OWASP Security Report

Balan Mihai-Stefan

Application: ChessNow

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Likelihood | Impact | Risk | Actions possible | Planned |
| A1: Broker access control | High | Severe | High | N/A | Yes |
| A2: Cryptoghraphic failure | Unlikely | Severe | Low | Data Encryption | Yes |
| A3: Injection | Very Unlikely | High | Low | Implementing JPA Repository, parameterized custom queries | Yes |
| A4: Insecure Design | Moderate | Severe | Moderate | More integration testing, frequent code reviews | Yes |
| A5: Security Misconfiguration | Moderate | High | High | Proper CORS filtering | No |
| A6: Vulnerable and Outdated Components | Unlikely | Low | High | Updating installed modules | No |
| A7: Identification and Authentication Failures | Likely | Severe | High | Adding prevention for Brute-force attacks | No |
| A8: Software and Data Integrity Failure | Moderate | High | Moderate | Practicing caution when importing and using libraries. Using SonarQube vulnerability report | Yes |
| A9: Security Logging and Monitoring | High | Low | High | Adding loggers to every operation | No |
| A10: SSRF | Moderate | Severe | High | Validating all URL’s of requests | Yes |

Context explanations

* A1: Actions are authorized based on roles. The roles are received based on the user’s JWT authentication token, which is sent in request headers.
* A2: Passwords are encrypted using SHA512 hashing, salting and peppering. In addition, no user sensitive information is sent from the server after being added, unless the user wishes to change it. The changing operation cannot be accessed by an unauthorized party.
* A3: For storing data the application uses an ORM which makes it impossible to inject malicious SQL, due to the way of storing
* A4: Unsecured endpoints or open ports may be left behind and go unnoticed, until proper integration and penetration testing. Also since cross server communication is done via HTTP instead of HTTPS, this leaves the app vulnerable to Man-In-The-Middle attacks.
* A5: CORS configurations check for specific allowed origins and not allow any requests made from any server other than the designated frontend server. The user should not be able to trigger error messages that would reveal any relevant information on how the application works. Also, there will not be any debugging outputs and that could be used in harmful ways. Since the application is not ready for production, this does not take priority.
* A6: Components are mainly updated with security updates, if they function properly and have been designed future-proof (as is the case most of the time). A vulnerability could arise if a component update is no longer compatible with the deployed version of the app. Updating the entire application (for example changing the JDK version) is a very tricky situation, which would result in using the outdated and vulnerable module for a certain period of time.
* A7: As the system may allow weak passwords right now, the responsibility is placed on the user to choose a strong password and secure his account. Communication between unauthorized parties and the server should be limited, to prevent password cracking via the use of Rainbow Tables or any type of brute-force attack.
* A8: The project is not using untrusted libraries, the only external libraries are open-source on github and maintained. If the project were to ever use an untrusted library, it would be picked up by the SonarQube vulnerability report (this is a premium feature, discussion is hypothetical). Npm also reports vulnerabilities when installing packages. If a vulnerable library is the only way to implement a feature, the feature will take priority, because of the nature of the project.
* A9: An error log should be present on the user side, keeping records of operations and their outputs. At this point in development this is not a priority.
* A10: Every URL is validated by the cross origin configuration, that only allows specified sources and ports for requests (example: <http://localhost:5173>).