Assessment Report: Bart Voet's Learning Application

Table of Contents

[Introduction 2](#_Toc156150424)

[Security Assessment Tools Used 2](#_Toc156150425)

[Static Analysis Security Testing (SAST) 2](#_Toc156150426)

[Dynamic Analysis Security Testing (DAST) 2](#_Toc156150427)

[Software Composition Analysis (SCA) 2](#_Toc156150428)

[Application Overview 2](#_Toc156150429)

[Access Information 2](#_Toc156150430)

[Running the application 2](#_Toc156150431)

[Initial Security Considerations 4](#_Toc156150432)

[Authentication, Authorization & Access Control 4](#_Toc156150433)

[Communication Channels 4](#_Toc156150434)

[Data Security and Privacy 4](#_Toc156150435)

[Security Assessment 4](#_Toc156150436)

[Static Analysis (SAST) 4](#_Toc156150437)

[Semgrep 4](#_Toc156150438)

[Findings 5](#_Toc156150439)

[Security Recommendations 6](#_Toc156150440)

[Dynamic Analysis (DAST) 7](#_Toc156150441)

[Zed Attack Proxy (ZAP) 7](#_Toc156150442)

[Findings 7](#_Toc156150443)

[Security Recommendations 7](#_Toc156150444)

[Software Composition Analysis (SCA) 8](#_Toc156150445)

[Dependabot 8](#_Toc156150446)

[Findings 8](#_Toc156150447)

[Security recommendations 9](#_Toc156150448)

[Conclusion 9](#_Toc156150449)

# Introduction

This security assessment aims to evaluate the effectiveness of secure development practices in Bart Voet's Learning Application. The report provides an overview of the application, highlights key security considerations, and offers recommendations to address identified issues, contributing to the overall improvement of the application's security.

# Security Assessment Tools Used

Three security assessment tools are used, each specifically tailored to assess distinct security domains. This diverse toolset facilitates a comprehensive examination, ensuring an understanding of vulnerabilities and strengths within each targeted area.

## Static Analysis Security Testing (SAST)

* Semgrep (CLI) was used for static code analysis to identify potential security vulnerabilities in the source code.

## Dynamic Analysis Security Testing (DAST)

* ZAP (Zed Attack Proxy) was used for dynamic analysis, evaluating the application's runtime behavior for security issues.

## Software Composition Analysis (SCA)

* Dependabot was used to analyse and monitor the dependencies used in the application, ensuring the use of up-to-date and secure components.

# Application Overview

Bart Voet's Learning Application is designed to facilitate exam preparation and self-testing by implementing three core concepts: Question, QuestionSeries, and QuestionSession. The application is structured around User-representation, allowing users to create and manage question series and sessions. The core algorithm is decoupled from the operating environment, enabling its use in various contexts, such as Spring Boot applications or command-line interfaces.

## Access Information

* Application Code: <https://github.com/bartvoet/ehb-enterpriseapps-learningappApplication>
* Runtime: Docker (and Kubernetes)

## Running the application

Launching the application involves running the container through the provided Docker Compose file, granting access to the application at <http://localhost:8080/>.

Upon reaching this URL, users encounter an initial login/registration screen.

A close up of a sign

Description automatically generated

A screen shot of a login form

Description automatically generatedA screenshot of a login form

Description automatically generated

Once successfully logging in or completing the registration process, users are directed to a comprehensive dashboard. Within this dashboard, users gain the ability to create, update, or initiate a new session.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

The application persists all data in a MySQL database.

# Initial Security Considerations

Prior to conducting the security assessment analysis with the specified security tools, it is essential to initially address the following security considerations.

## Authentication, Authorization & Access Control

* The application implements secure user authentication through Spring Security, binding to JPA for user details retrieval.
* Passwords are securely stored using BCrypt, reducing the risk of password compromise.
* The use of DaoAuthenticationProvider and UserDetailsServiceImpl enhances security.
* Designed for personal use, the application uses a simple login process as its access control. Once logged in, users can only access information relevant to their individual profile.

## Communication Channels

* The application enforces HTTPS through a self-signed PKS12 file, ensuring secure communication.

For the purpose of this assignment, it's important to clarify that the application was solely deployed within a Docker container using Docker Compose. In this scenario, communication occurs over HTTP.

* Security configuration is profile-based, differentiating between development and production environments.

## Data Security and Privacy

* Sensible information, like passwords, is not stored in clear text, promoting good security practices. The application leverages BCrypt for password encoding, enhancing data security.
* User data is bound to JPA and follows industry best practices for secure data handling (Spring Application).
* The application stores the minimal essential information required for its operation. In doing so, it refrains from retaining any extraneous data beyond what is strictly necessary for the functioning of the application.

# Security Assessment

## Static Analysis (SAST)

### Semgrep

I used the Semgrep Command-Line Interface (CLI) to conduct Static Analysis Security Testing. Since I was operating on a Windows machine, the initial step involved installing the Linux Windows Subsystem for Linux (WSL). Following the installation, I proceeded to install the necessary tools and cloned the target repository.

To streamline the reporting process, I logged into my Semgrep Cloud platform, facilitating the transmission of all findings from my local machine to the platform. The final step involved executing the Semgrep command to scan the code thoroughly and automatically send the report to my Semgrep Cloud account. This systematic approach ensured a robust security assessment of the codebase.

A close-up of a white background

Description automatically generated

### Findings

The scan reveals the presence of 14 security vulnerabilities:

A screenshot of a computer

Description automatically generated

**1. django-no-csrf-token**

* **Severity**: **Medium**
* **Description**: Manually-created forms in Django templates lack a specified csrf\_token, leaving them vulnerable to CSRF attacks.
* **Files Affected**:
  1. create.html
  2. createserie.html
  3. questionseriedetails.html
  4. questionsession.html

**2. no-new-privileges**

* **Severity**: **Low**
* **Description**: Privilege escalation via setuid or setgid binaries is possible in the specified service.
* **Files Affected**: docker-compose.yml (lines 3, 18)

**3. writable-filesystem-service**

* **Severity**: **Low**
* **Description**: The specified service runs with a writable root filesystem, potentially allowing malicious applications to download and run additional payloads or modify container files.
* **Files Affected**: docker-compose.yml (lines 3, 18)

**4. run-as-non-root (optional as only relevant for k8s)**

* **Severity**: **Low**
* **Description**: Containers running applications as the root user in Kubernetes may be vulnerable to privilege escalation attacks.
* **Files Affected**:
  1. app-deployment.yaml (line 15),
  2. mysql-deployment.yaml (line 54)

**5. allow-privilege-escalation-no-securitycontext (optional as only relevant for k8s)**

* **Severity**: **Medium**
* **Description**: Certain container images may contain binaries allowing privilege escalation.
* **Files Affected**:
  1. app-deployment.yaml (line 17)
  2. mysql-deployment.yaml (line 56)

**6. unrestricted-request-mapping**

* **Severity**: **Low**
* **Description**: Detected a method annotated with RequestMapping that does not specify the HTTP method. CSRF protections are not enabled for GET, HEAD, TRACE, or OPTIONS, and by default all HTTP methods are allowed when the HTTP method is not explicitly specified. This means that a method that performs state changes could be vulnerable to CSRF attacks.
* **File Affected**: UserSeriesController.java (line 155)

### Security Recommendations

With the identified security findings, the following recommendations are suggested to enhance the overall security posture of the codebase:

**1. django-no-csrf-token:**

Ensure all Django templates with manually created forms include a csrf\_token to prevent CSRF attacks. CSRF tokens protect against unauthorized form submissions, enhancing overall application security.

**2. no-new-privileges:**

Add “no-new-privileges:true” in “security\_opt” for the specified service in “docker-compose.yml”. This prevents potential privilege escalation via setuid or setgid binaries, strengthening container security.

**3. writable-filesystem-service:**

Add “read\_only: true” to the specified service in “docker-compose.yml”. Restricting filesystem write access mitigates the risk of malicious applications modifying or executing payloads.

**4. run-as-non-root (optional as only relevant for k8s):**

Add a securityContext to set “runAsNonRoot” to true in “app-deployment.yaml” and “mysql-deployment.yaml”. Running containers as non-root users in Kubernetes limits potential damage from privilege escalation attacks.

**5. allow-privilege-escalation-no-securitycontext (optional as only relevant for k8s):**

Add a securityContext to set “allowPrivilegeEscalation” to false in “app-deployment.yaml” and “mysql-deployment.yaml”. Preventing privilege escalation in containerized applications enhances overall security and protects sensitive resources.

**6. unrestricted-request-mapping:**

Specify the HTTP method (e.g., RequestMethod.POST) for the method annotated with RequestMapping in “UserSeriesController.java”. Defining the HTTP method mitigates CSRF vulnerabilities, particularly for methods performing state changes.

## Dynamic Analysis (DAST)

### Zed Attack Proxy (ZAP)

Zap automated scan was used to conduct Dynamic Application Security Testing for the application.

#### Scan Configuration

The AJAX Spider functionality has been activated due to the application's use of HTTP requests to communicate with the server. It's important to note that the nature of the application is dynamic rather than static, necessitating the use of AJAX Spider for a more thorough and accurate security assessment.

A screenshot of a computer

Description automatically generated

Executing the attack process led to a series of actions aimed at probing the application for vulnerabilities. Within a few minutes, alerts/findings are being reported.

Only warnings (Low to High, excluding informative) pertaining to **http://localhost** will be addressed to maintain clarity and conciseness in the assessment and recommendations.

### Findings

A white background with black text

Description automatically generated

### Security Recommendations

**1. Content Security Policy (CSP) Header Not Set (Medium):**

“Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.”

**2. Hidden File Found (Medium):**

“Consider whether or not the component is actually required in production, if it isn't then disable it. If it is then ensure access to it requires appropriate authentication and authorization, or limit exposure to internal systems or specific source IPs, etc.”

**3. Vulnerable JS Library (Medium):**

Upgrade to the latest version of jquery.

**4. Cookie without SameSite Attribute (Low):**

“Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies.”

## Software Composition Analysis (SCA)

### Dependabot

Security Composition Analysis was conducted using Dependabot. To leverage Dependabot, we first need to enable it in the repository beforehand. In preparation for the assessment, a fork of the learning app repository was generated, followed by the activation of Dependabot. The fork can be accessed via the following URL: <https://github.com/STUYB/ehb-enterpriseapps-learningapp>.

### Findings

After activation, Dependabot identified and notified us about one vulnerability:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### Security recommendations

The vulnerability's description is clear from the provided screenshot. A vulnerability has been identified in the used version of Hibernate Validator.

For a thorough examination of the security recommendation, we can open up the pull request and commits:

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

**To address this issue, it is recommended to resolve it by updating the Hibernate Validator version to 6.0.20.**

# Conclusion

In conclusion, this assessment employed various tools to pinpoint security vulnerabilities in Bart Voet's learning application. By adhering to the recommendations outlined in each section of the security assessment (SAST, DAST, and SCA), Bart Voet can enhance the security of the application, ensuring not only its functionality but also its overall security posture.