OWASP TOP 10 GAP ANALISIS

A1 – Injection

Injection flaws, such as SQL, 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.

To prevent SQL injection we are using the latest versions of Hibernate and Spring data JPA which both provide parameterized interface which are used to prevent SQL injection. Further data security is implemented through front-end and back-end data validation using regular expression patterns.

[HQL secure example](https://software-security.sans.org/developer-how-to/fix-sql-injection-in-java-hibernate), [Spring data jpa](https://docs.spring.io/spring-data/jpa/docs/current/reference/html/), [Hibernate data jpa](http://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html)

A2 – Broken Authentication and Session Management

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities.

All user credentials are cyphered with bcrypt strong hashing function. Credentials cannot be guessed nor brute forced. User credentials cannot be overwritten through weak account management functions because they are protected by dynamic roles and protocols. Session id’s are not exposed though URL. The user has a user timestamp which guarantees the session validity during that time. During logout SSL authentication tokens are properly invalidated. All communication is enabled though HTTPS can be implemented with Spring secutiry framework which guarantees password security. For better user security one of the best practices is to use (Google,Facebook,Twitter…) authentication and authorization services (For example OAuth 2.0). Also to improve secutiry the user should have a limited amount of attempts on the log in page.

[Outh 2.0](https://oauth.net/2/) , [Bcrypt](https://en.wikipedia.org/wiki/Bcrypt) , [Spring secutiry](https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/)

A3 – Cross-Site Scripting (XSS)

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. 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.

In order to prevent XSS we should look into the XSS Prevention cheat sheet. These rules should provide good and solid XSS prevention on the front-end of our application. Also it displays a guide on the best practice front-end development.

[XSS prevention cheat sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)

A4 – Insecure Direct Object References

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

For all vulnerable resources which are available for the user must refer to the current logged user. The application should not expose any data which do not refer to the logged user. This protects the integrity of all database resources.

Testers can easily manipulate parameter values to detect such flaws. Code analysis quickly shows whether authorization is properly verified. Each use of a direct object reference from an untrusted source must include an access control check to ensure the user is authorized for the requested object.

[CWE Entry 639 on Insecure Direct Objects](http://cwe.mitre.org/data/definitions/639.html) ,  [CWE Entry 22 on Path Traversal](http://cwe.mitre.org/data/definitions/22.html)

A5 – Security Misconfiguration

Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.

The application OS, Web server, DBMS and all libraries which are used are up to date with the latest standards. The admin user console should be disabled. The server exception stack trace should not be available to the public eye during uptime. Test data should not be available during runtime especially user sensitive data. Always change default passwords and eliminate unnecessary accounts (such as guest).

[Web Server Hardening](http://www.pcmag.com/article2/0,2817,11525,00.asp)

A6 – Sensitive Data Exposure

Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

Any sensitive data cannot be saved as clear text format in the database. The application cryptography algorithm is not outdated and is trustworthy. User sensitive data are hashed before being added in the database. To achieve this we used the bcrypt cryptographic algorithm. By using HTTPS protocol we achieve safe data exchange between the web browser and the servers restful services. SOAP web services are cyphered with certificate public keys from the other communication side(Spring WS). During the soap data receiving the data is deciphered with the users private key from the certificate. For best practice also disable caching pages which contain sensitive data.

[Spring WS](http://docs.spring.io/spring-ws/docs/current/reference/htmlsingle/)

A7 – Missing Function Level Access Control

Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

The application has an interceptor handler which handles endpoint security through user role base access control. This service provides access control on each endpoint of the application. If the user is authorized and has the permission then he is permitted for that action. Best practice is to prevent user from accessing functionalities which they do have the permission for.

[Spring interceptor handler](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/handler/HandlerInterceptorAdapter.html)

A8 - Cross-Site Request Forgery (CSRF)

A CSRF attack forces a logged-on victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim’s browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

CSRF is manifested through malicious web pages with the goal of stealing sensitive user data. To prevent CSRF request it is advised to use CSRF Guard which can automatically include CSRF tokens in most object oriented languages(Java EE, .NET, PHP …). These unique tokens have the role of validating a users HTTP request and give them trustworthy value. One cannot prevent the sending of a malicious HTTP request, but it can neutralize it. CSRF can be generated on multiple ways (fake web sites, JSON objects, false input forms…)

[Spring CSRF](https://docs.spring.io/spring-security/site/docs/current/reference/html/csrf.html)

A9 - Using Components with Known Vulnerabilities

Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

The most important thing to do before implementing any software is to validate all libraries being used for the creation of the project. The NVD vulnerability database gives you the information of any library planned for the project. This product allows us to give a better analysis on the given libraries and it provides us with the necessary information with which we can give judgement on the specified libraries and choose the correct ones in our implementation. Most project components do not create vulnerability patches for older versions. Instead, most simply fix the problem in the next version. To prevent system vulnerability it is advised not to use highly vulnerable libraries. If the usage of a certain library cannot be prevented, then that library is wrapped up in a security wrapper.

[NVD vulnerability database](https://nvd.nist.gov/vuln/search)

A10 – Unvalidated Redirects and Forwards

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

It is advised when using redirect and forwards to prevent any parameters to ensure safe communication. If the usage of these services cannot be prevented, it must be ensured that the supplied value is valid, and authorized for the user. It is recommended that any such destination parameters be a mapping value, rather than the actual URL or portion of the URL, and that server side code translate this mapping to the target URL.

[URI best practice guides](https://webmasters.googleblog.com/2009/01/open-redirect-urls-is-your-site-being.html)