. Security Misconfiguration

This broad category includes unpatched flaws, default configurations, unnecessary services, and open cloud storage. Regular security audits and following best practices can mitigate these risks.

e. Session Management

Secure session management is vital to prevent unauthorized access. Techniques include secure cookie attributes, timeout policies, and proper session invalidation.

f. Encryption

Using HTTPS, employing strong algorithms, and proper key management practices are essential to ensuring data confidentiality and integrity.

These fundamental concepts form the backbone of web security strategies, aiming to protect data integrity, prevent unauthorized access, and ensure the confidentiality of user data. Implementing these along with regular security assessments like dependency checks ensures a robust security posture for web applications.