**OWASP Report**

***<<Warehouse project>>***

*<<Warehouse & CO>>*

|  |
| --- |
| **Date : 17.01.2021** |
| **Version : 0.3** |
| **State : For review** |
| **Author : Bojidar Balabanov** |

#### Version history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Changes** | **State** |
| 0.1 | 15.12.2020 | B.Balabanov | First draft | First draft |
| 0.2 | 13.01.2020 | B.Balabanov | Added overview | For review |
| 0.3 | 17.01.2020 | B.Balabanov | Better injection explanation and added cross-site scripting explanation | For review |

Contents

[1. OWASP top 10 security risks 3](#_Toc61805382)

[2. State of the application 5](#_Toc61805383)

[3. Overview 8](#_Toc61805384)

# OWASP top 10 security risks

1. [**Injection**](https://owasp.org/www-project-top-ten/2017/A1_2017-Injection). Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
2. [**Broken Authentication**](https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication). Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities temporarily or permanently.
3. [**Sensitive Data Exposure**](https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure). Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
4. [**XML External Entities (XXE)**](https://owasp.org/www-project-top-ten/2017/A4_2017-XML_External_Entities_(XXE)). Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
5. [**Broken Access Control**](https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control). Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users’ accounts, view sensitive files, modify other users’ data, change access rights, etc.
6. [**Security Misconfiguration**](https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration). Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.
7. [**Cross-Site Scripting (XSS)**](https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS)). XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
8. [**Insecure Deserialization**](https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization). Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.
9. [**Using Components with Known Vulnerabilities**](https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities). Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.
10. [**Insufficient Logging & Monitoring**](https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring). Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

# State of the application

##### Injection

SQL Injection-The application has no injection flaws due to the fact that Spring Boot and more specifically the JPA repository library has innate prevention within the library itself that targets this issue. It provides developers with methods which generate parameterized queries for the SELECT, UPDATE, INSERT, DELETE SQL functions based on the class which is specified when initializing the repository. The application has no usages of complex queries so the only used ones are the basics provided by JPA. Another thing user input is being sanitized by angular and then sent to the backed, also angular sanitizes all data that comes from the back-end before visualizing it.( <https://www.baeldung.com/sql-injection>)

NoSQL Injection- Not applicable because this type of injection is concernes databases which do not use SQL queries.( [https://www.netsparker.com/blog/web-security/what-is-nosql-injection/)](https://www.netsparker.com/blog/web-security/what-is-nosql-injection/)

OS command Injection- Not applicable because application does not pass any data to the system shell. (<https://owasp.org/www-community/attacks/Command_Injection>)

ORM Injection- As specified under SQL Injection the used ORM is JPA and the only functions that are used are the safe ones provided by the ORM itself.

LDAP Injection- Not applicable because LDAP is not used by the application

##### Broken Authentication

The application has a flaw connected to this issue, because the implemented type of authentication is Basic and since its encoding type is base64 this would mean that the application is not really secure. Normally if this was an official project it would require that security is given a lot more attention, but since it is a university solo project it fulfills the minimum requirement for security.

##### Sensitive Data Exposure

As mentioned above since the application supports weak encryption (base64) the information being shared between API and client can be described as unprotected. Although passwords are only being sent form client to api and the comparison needed for logging in is being done there where it is secure.

##### XML External Entities

Not applicable. XML is not being used.

##### Broken Access Control

The client always asks the API if the user who is logged in should access the requested resource and prevents unauthorized users from accessing restricted content.

##### Security Misconfiguration

The application does not share sensitive data such as stack traces when an error occurs, instead it returns an error 500 Bad Request.

Due to the usage of “Basic Auhtentication” the HTTP “Authorization” header may be considered as unprotected.

Unnecessary/unused features and components are removed.

##### Cross-Site Scripting (XSS)

Angular provides built-in input sanitization and treats all inputs as untrusted by default, which mitigates most of the risks around cross-site scripting.

Angular sanitizes untrusted values for HTML, styles, and URLs; sanitizing resource URLs isn't possible because they contain arbitrary code. In development mode, Angular prints a console warning when it has to change a value during sanitization.

For further security measures it will be a good decision to enable CSP(Content security policy) and set up appropriate Content-Security-Policy HTTP header in the communication between back-end and front-end.

##### Insecure Deserialization

Not applicable. Serialization is not being used.

##### Using Components with Known Vulnerabilities

Unused dependencies, unnecessary features, components, files are removed.

All dependencies, features, components, files, and documentation are kept up to date both in client and server sides.

##### Insufficient Logging & Monitoring

Currently there is only logging and monitoring included for restocking products and preparation tasks.

The application may be considered vulnerable due to authentication, product addition being unsupervised

# Overview

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Status | Reason | Suggestion |
| Injection | Ok |  |  |
| Broken Authentication | Insecure | Usage of basic authentication. | Upgrade to a non-vulnerable security method like JWT with OAuth 2. |
| Sensitive Data Exposure | At risk | Due to authentication being broken | Follow broken authentication suggestion. |
| XML External Entities | Not applicable. | XML is not being used. |  |
| Broken Access Control | OK |  |  |
| Security Misconfiguration | OK |  |  |
| Cross-Site Scripting(XSS) | Could be better | Although angular’s built-in sanitization covers most of the vulnerabilities around XSS, there is room for improvements. | Enable and configure CSP headers. |
| Insecure Deserialization | Not applicable. | Serialization is not being used. |  |
| Using Components With Known Vulnerabilities | OK |  |  |
| Insufficient Logging & Monitoring | Could be better | Restocking and preparation task operations are the only things being logged and monitored. | Add authentication, product register logging |