Individual Project
Security Report

Date: 04.06.2024

Version: Version 1.0

Version History

Version	Date	Author	Changes	State
1.0	04.06.2024	Claudiu Badea	Initial Document	Complete

Contents

Vulnerability	Likelihood	Impact	Risk
Injection	Moderate	High	High
Broken Authentication	Low	High	Moderate
Sensitive Data Exposure	Low	High	Moderate
XML External Entities	Low	Moderate	Low
Broken Access Control	Moderate	High	High
Security Misconfiguration	Moderate	High	High
Cross-Site Scripting (XXS)	Moderate	High	High
Insecure Serialization	Low	High	Moderate
Using Components with Known Vulnerabilities	Moderate	High	High
Insufficient Logging and Monitoring	Low	High	High

1 Introduction

The security report for **QWEST** adheres to the 10 OWASP principles to ensure the application is secure and resilient against a variety of security threats. These principles provide best practice guidelines for application security, aiding in the identification and mitigation of potential vulnerabilities.

2 Injection (A1)

SQL injection can occur when database queries are improperly constructed and lack proper input validation. For instance, attackers can exploit input fields to execute arbitrary SQL commands if user inputs are directly concatenated into SQL queries without adequate sanitization.

Compared to raw SQL queries, using a JPA (Java Persistence API) repository significantly reduces the risk of SQL injection. JPA repositories leverage object-relational mapping (ORM) and predefined query methods to keep query structures separate from user inputs, effectively preventing SQL injection attacks.

Predefined Query Methods: (**implemented**) In JPA repositories, query methods can be specified using method names. The underlying JPA implementation then automatically generates the corresponding SQL queries. This approach minimizes the risk of injection vulnerabilities because the framework safely converts method names into secure SQL queries.

Named Queries: (**implemented**) Named queries are used in the program, facilitated by XML configurations or annotations. This feature allows for a centralized declaration of queries, ensuring that they are predefined and thoroughly validated, thereby reducing the risk of injection.

Query Parameters: (**partially implemented**) The program employs named queries, which are enabled through XML configurations or annotations. Named queries provide a centralized way to declare queries, ensuring they are predefined and thoroughly validated, thus reducing the risk of injection. However, further implementation of query parameters is needed to fully leverage this functionality and enhance security.

3 Broken Authentication (A2)

Vulnerabilities may arise from hard-coded credentials, ineffective session management, or lax password policies. For example, user accounts can be compromised if passwords are stored in clear text or if inadequate encryption techniques are used.

Effective use of access tokens (JWT) and securely salted passwords significantly reduces the risk of authentication failure. A comprehensive security posture requires considering the broader context of authentication and session management.

Password Hashing: (**implemented**) Robust, adaptive hashing algorithms like BCrypt are used for password hashing to prevent brute-force and rainbow table attacks.

JWT Implementation: (**implemented**) JSON Web Tokens (JWT) are implemented following industry best practices, including secure signatures, specified expiration dates, and careful server-side verification to prevent tampering.

Session Management: (**implemented**) Secure session management practices are in place, such as session regeneration after login and session timeouts. However, secure cookie properties like HttpOnly and Secure are not yet used.

Secure Token Storage: (**implemented**) Access tokens are stored securely in session storage. Care is taken to minimize the risk of cross-site scripting (XSS) attacks compromising token integrity.

Secure Communication: (**not implemented**) Ensuring that all client-server communication, especially during authentication, occurs over secure channels (HTTPS) is crucial to reduce the likelihood of man-in-the-middle attacks.

Password Reset Mechanism: (**implemented**) A secure password reset procedure is in place, employing multi-factor authentication where possible to help users safely recover access to their accounts.

4 Sensitive Data Exposure (A3)

Data Exposure may occur from storing sensitive data (e.g., credit card numbers) without proper encryption or from exposing sensitive data through unreliable APIs. Unauthorized parties could access this data due to inadequate protection measures.

The main cause of the Sensitive Data Exposure risk is improper handling or inadequate security of private or personally identifiable information (PII). The risks associated with sensitive data exposure are naturally reduced if the application does not handle sensitive data.

Security by Design: (**partly implemented by default**) Incorporating security measures into the design and development process of your application, even if it doesn't handle sensitive data, is a good practice. This approach establishes a solid security foundation for any features or updates that may be added in the future.

5 XML External Entities (XXE) (A4)

An attacker may exploit improperly validated XML input that a program parses to include external entities, exposing confidential information or causing a denial-of-service attack. This can happen if the application uses unreliable XML processing configurations or libraries.

Lower Risk of XXE Attacks: Since the application does not use XML and does not process XML input, the risk of XXE (XML External Entity) attacks is significantly reduced. The specific danger related to XXE is not relevant if the program does not process XML.

6 Broken Access Control (A5)

Unauthorized users may be able to access restricted capabilities due to inconsistent access control checks. For instance, if access controls are not correctly implemented on the server side, users may gain access to privileged actions without proper authorization.

Role-Based Access Control (RBAC): (**implemented**) Clear roles and corresponding permissions have been defined through the implementation of role-based access control. The program strictly enforces these restrictions at every layer to ensure safe and reliable access management.

Least Privilege Principle: (**implemented**) The least privilege concept has been implemented to ensure that roles and users have access to only the minimal amount of information required to perform their assigned duties. Overly permissive roles that could provide unnecessary access have been avoided.

7 Security Misconfigurations (A6)

Security flaws could be revealed by improperly configured server settings, permissions, or unnecessary services. This could happen if default configurations for security settings are left in place or due to improper configurations.

8 Cross-Site Scripting (XSS) (A7)

XSS Vulnerabilities may result from a lack of output encoding and input validation. If user inputs aren't sanitized before being displayed, attackers could insert malicious scripts that execute in the browsers of other users.

9 Insecure Deserialization (A8)

An attacker may alter serialized data to execute arbitrary code if an application accepts serialized objects from untrusted sources without performing the necessary validation. This vulnerability can occur if the program does not secure and validate deserialization operations properly.

10 Using Components with Known Vulnerabilities (A9)

The application may be vulnerable to known security issues if it uses out-of-date libraries, frameworks, or dependencies without receiving regular updates. Keeping these components up-to-date is crucial to mitigate the risk of exploiting known vulnerabilities.

11 Insufficient Logging and Monitoring (A10)

Delays or inefficiencies in incident response might result from inadequate logging and monitoring procedures. Security incidents could go unnoticed if the program doesn't implement sufficient logging techniques or fails to monitor logs for suspicious activity.

12 Conclusion

Strong password hashing, JWT-based access control, and role-based access management are just a few of the security measures implemented to protect the application. These steps, in line with the OWASP Top Ten best practices, effectively reduce potential hazards.

Although the current security posture is commendable, constant vigilance is essential. The program's robustness is enhanced by frequent audits, secure coding techniques, and ongoing monitoring. Maintaining this proactive strategy ensures that the application remains secure even in the face of evolving threats.

Finally, the security measures in place provide a strong foundation. However, maintaining long-term integrity and resistance to threats requires a commitment to continuous security awareness and improvement.