Exam Synopsis – Securing CI/CD Pipeline with SNYK, CodeQL, and Trivy

PB Software Development

Software Security

Rasmus Guldborg

Fall 2025

Synopsis by Marco Gabel & Fei Gu

Table of Contents

1. Introduction

This is a company case based on the company Agramkow, who's develop software systems for industrial companies around the world. Their PLIS platform uses containerized software components to connect different devices, such as machines, productions lines and such.

This project is about the integration of automated security analyses into the CI/CD pipeline. The purpose of this being to display how implement early security checks in development in compliance with the Cyber Resilience Act (CRA), with the help of automated tools.

To demonstrate this, we use a Nest.js application to mimic the Agramkow system. The project will be kept clean and simple as to focus on the pipeline and security tools instead.

2. Problem Statement

This project investigates how a CI/CD pipeline can be secured using automated analysis tools such as SNYK (Software Composition Analysis), CodeQL (Static Application Security Testing) and Trivy (Container Scanning).

The goal is to identify vulnerabilities during the build process and assess how this security controls support the Cyber Resilience Act requirements for secure software development and supply-chain protection.

3. Implementation

3.1 Pipeline architecture

This projects CI/CD pipeline follows the shift-left security approach, this for the reason being that it runs security checks as early as possible. The automatic trigger for the pipeline is set to be when a developer push or pull requests.

The structure of the flow is as follows:

> **Commit → SNYK (SCA) → CodeQL (SAST) → Build → Trivy (Container Scan) → Security Gate → Deploy**

Elaboration of the flow:

1. Developer commits.
2. SNYK scans the dependencies (package.json) and compares them to lists of known vulnerabilities (CVEs) and then passes if none are found.
3. CodeQL performs a static analysis of the code (TypeScript), detecting any insecure coding patterns.
4. Application builds as a docker image.
5. Trivy scans the produced docker image for vulnerabilities in the packages and system.
6. Successfully deploys the request, if all steps are successful.

3.2 Tool integration

All tools used are integrated through the GitHub Actions, which is a realistic environment for the development team. Similar to Bitbucket Pipeline, which has the same functionality.

The NestJS application is structured to follow a layered architecture as follows:

1. **Core** – Domain Models and Interfaces.
2. **Infrastructure** – Repository implementation, in this case in-memory.
3. **Application** – Controllers and Services.

REST endpoints are being exposed that are documented with Swagger, which is similar to the Agramkow structure for the PLIS platform, that exposes APIs.

With each security tool running automatically, via GitHub Actions the developers can then analyse each tool, see its output and this way retrieve feedback without having to use any tool manually.

3.3 Security findings

The repository uses a dependency set and container base image to showcase the tooling output, the most common being as follows:

1. **SNYK** – Vulnerable or outdated packages with CVEs, and suggestions for upgrades.
2. **CodeQL** – Analysis alerts about questionable patterns, potential injections points to expose.
3. **Trivy** – Vulnerabilities in the docker image and misconfigurations therein.

When each step has been completed, the CI workflow then collect its results and displays it. Showing how potential vulnerabilities and can be triaged and tracked.

3.4 CRA compliance mapping

Several key principles of the Cyber Resilience Act are being supported by our CI/CD implementation.

The 3 key articles being as follows:

1. **Article 10 – Secure Development Process**

   CodeQL – Static analysis that ensures secure coding practices are enforced by default.

2. **Article 13 – Vulnerability Handling**

   SNYK/Trivy: – Automated scanning that detects vulnerabilities continuously and enabling early remediation.

3. **Article 14 – Supply Chain Security**

   SNYK – Dependency analysis and generation of SBOM, with improved traceability and transparency.

CRA Requirement	Tools	Implementation	Status
Secure Development Process	CodeQL	SAST integrated into CI/CD pipeline for secure coding practices.	Implemented
Vulnerability Management	SNYK, Trivy & CodeQL	Automated scanning on every commit for early detection.	Implemented
Supply Chain Security	SNYK	Dependency scanning and automated SBOM generation.	Implemented
Security by Default	All Tools	Shift-left security integrated into workflow.	Implemented
Vulnerability Disclosure	GitHub Security	Security reporting and alerting centralized.	Implemented
Timely Updates	Dependabot	Automated security patching and dependency updating.	Implemented

Table 1: CRA Requirements Implementation Overview

4. Analysis and Results

The result of this project is a fully functional CI/CD pipeline that shows integration of automated security analysis into a workflow which is built to be similar to that of Agramkow. The primary focus of the analysis is how the security tools we have chosen works and supports the Cyber Resilience Act requirements.

The pipeline runs all the security checks as intended automatically on push and pulls. This displays a great example of how security can be integrated into CI/CD without the developer actually having those manual steps, that would otherwise be required. Each of the tools used contributes to the security as a whole, differently.

SNYK is identifying vulnerabilities which is very important in an environment like Agramkow in our case, where their different services rely on shared libraries. CodeQL checks and detects any insecure coding patterns through its static analysis, preventing flaws from entering production. Trivy then scans the built docker image and reports back with the vulnerabilities in the base image and packages, even with the code within the application being secure.

The analysis also presents that the automated tools we use have limitations. That meaning that a developer still needs to manually review the results and its aspects, that includes out of scope incident handling and runtime security.

In conclusion, the results retrieved from this project confirms that automated CI/CD security tools can be an effective for secure software development. This reflects the very realistic scenario where tools support the developer but does NOT replace secure development practices.

5. Conclusion

The project demonstrates how to secure the CI/CD Pipeline using the chosen automated tools. By integrating Trivy, CodeQL and SNYK we can display any vulnerabilities that can be found, fast and consistently throughout development.

Without focusing on the application itself, the design of the pipeline is relevant for systems such as, and in this case Agramkow's platform. The pipeline shows how the vulnerabilities in code, container images, dependencies, and alike can be automatically detected and caught before reaching deployment, this in turn highlights the term Secure by Design from the Cyber Resilience Act.

5. Conclusion

The project demonstrates how to secure the CI/CD Pipeline using the chosen automated tools. By integrating Trivy, CodeQL and SNYK we can display any vulnerabilities that can be found, fast and consistently throughout development.

Without focusing on the application itself, the design of the pipeline is relevant for systems such as, and in this case Agramkow's platform. The pipeline shows how the vulnerabilities in code, container images, dependencies, and alike can be automatically detected and caught before reaching deployment, this in turn highlights the term Secure by Design from the Cyber Resilience Act.