Developing a Framework for Integrating Security Testing into the CI/CD Pipeline using Automation.

MSc Practicum Part 2
MSc Cybersecurity

Praveen Derenda Seetharam
Student ID: 23174501

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Praveen Derenda Seetharam		
Student ID:	23174501		
Programme:	MSc Cybersecurity	**Year:**	2024-2025
Module:	MSc Practicum Part 2		
Supervisor:	Vikas Sahni		
Submission Due Date:	12/12/2024		
Project Title:	Developing a Framework for Integrating Security Testing into the CI/CD Pipeline using Automation.		
Word Count:	7368	**Page Count**	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Praveen D S

Date: 11/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Developing a Framework for Integrating Security Testing into the CI/CD Pipeline using Automation.

Praveen Derenda Seetharam
23174501

Abstract

The increasing frequency of security breaches in software applications proves the importance of more frequent and comprehensive security testing, integrated into CI/CD deployment pipelines. This research presents a novel framework that incorporates security testing at all stages of software development life cycle. The unique mix of SAST, DAST, and dependency vulnerabilities scanning, the framework provides developers and security team with a diverse set of security testing tools. The framework is implemented using GitHub Actions to streamline automation and integrates tools such as Lint for code formatting, CodeQL for static analysis, CodeClimate for maintainability checks, Snyk for dependency vulnerability scanning, and OWASP ZAP for dynamic application security testing. The output of these processes is then containerized and pushed to Docker Hub, ensuring a seamless integration with modern CI/CD workflows. OWASP Juice Shop, a deliberately vulnerable web application, serves as the target application for demonstrating the effectiveness of this framework. By automating security testing and embedding it into CI/CD pipelines, this research provides a robust approach to mitigating risks earlier in the development process while maintaining the agility and speed of modern software delivery practices.

1 Introduction

CI/CD is now recognized as one of the most important and effective processes for enabling rapid development and deployment of modern software. These pipelines allow for swift feature updates and enhancements, significantly improving the software development lifecycle's efficiency. However these pipelines provide an excellent and larger work carrying capacity, these pipelines pose new challenges especially when addressing security issues. The conventional security testing techniques are generally inadequate in providing timely or consistent protection against new threats, thereby allowing vulnerabilities to go unnoticed during development. This growing concern has led to the need to integrate security testing into CI/CD pipelines while maintaining the speed and scalability required for modern software delivery.

Security in CI/CD pipelines has emerged as a critical subject of study over the past several years, based on the relatively well-established field of software quality assurance, as well as secure development methodologies. The earlier techniques for security testing are certainly ad hoc, often taking significant time and work and mostly performed in isolation of development. For the past few years, the robust network security instruments as well as characteristic life cycle have enabled security penetration into the development life cycle. By incorporating security into the DevOps workflow, organizations can guarantee that their applications are secure and devoid of vulnerabilities prior to deployment in the production environment (Ahmed & Francis, 2019) However, there is still a lack of knowledge on how the usage of these tools can be most effectively integrated into constantly growing CI/CD ecosystems. This project aims to address this gap by developing a framework that incorporates security testing into CI/CD pipelines while ensuring a balance between security and delivery speed.

This research is based on previous research in secure software development as well as automated testing. Hence it seeks to offer a realistic approach to meet the needs of software development teams, something that has not been adequately addressed in literature and practice. The problem is that most organizations face difficulties in choosing the correct set of tools and methodologies to solve their problems within security protection systems. This project finding contributes to the general societal challenge of increasing focus on software security without slowing down delivery by identifying the most effective tools and methods and showing how they may be used collectively.

This framework will not only minimize the vulnerability of an application to a security breach, but also be a repeatable model that can be adopted for security testing on any CI/CD pipeline. They should provide practical solutions for developers, security analysts and researchers striving to enhance the interconnection between security and software development speed.

Research Questions and Objectives

The project address one primary research question:

1. How can security testing be effectively integrated into CI/CD pipelines?

To address this question, the project focuses on the following objectives:

- Identify and evaluate leading security tools for static analysis, dynamic analysis, and dependency scanning.
- Develop a reusable framework for integrating these tools into CI/CD workflows.
- Test and validate the framework's effectiveness in detecting vulnerabilities without significantly affecting pipeline efficiency.

Limitations and Assumptions

While aiming to provide a robust security testing framework, this project operates under specific constraints, the scan is limited to open-source software and public repositories because of the resource limitation. Furthermore, the testing environment is artificial, and it does not clearly address all real-world issues like legacy systems and relatively elaborate CI/CD setups.

Key assumptions include the use of standard pipeline infrastructure and adherence to DevSecOps principles by the target audience. Despite these limitations, the framework is designed to offer practical insights and scalable solutions for integrating security testing into modern CI/CD pipelines.

Structure of the Paper

The remainder of the paper is structured as follows:

- **Section 2** provides a literature review of previous work by researchers and practitioners in cloud and pipeline security.
- **Section 3** details the research methodology employed in the project.
- **Section 4** describes the design of the proposed model, including the security techniques and frameworks used to secure the CI/CD pipeline.
- **Section 5** outlines the implementation of the security solutions within the pipeline.
- **Section 6** evaluates the framework, comparing CI/CD pipelines with and without security.

2 Related Work

2.1 Introduction to CI/CD Security Challenges

The adoption of DevOps and CI/D practices have transformed the paradigm of software development and reducing time for new releases cycles and automation for deployments. Still, the acceleration factor and the automated nature of CI/CD pipeline concept are two sides of the coin that increases security threats. This is especially true when new releases are packaged frequently as the application becomes more exposed to these vulnerabilities if security features are not rigorously tested by the pipeline.

DevSecOps[1] aims at continuing the evolution of DevOps security to catch up with the speed of the process. Myrbakken and Colomo-Palacios (2017) note that, although the incorporation of security into DevOps is difficult, the potential rewards are great. Mao et al. (2020) focus on DevSecOps principles targeting security from planning, coding, building and testing phases and infrastructural practices that include secrets management and container security scan. At the same time, Chernyshev et al. (2021) review different approaches to application security such as threat modeling, static and dynamic analysis and software composition analysis, and identified that there are short emergences and making gaps in the real-world practice and advancements.

Sindhu (2021) describes typical challenges in DevOps including fabrication of communication between the development and security team that results in insecure and slow delivery. Extrapolated from the study, one should recommend the DevSecOps approach, PAM, and compliance with the necessary security regulations to minimize threats, and especially when working with clouds. Mohan et al. (2018) and Battina & Sindhu (2017) also reported that there is a lack of effective implementation of security tools in practice. The solutions addressed in these studies are security policies, identity and access management, vulnerability scanning, and continuous monitoring; however, these are also mostly hypothetical.

Rahman et al.,(2022) explains how security has been neglected mostly in CI/CD processes, given that DevOps is aligned with speed and automation. Since security tools used in

[1] https://www.techtarget.com/searchitoperations/definition/DevSecOps

traditional security model would imply human intervention, which is not proper in the context of CI/CD pipeline that takes place every time a software is built and deployed. This paper touches on several crucial aspects of the shift-left approach in terms of developing a security check throughout the developmental phases before products get to the production level.

Garcia et al. (2022) also pointed out that when there are multiple integrations into the code, new threats emerge because new and untested dependencies will be introduced coupled with hasty deployment. Moreover, Sato et al. (2023) described how traditional security solutions cannot address the modern, DevOps-based system development, and therefore, security practitioners must employ agile and automated approaches to perform continuous security evaluations of CI/CD pipelines.

2.2 Existing Approaches for Security in CI/CD

A few automated security testing tools have been created and implemented at the CI/CD pipeline level. The commonly used types of vulnerability identification tools are found in the developers' environment during the coding phase and include Static Application Security Testing (SAST)[2], Dynamic Application Security Testing (DAST), and Dependency Scanning.

ZiYue Pan (2024) conducted a systematic analysis of security threats from open-source CI/CD pipelines, where 327310 GitHub repositories were surveyed. This work highlights a number of attack vectors and then measures the level of risk, which shows that CI/CD pipelines contain code injection and VUs. The scenarios are suggested in the paper together with the detailed threat model and attack techniques, accompanied by five true stories of attacks. It also includes Include methods for addressing these threats for CI/CD configurations and scripts and highlights the requirement for enhancement of the security measures within CI/CD environments.

Mangla (2023) discussed how security tools can be incorporated into CI/CD practices, with a major focus on the automation of misconfiguration. To embed security across the entire CI/CD process, the study takes a DevSecOps approach. It touches on the importance of constant discovery of the security problems at the different stages of the pipeline and presents a guideline for improving on the existing security solutions. Thus, by adopting automation in these areas, the research seeks at enhancing generic security outcomes and minimising security threats inherent to software delivery.

Feio et al. (2024) propose a DevSecOps framework based on the continuous security testing method which can be implemented in different aspects of software development life cycle. The paper pays special attention to the methodology of incorporating security into the development life cycle to eliminate the segregation of development, security, and operations.

[2] https://checkmarx.com/glossary/static-application-security-testing-sast/

CI/CD practices are described here with a pipeline and sequence of activities defined for each step and tools used for automation. A case study is presented that illustrates how using the presented framework helps in the proactive discovery of issues which improve the security of applications within this framework.

2.3 Challenges with Automated Security Testing

The incorporation of AST technologies within the CI/CD approaches has several challenges. One of the most commonly mentioned problems is the high number of false positives in SAST and DAST tools. Similar observations were made by researchers; Kumar et al. (2022) pointed out that most of these instruments tend to detect seemingly innocent code to be a security threat, and therefore, a developer will have to waste time going through these results from the instruments, which goes against the concept of automation.

Another big problem falls under the category of non-functional requirements, where one is performance bottlenecks. Awad et al. (2021) discovered that, incorporating DAST tools within a CI/CD system results in protracted build times because significant time is devoted to performing analysis of running applications. This results in conflict between security and velocity, the latter being a force applied on developers to deliver updates as soon as possible. The last one is scalability which is a big issue for big firms running microservices architecture. As highlighted by Zhao et al. (2021), it becomes almost impossible not to burden large resource costs or slow the release pipeline when attempting to run security tests across numerous services.

Smith et al. (2023) wanted to know how DAST tools affect performance within CI/CD pipelines, something especially important for organizations that use microservices in big and complex environments. They found that DAST tools bring in the problem of scalability the rate at which security testing is conducted decreases as the number of services grows.

Rajapakse et al. (2022) present an approach to integrate three automated dynamic security testing techniques into a CI/CD pipeline and provide an empirical analysis of the introduced overhead. The paper identifies unique research and technology challenges that the DevSecOps community faces and proposes preliminary solutions to these challenges. The findings aim to enable informed decisions when employing DevSecOps practices in agile enterprise application engineering processes and enhance overall enterprise security.

Rangnau et al. (2020) describe a study to understand how the application of dynamic security testing tools may be adopted and performed integrated within CI/CD pipelines. The paper also focuses on the role of security in software development, especially at the current time when its relevance is rapidly increasing in connection with the implementation of DevOps practices. The authors stress that the existing security management approaches cannot compete with the level of application development, and, as a result, a new approach is introduced – DevSecOps, which focuses on integrating security into the DevOps approach.

The study integrates three automated dynamic testing techniques: There is Web Application Security Testing (WAST) using Zed Attack Proxy (ZAP)[3], Security API Scanning (SAS) by JMeter and Behaviour Driven Security Testing (BDST) by using SeleniumBase. The authors give a quantitative assessment of the overhead incurred by these integrations and define novel issues experienced by DevSecOps professionals when incorporating security testing into the CI/CD pipeline. The paper is intended for the result to help development teams in satisfactorily establishing security testing practices for the agile software development environment.

2.4 Novel Approaches for Security Automation in CI/CD

A recent research investigation has discussed machine learning and AI integrated security in the automated security testing aspects in CI/CD pipelines. L. A. Nikolov (2023) deals with the integration of automation in DevSecOps context stressing that it is crucial to participate everyone with automation in order to improve the security. In this paper, authors consider how DevSecOps practices improve identification of issues and implementation of security measures across application development phases. CI/CD integrates automation steps which make security testing also automated in one way or another, and this paper focuses on various approaches to automate security testing aspects in CI/CD pipelines, with further discourse on the discussed problems and solutions so that organizations may have a guide into correctly implementing security into the development cycle. N. M. Grigorieva (2024) devotes this topic to discovering the significance of DevSecOps in strengthening software protection and using new approaches to incorporate protection automation into CI/CD pipelines. This article focuses on the best practices required for automation of security practices that can be adopted for continuing security monitoring as well as vulnerability assessment constituting the main agenda in the formation of a defensive software development life cycle. It raises awareness on the decentralization of automatic security practices within organizations; hence making security an intrinsic part of the software development life cycle rather than an annexation.

A. K Reddy(2021) seeks to analyse how this best practice is included and the changing culture referred to as DevSecOps. It is evident from the study that security should be integrated throughout the CI/CD with no emphasis placed on speed. Applying the synthesising phase of the current research paradigm, the paper outlined the following successful DevSecOps strategy enablers which include; the ‘Security as Code’ concept and culture shift in organisations. It stresses a need to have security tools included in the CI/CD process to reduce vulnerabilities and increase overall interaction among all parties engaged in software development. The results imply that implementing DevSecOps is useful for achieving significant improvements in security, performance, and compliance, pointing to a fruitful line of development for organizations striving to improve the security of the software they produce.

[3] https://www.hackerone.com/knowledge-center/owasp-zap-6-key-capabilities-and-quick-tutorial

2.5 Frameworks Combining SAST, DAST, and Dependency Scanning

There are many frameworks are being developed that consider more than one approach to test security, that is, SAST, DAST, and SCA. A unified security model that includes SonarQube, OWASP ZAP, Snyk[4] is explained in detail by Syed et al. (2021) that is implemented using the CI/CD pipeline with Jenkins. The framework performs security testing throughout the pipeline across the coding stage with SAST, at the running phase with DAST, and finally using SCA to identify optimum vulnerable third parties.

In the same vein, Gupta et al. (2022) presented a security model that encompasses SAST, DAST, and dependency-valve scanning, performed inside the GitLab CI/CD pipeline. In their case study of HospitalRun, – a healthcare management application they developed – they illustrated how a unified framework enhanced the identification of vulnerabilities at various development phases. However, the research demonstrated difficulties in tool configurations and appointed the important question of the fine-tuning to decrease the number of false alarms.

In more detail, Wang et al. (2023) presented an updated approach based on integrating the above-mentioned methods with container security solutions such as Aqua Security and Trivy. This end to end security solution also makes sure that any security flaws will be not only in the host code but also with the containers where the application is hosted on.

2.6 Open-Source Tools for CI/CD Security

In particular, there are number of open-source security tools that are currently used to address security between CI/CD phases. Open-source tools such as OWASP ZAP, SonarQube, Snyk and Trivy works under the SAST, DAST and dependency scanning categorizations, and are widely adopted by developers who want to integrate security in their CI/CD processes.

In another recent project by Chen et al. (2020), the authors discussed integration of OWASP ZAP into Jenkins pipelines with reporting of typical web application threats including SQL injection and cross-site scripting (XSS). But it also shifted the focus to the fact that the resource optimization by ZAP can significantly worsen when the engine is scanning large and complex applications.

Zhao et al., (2021) devoted their attention to specific SAST tool, namely SonarQube which they pointed out as an effective tool of identifying security vulnerabilities at the very early stages of development. But the study raised the argument that static analysis in SonarQube is very likely to produce false positive results especially when working on large applications having many layers of microservice architectures.

The below table lists the open source security tools that can be Integrated with CI/CD.

[4] https://snyk.io/learn/what-is-ci-cd-pipeline-and-tools-explained/

Tools	Integration	Features	Limitations
Jenkins	GitHub, Bitbucket, GitLab	- Highly customizable with 1500+ plugins - Strong community support - Supports distributed builds	- Steeper learning curve for beginners - Maintenance overhead due to plugin management
GitLab CI/CD	GitLab repositories	- Built-in CI/CD pipelines - Auto DevOps for automation - Comprehensive security features	- Limited to GitLab ecosystem - Some advanced features may require higher-tier plans
CircleCI	GitHub, Bitbucket	- Cloud-based automation - Docker support - Auto-scaling capabilities	- Free tier has limitations on build minutes - Configuration can be complex for large projects
Travis CI	GitHub	- Simple YAML configuration - Integrated with GitHub - Free for open-source projects	- Limited support for private repositories in the free tier - Slower build times compared to competitors
GoCD	Various version control systems	- Pipeline visualization - Value stream mapping - Native Docker support	- Can be complex to set up and manage - Less community support compared to Jenkins
GitHub Actions	GitHub	- Automates CI/CD directly from GitHub - Event-based triggers	- Limited to GitHub repositories only - Workflow complexity can increase with larger projects
SonarQube	Various CI/CD tools	- Continuous inspection of code quality - Integrates static analysis	- Requires additional setup for full functionality - Performance may degrade with large codebases
OWASP ZAP	Integrates with CI/CD pipelines	- Dynamic application security testing (DAST) - Automated vulnerability scanning	- May produce false positives - Requires configuration and tuning for optimal results

Table 1: Open-Source tools

This work addresses no framework that use only opensource tools there are numerous other tools are also available that can be integrated into the framework to meet specific requirements, the above table highlights a selection of open-source tools.

3 Research Methodology

The study is based upon the fundamental principles of DevSecOps, provisions of placing security as a forefront practice in software development process. Ahmed and Francis (2019) research showed the need for integrating security in the DevOps workflow to reduce vulnerabilities. Previous works of Kumar & Goyal (2021) and Battina & Sindhu (2017) also state that automated security testing decrease manual efforts and reduce response time to vulnerabilities. To address gaps identified in prior research, this research focuses on developing and implementing a framework for integrating security testing into CI/CD pipelines using automation tools and technologies, the evaluation of the framework's performance in detecting vulnerabilities while maintaining the efficiency of the CI/CD pipeline. The framework includes GitHub Actions and incorporates security tools such as Lint, CodeQL, CodeClimate, Snyk, and OWASP ZAP. The OWASP Juice Shop, a vulnerable application designed for security training, is used as a test case to validate the framework.

Stages of Secure CI/CD pipeline

Continuous Integration

The CI/CD pipeline is designed and built using GitHub Actions. Key configurations of the pipeline includes GitHub repository is used as Source Code Management to store and manage the OWASP Juice Shop application code. GitHub Actions workflows are defined in .yml files to automate CI tasks such as code formatting checks using Lint and static code analysis using CodeQL. The workflows were triggered by events such as code pushes, pull requests, and merges, ensuring continuous integration and testing. Docker was used to containerize application artifacts, which were then pushed to Docker Hub, ensuring reproducibility and consistency across environments.

Continuous Security

A continuous security workflow using GitHub Actions is used to ensure the security of a codebase and to automate security testing. The GitHub Actions workflow was adopted by authors (Benedetti, et al., 2022) and (Kinsman, et al., 2021) and discussed the importance of using GitHub Actions as the latest technology introduced by GitHub to automate the workflows. CodeQL was employed to scan the codebase for vulnerabilities, such as injection flaws and insecure coding practices. Snyk was utilized to identify vulnerabilities in third-party dependencies. OWASP ZAP was integrated into the workflow to simulate attacks and identify vulnerabilities in the running application. The continuous security workflow was configured to automatically notify developers of detected vulnerabilities or failed security checks, enabling prompt remediation. The continuous security workflow was configured to automatically notify developers of detected vulnerabilities or failed security checks, enabling prompt remediation.

Code Analysis

The pipeline is incorporated with multiple tools to perform comprehensive security checks various static code analysis tools were used to scan the codebase for potential vulnerabilities **Lint** Ensured adherence to coding standards and detected formatting issues ,**CodeClimate** measured maintainability and identified technical debt, **CodeQL** Provides deep static analysis for identifying security vulnerabilities in the application’s source code and **Snyk**

Checks for outdated dependencies in the project. These tools operates seamlessly within the GitHub Actions workflows, triggered automatically upon code changes or feature merges.

Continuous Delivery

In this phase, the application code is deployed to a staging environment where it can be tested in a staging environment. The application, after passing security checks, was deployed to a staging environment to simulate production conditions. Docker images are pushed to Docker Hub, enabling efficient delivery of containerized applications. Additional tests are executed in the staging environment to validate application functionality and security.

4 Design Specification

The presented framework for integrating security testing into a CI/CD pipeline enables a secure, automated solution for software deployment. This section gives the detail requirements, the tools utilized, and the architecture supporting the framework while providing an overview of its key components. To tackle problems in modern software development, the framework combines security measures in the CI/CD lifecycle to facilitate secure delivery without sacrificing speed. We spin around it by implementing code testing, security scanning and deployment automation as a series of well-known tools and workflows. GitHub Actions was used as the primary CI/CD automation server to develop the framework. This choice was made based on how extensively it could integrate and adapt to handle workflows, triggered by developer actions such as code commits and or pull requests. The system uses security tools: Lint, CodeQL, Snyk, CodeClimate and OWASP ZAP to carry out static and dynamic code analysis, dependency checks and vulnerability assessments.

In this design, the pipeline is centered around modular stages, each performs a specific functional task in delivery, from code validation to deployment. The stages ensure that at every checkpoint, potential issues whether functional or security-related are identified and resolved promptly.

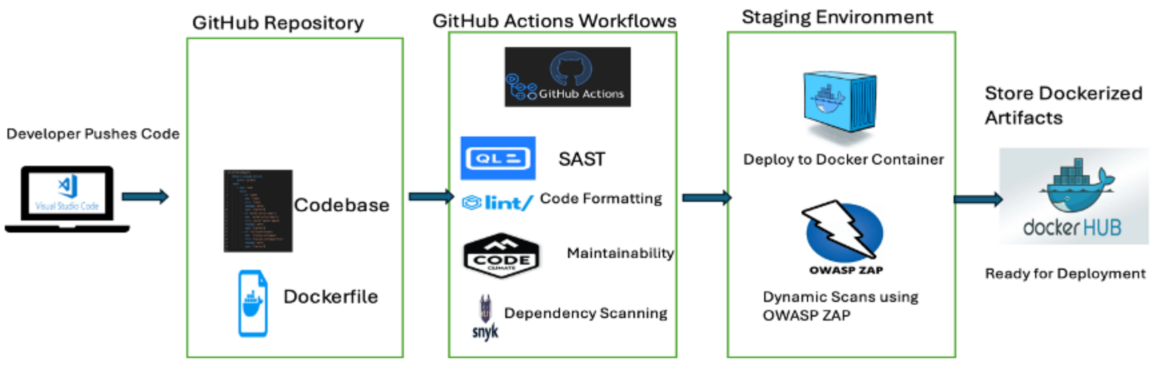


Figure 1:Framework Design

Key Features and Requirements

The design requirements of the framework include:

1. **Automation:** All the stages of the CI/CD process starting from integration, testing and deployment all the stages must be automated to reduce manual intervention.
2. **Security Integration:** Security checks must be conducted at every stage of the pipeline to detect and address vulnerabilities in the early stages.
3. **Reproducibility:** The pipeline ensures consistent results across different environments using Docker for containerization.
4. **Error Reporting:** Errors at any stage are logged and communicated to the development team, ensuring swift remediation.

The framework operates under a layered security model, employing multiple testing methodologies such as static analysis, dynamic analysis, and software composition analysis to ensure comprehensive protection.

Pipeline Workflow Design

The CI/CD pipeline is organized into sequential stages:

1. **Code Validation and Integration:** When developers push code to the GitHub repository, the process triggers a pipeline. The pipeline starts by validating the code using **Lint** to enforce consistent formatting. Following this, **CodeQL** performs static analysis to detect coding flaws and potential security issues like SQL injection or access control misconfigurations.
2. **Dependency Scanning:** Using the tool **Snyk**, the pipeline examines the project's third-party dependencies for known vulnerabilities. This step ensures the software does not inherit security risks from external libraries or modules.
3. **Code Maintainability:** The **CodeClimate** tool assesses code quality, maintainability, and adherence to best practices. It provides developers with actionable insights to improve code robustness and readability.
4. **Dynamic Security Testing:** The application, once built, is deployed into a Dockerized staging environment for runtime analysis. Here, **OWASP ZAP** conducts vulnerability scans to identify potential exploits in a live scenario.
5. **Artifact Packaging and Deployment:** After passing all security and functionality checks, the application is containerized using Docker. The resulting artifact is stored in **Docker Hub**, ready for deployment to production environments.
6. **Monitoring and Reporting:** Post-deployment, the framework monitors the application for runtime vulnerabilities. Logs and metrics are generated to evaluate performance, detect security issues, and provide feedback for continuous improvement.

Toolset Description

1. **GitHub Actions:** GitHub Actions is the pipeline engine that executes the workflows that are described in .yml files in this framework. Its event-driven architecture yields actions depending on changes in a repository, so for a good CI/CD.
2. **Lint:** This tool checks the formatting of code, ensuring production of code which has already been standardized and minimizes on error arising from structuring of codes.

3. **CodeQL:** This tool is a very powerful SAST tool that scans through the codebase in order to identify vulnerabilities and bad practices early on.
4. **Snyk:** Moreover, the SCA of Snyk allows for determining vulnerabilities in project dependencies and suggests an appropriate solution that requires fix.
5. **CodeClimate:** Code health checks, this tool is used for measuring the maintainability of the code that is, areas that requires optimization besides areas that can trigger poor performance.
6. **OWASP ZAP:** The premier choice for a DAST tool, OWASP ZAP functions in a staging environment and probes for vulnerabilities by attempting to attack an application.
7. **Docker:** Docker then facilitates the running of the application in the development, testing and the production environment by encapsulating them within small, light weight and independent modules called containers.
8. **Docker Hub:** This is a cloud based container registry to store and distribute Docker images in a simpler and easier manner.

Tools	Language Support	Provider	Approach
GitHub Actions	All languages supported by GitHub.	GitHub	Event-driven automation with YAML configuration for tasks like building, testing, and deploying.
CodeQL	JavaScript, Python, Java, C++, Go, etc.	GitHub	Uses semantic code analysis to detect vulnerabilities and provide remediation guidance.
OWASP ZAP	Language-agnostic	OWASP	Simulates runtime attacks to identify vulnerabilities like SSRF and XSS.
Snyk	JavaScript, Python, Ruby, Java, etc.	Snyk	Integrates dependency checks and provides automated fix suggestions.
Docker	Language-agnostic	Docker Inc.	Packages application and dependencies into portable containers.
Lint (ESLint)	JavaScript/TypeScript	OpenJS Foundation	Static analysis tool to identify and fix coding standard violations.
CodeClimate	Ruby, JavaScript, Python, Go, etc.	Code Climate	Analyzes codebase for maintainability and test coverage alongside security issues.

Table 2:Security Tools

Each tool was designed to run smoothly in every stage of pipeline, without the need to interact with another tool, and pass on artifacts or results. Adding this modularity helps to maintain the stability and expandability while ensuring security checks are not skipped.. For instance, use of static analysis tools ensures the code is correct before packaging, and dynamic tests that simulate real-world attacks post build. The proposed framework integrates robust security process into the CI/CD process to make sure that application is production ready and also protecting them from threat potential.

5 Implementation

The results obtained using the methodology and proposed design identified various vulnerabilities and addressed them in the early development cycle. The implementation stage involved creating a fully automated CI/CD pipeline that integrates comprehensive security testing at every step. This was achieved using GitHub Actions to define workflows that automated the process of building, testing, securing, and deploying applications. The OWASP Juice Shop application was used as the testbed for the framework due to its rich set of realistic vulnerabilities, making it an ideal choice for validating security processes.

The implementation resulted in several key deliverables. Firstly, custom workflows were written using YAML scripts. These workflows describes the sequence of tasks triggered by specific GitHub events such as code pushes, pull requests, or merges into the main branch. These workflows were configured to perform security scans, quality checks, and automated deployments without requiring manual intervention. By orchestrating these processes within GitHub Actions, the pipeline ensured that all stages of integration and delivery were both repeatable and efficient.

Another critical output of the implementation was the creation of Dockerized application images. The codebase was built and packaged into containerized images that could be deployed across any environment with minimal configuration. These Docker images were rigorously tested for vulnerabilities and then uploaded to Docker Hub, ensuring a consistent and secure deployment process.

The framework also produced detailed security reports. These reports, generated during each pipeline execution, highlighted vulnerabilities found through static and dynamic security testing. Tools such as CodeQL performed static application security testing to identify potential flaws in the source code, while Snyk conducted software composition analysis to detect issues in third-party dependencies. OWASP ZAP was used in the staging environment for dynamic application security testing, simulating real-world attack scenarios to uncover runtime vulnerabilities. These reports were integrated into GitHub's interface, enabling developers to review issues directly within their workflow.

In addition to security outputs, the pipeline also provided insights into code quality. By incorporating tools like CodeClimate, the implementation measured code maintainability, identifying areas of technical debt, complexity, and code smells. These quality metrics allowed the development team to address potential issues early, improving the overall health of the codebase.

All processes were designed to operate seamlessly within the CI/CD pipeline. The implementation ensured that once new code was committed to the GitHub repository, the pipeline would automatically initiate the build process. This involved compiling the application, resolving dependencies, and preparing the Docker image for testing. Automated testing followed, with tools like JUnit verifying the functionality of individual components and integrated security tools ensuring the absence of vulnerabilities.

The entire implementation relied on a robust set of tools and languages. GitHub Actions formed the backbone of the pipeline, orchestrating tasks defined in YAML scripts. The primary language of the application, JavaScript (Node.js), was used alongside custom scripts written in Python for specialized data processing and reporting tasks. Docker played a crucial role in creating consistent, portable application containers, while Docker Hub served as a centralized repository for storing and sharing these images.

In conclusion, the implementation successfully delivered an integrated CI/CD pipeline that automated the development and security testing processes. It achieved the primary goal of embedding security into the software delivery lifecycle, ensuring that every release was tested for vulnerabilities before reaching production. By producing actionable security reports and maintaining high levels of automation, the framework represents a significant step forward in integrating DevSecOps practices into modern software development.

6 Evaluation

This experiment aimed to detect vulnerabilities in every stage of CI/CD Pipeline. The secure module of the CI/CD Pipeline is built by integrating security tools to identify security misconfigurations and vulnerabilities in every stage of the CI/CD pipeline. The evaluation was performed between the CI/CD Pipeline and the Secure CI/CD Pipeline. The pipeline without security does not detect any vulnerabilities. However, this presented pipeline with security detected the security vulnerabilities that benefit compliance and security. The output produced from the security tools and GitHub Actions workflows showed the security misconfigurations and vulnerabilities on OWASP Juice Shop, a deliberately vulnerable web application.

6.1 CI/CD Pipeline Without Security Testing

Without security testing, the CI/CD pipeline only focuses on things like automating the application build, testing, deployment. The pipeline is blind to security flaws in the codebase, dependencies, and the configurations without the proper security tools integrated.

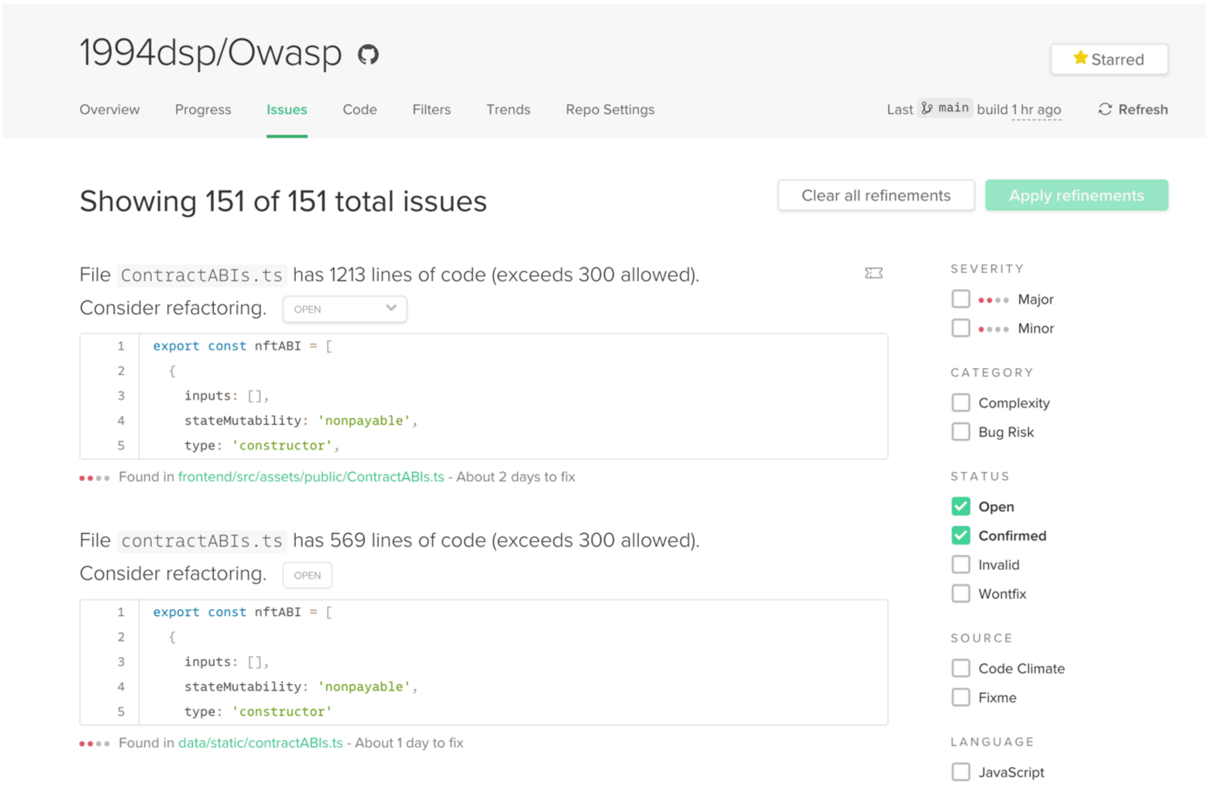
- **Code Quality and Security**: The absence of static analysis tools like Lint or CodeQL which means that issues like syntax errors, code smells, and potential vulnerabilities will go undetected, and which leads the code base to exploit.
- **Dependency Vulnerabilities**: Tools like Snyk and other dependencies scanning tools, which scan dependencies for known vulnerabilities, are not part of the pipeline. This can result outdated libraries used in the applicating with known exploits and putting the application at risk.
- **Vulnerability Assessments**: Without the dynamic analysis tools like OWASP ZAP, which lacks any proactive measures to scan the application for runtime vulnerabilities or misconfigurations in the infrastructure, such as insecure cloud configurations or improper access controls.

The pipeline without the security testing tools leaves the application vulnerable and inefficiencies in detecting and addressing security risks at the early stages of the development cycle.

6.2 CI/CD Pipeline with Security Testing

With help of mentioned above instruments such as Lint, CodeQL, Snyk, CodeClimate, and OWASP ZAP the CI/CD pipeline will transform into very secure innovative automated framework that scans, identify and prevent most of the security threats at each stage of the development process.

Code Analysis and Linting: Lint, CodeClimate, in this pipeline, are used detect syntax error, code smell, and likely issues at the beginning of the SDLC process. These tools help maintain code quality and enforce secure coding practices, reducing the risk of security vulnerabilities introduced during development. The below diagram shows the vulnerabilities identified by Lint scan and the CodeClimate.



Showing 151 of 151 total issues

File `ContractABIs.ts` has 1213 lines of code (exceeds 300 allowed). Consider refactoring. OPEN

```
export const nftABI = [
  {
    inputs: [],
    stateMutability: 'nonpayable',
    type: 'constructor',
```

Found in frontend/src/assets/public/ContractABIs.ts - About 2 days to fix

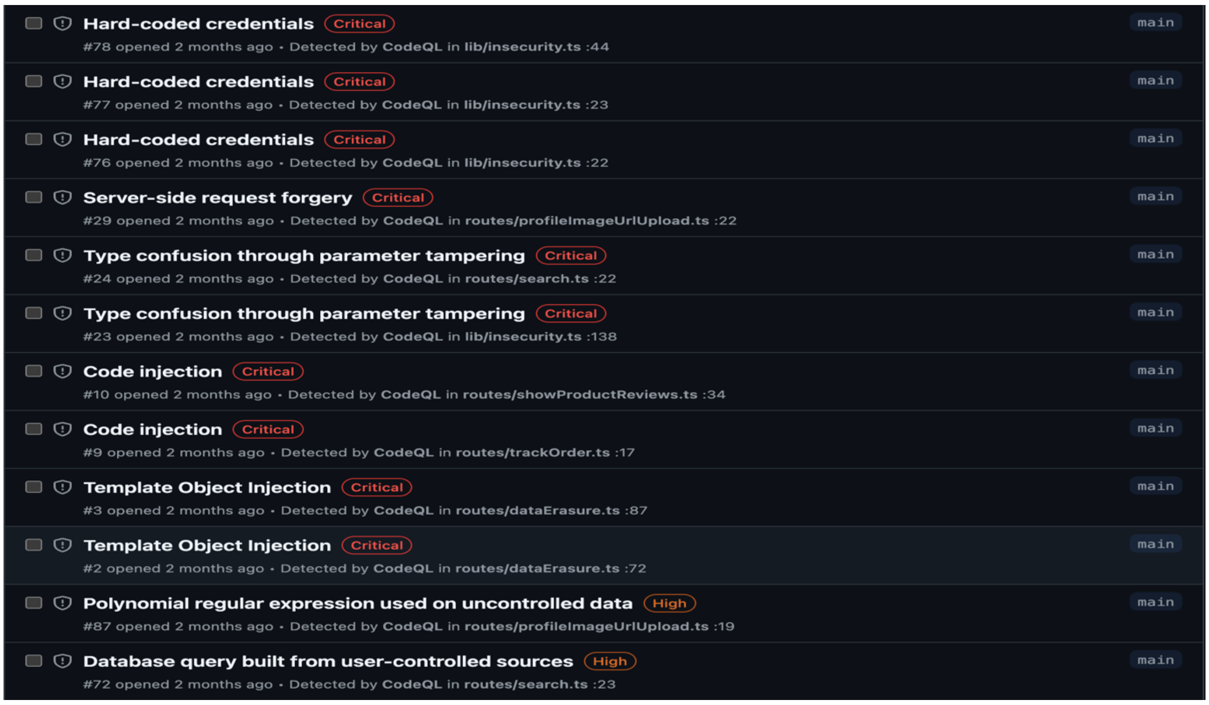
File `contractABIs.ts` has 569 lines of code (exceeds 300 allowed). Consider refactoring. OPEN

```
export const nftABI = [
  {
    inputs: [],
    stateMutability: 'nonpayable',
    type: 'constructor'
```

Found in data/static/contractABIs.ts - About 1 day to fix

Figure 2: CodeClimate Analysis

Static Code Analysis with CodeQL: CodeQL performs deep static code analysis, it detectes the vulnerabilities such as SQL injection, cross-site scripting (XSS), and buffer overflow vulnerabilities. This ensures that the code is fully examined for critical security issues before it reaches production. CodeQl has identified 119 issues on the OWASP Juiceshop application with Hard-Coded Credentials, Server-Side request forgery, Code injection etc.


Hard-coded credentials Critical
#78 opened 2 months ago • Detected by CodeQL in lib/insecurity.ts :44 main

Hard-coded credentials Critical
#77 opened 2 months ago • Detected by CodeQL in lib/insecurity.ts :23 main

Hard-coded credentials Critical
#76 opened 2 months ago • Detected by CodeQL in lib/insecurity.ts :22 main

Server-side request forgery Critical
#29 opened 2 months ago • Detected by CodeQL in routes/profileImageUrlUpload.ts :22 main

Type confusion through parameter tampering Critical
#24 opened 2 months ago • Detected by CodeQL in routes/search.ts :22 main

Type confusion through parameter tampering Critical
#23 opened 2 months ago • Detected by CodeQL in lib/insecurity.ts :138 main

Code injection Critical
#10 opened 2 months ago • Detected by CodeQL in routes/showProductReviews.ts :34 main

Code injection Critical
#9 opened 2 months ago • Detected by CodeQL in routes/trackOrder.ts :17 main

Template Object Injection Critical
#3 opened 2 months ago • Detected by CodeQL in routes/dataErasure.ts :87 main

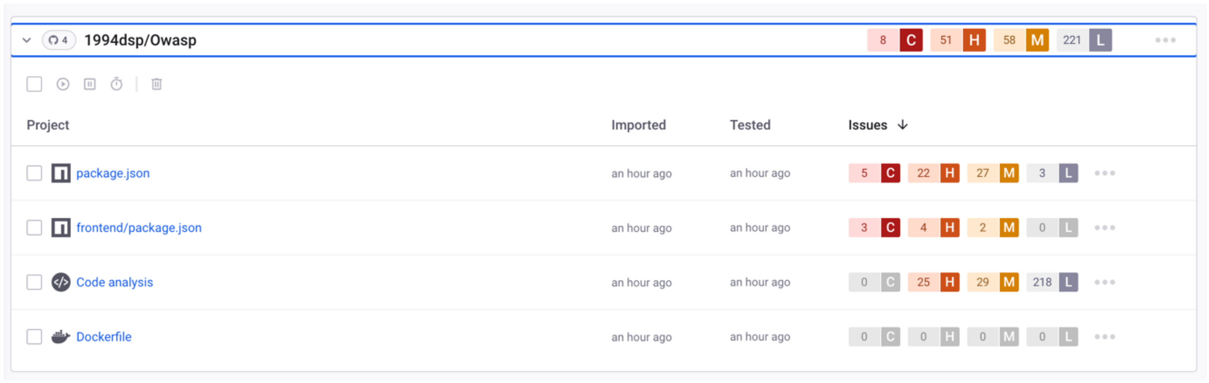
Template Object Injection Critical
#2 opened 2 months ago • Detected by CodeQL in routes/dataErasure.ts :72 main

Polynomial regular expression used on uncontrolled data High
#87 opened 2 months ago • Detected by CodeQL in routes/profileImageUrlUpload.ts :19 main

Database query built from user-controlled sources High
#72 opened 2 months ago • Detected by CodeQL in routes/search.ts :23 main

Figure 3: CodeQl Scan

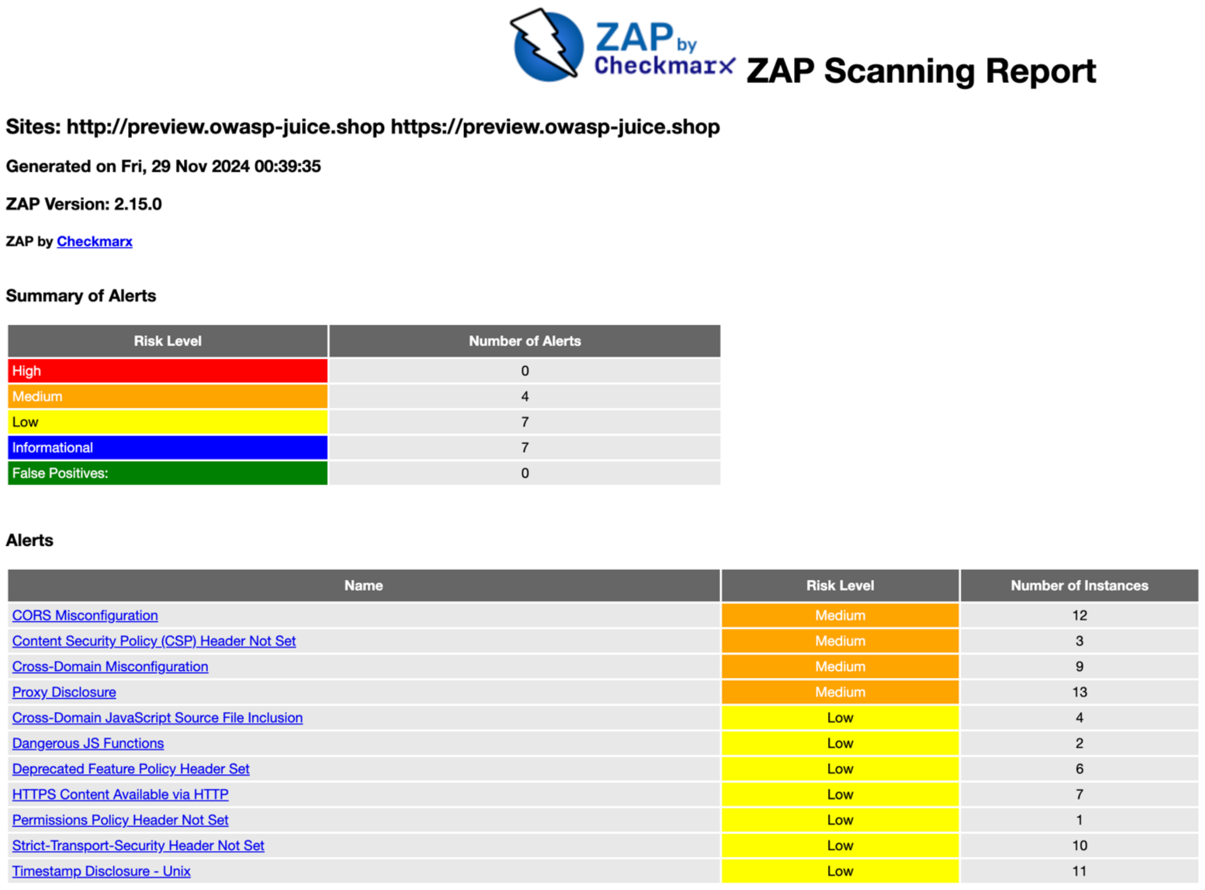
Dependency Management with Snyk: Snyk scans the application to detect a list of all dependencies and run them through matched databases of known vulnerabilities. This tool is specifically essential for the scanned application because third-party libraries, a common risk with modern applications, often have exploitable vulnerabilities. The below figure shows Snyk scan result for the repository.


4 1994dsp/Owasp 8 C 51 H 58 M 221 L

Project	Imported	Tested	Issues ↓
package.json	an hour ago	an hour ago	5 C 22 H 27 M 3 L
frontend/package.json	an hour ago	an hour ago	3 C 4 H 2 M 0 L
Code analysis	an hour ago	an hour ago	0 C 25 H 29 M 218 L
Dockerfile	an hour ago	an hour ago	0 C 0 H 0 M 0 L

Figure 4: Snyk Scan

Security and Vulnerability Scanning with OWASP ZAP: During the dynamic analysis phase, OWASP ZAP simulates an attack on the running Application, more specifically the program searches for cross-site scripting (XSS), broken authentication, and insecure direct object references (IDOR). With ZAP running during staging or pre-production it is possible to identify some of the run time vulnerabilities and rectify them before going live. From the below figure we can see the ZAP scan for OWASP Juice shop done before deploying in to production environment.



ZAP Scanning Report

Sites: http://preview.owasp-juice.shop https://preview.owasp-juice.shop

Generated on Fri, 29 Nov 2024 00:39:35

ZAP Version: 2.15.0

ZAP by Checkmarx

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	4
Low	7
Informational	7
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
CORS Misconfiguration	Medium	12
Content Security Policy (CSP) Header Not Set	Medium	3
Cross-Domain Misconfiguration	Medium	9
Proxy Disclosure	Medium	13
Cross-Domain JavaScript Source File Inclusion	Low	4
Dangerous JS Functions	Low	2
Deprecated Feature Policy Header Set	Low	6
HTTPS Content Available via HTTP	Low	7
Permissions Policy Header Not Set	Low	1
Strict-Transport-Security Header Not Set	Low	10
Timestamp Disclosure - Unix	Low	11

Figure 5: ZAP Scan

Automated Security Workflow: These security tools are seamlessly integrated into the pipeline using **GitHub Actions**, which automates the triggering of scans and vulnerability assessments. With every commit or pull request, the tools run automatically to analyze the code, dependencies, and runtime behavior, ensuring that no new security issues are introduced into the codebase. Additionally, if any security flaws are detected, automated alerts and reports are generated, allowing the team to take immediate corrective action.

6.3 Discussion

Figure 6 represents the results of an experiment that evaluated the effectiveness of tools CodeQl, Synk, Codeclimate and OWASP ZAP in identifying vulnerabilities in a pipeline integrated with security. The results of the experiment showed that all the tools identified a relatively high number of vulnerabilities. The integration of security testing into every stage of the CI/CD pipeline substantially improves the security posture of the application. Compared to a pipeline without security checks, the secure pipeline provides significant benefits in identifying and mitigating vulnerabilities early in the development process. As a result, the application is more secure, compliant, and resilient to potential attacks. By doing so, it was possible to quickly address any issues that were encountered and maintain the overall security of the pipeline over time. In summary, this approach not only strengthens the overall security framework but also enhances the ability to deliver secure software quickly and efficiently.

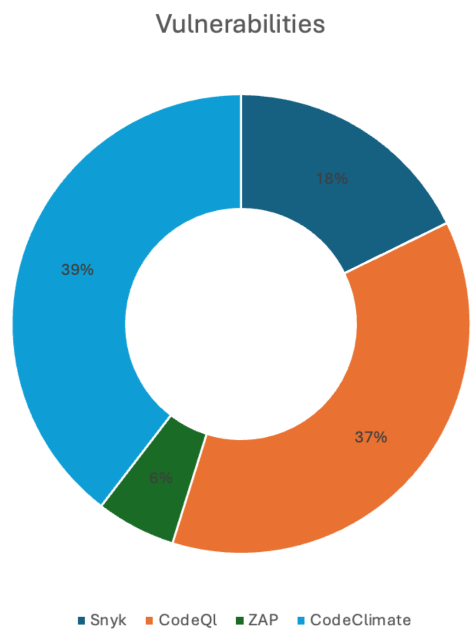


Figure 6: Vulnerabilities Analysis

7 Conclusion and Future Work

To answer the research question, this research employed a framework that integrates the security testing into all stages of the CI/CD process The conclusion thus shows that this study provides proof that security testing at the CI/CD pipeline as proposed in this study can indeed work at the speed and scale needed in the current software development methods.s. The LimeLM product is an effective solution that is integrated with a SA&ST approach that combines SAST, DAST, and Dependency Vulnerability Scanning. This is evidenced by the fact that popular tools Lint, CodeQL, CodeClimate, Snyk, and OWASP ZAP can be integrated into the automated work cycle, which is performed by GitHub Actions.

For proving the practicality and basic sanity of the framework for vulnerability consistency, the evaluation is carried out in OWASP Juice Shop. In comparison to a standard CI/CD pipeline where there is no security integration the secure pipeline serves as a more effective means of analyzing for and identifying vulnerability and misconfigurations at stages in the deployment process which in turn increases security.

The contribution of this research is a reusable, scalable and efficient framework on DevSecOps, aiming to improve the challenge in CI/CD contexts of integrating security into fast development delivery. It gives guidance to other organizations on how they can use a similar approach to enhance application security and mitigate risks at stage one of the development process. Although this research effectively applies security testing to the CI/CD process in this work, further research studies and eventual implementation of other sophisticated security tools like the fuzz testing tools and AI security testing systems could be explored and incorporated to enrich frameworks' functions. This solution might contain functionalities for automatic patching of threats and continuous threat detection and escalation to threats and incidents. When implemented in CI/CD pipeline, this solution can enhance organizations’ security and limit the number of breaches or consequent security incidents.

References

Ahmed, Z. & Francis, S. C., 2019. Integrating Security with DevSecOps: Techniques and Challenges. 2019 International Conference on Digitization (ICD), pp. 178-182.

Battina & Sindhu, D., 2017. BEST PRACTICES FOR ENSURING SECURITY IN DEVOPS: A CASE STUDY APPROACH. International Journal of Innovations in Engineering Research and Technology, pp. 38--45.

Mao, R. et al., 2020. Preliminary Findings about DevSecOps from Grey Literature. 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), pp. 450-457.

Chernyshev, M., Baig, Z. & Zeadally, S., 2021. Cloud-Native Application Security: Risks, Opportunities, and Challenges in Securing the Evolving Attack Surface. Computer, pp. 47-57.

Rahman, N. H. B. M. (2023). Exploring the role of continuous integration and continuous deployment (CI/CD) in enhancing automation in modern software development: A study of patterns, tools, and outcomes. *Quarterly Journal of Emerging Technologies and Innovations, 8*(12), 10–20.

C. Feio, N. Santos, N. Escravana and B. Pacheco, "An Empirical Study of DevSecOps Focused on Continuous Security Testing," 2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Vienna, Austria, 2024, pp. 610-617.

Mangla, M. (2023). Securing CI/CD Pipeline: Automating the Detection of Misconfigurations and Integrating Security Tools. National College of Ireland

Z. Pan *et al*., "Ambush From All Sides: Understanding Security Threats in Open-Source Software CI/CD Pipelines," in *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 1, pp. 403-418.

M. Efendi, T. Raharjo and A. Suhanto, "DevSecOps Approach in Software Development Case Study: Public Company Logistic Agency," 2021 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS, Jakarta, Indonesia, 2021, pp. 96-101.

M. F. Domínguez-Acosta and G. A. García-Mireles, "Identifying Activities for Enhancing Software Quality in DevOps Settings," *2021 10th International Conference On Software Process Improvement (CIMPS)*, Torreón, Coahuila, Mexico, 2021, pp. 84-89.

D. B. Cruz, J. R. Almeida and J. L. Oliveira, "Open Source Solutions for Vulnerability Assessment: A Comparative Analysis," in IEEE Access, vol. 11, pp. 100234-100255.

Fernández González, D., Rodríguez Lera, F.J., Esteban, G. et al. “SecDocker: Hardening the Continuous Integration Workflow.” SN COMPUT. SCI. 3, 80 (2022).

Danilo Sato and Andrei Macedo. Challenges and benefts in implementing continuous delivery: A systematic literature review. 2016 IEEE 9th International Conference on Software Testing, Verifcation and Validation (ICST), pages 13–24, 2016.

L. A. Nikolov and A. P. Aleksieva-Petrova, "Action Research on the DevSecOps Pipeline," 2023 International Scientific Conference on Computer Science (COMSCI), Sozopol, Bulgaria, 2023, pp. 1-6.

T. Rangnau, R. v. Buijtenen, F. Fransen and F. Turkmen, "Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines," *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, Eindhoven, Netherlands, 2020, pp. 145-154.

R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, “Challenges and solutions when adopting DevSecOps: A systematic review,” Information and Software Technology, vol. 141, p. 106700.

N. M. Grigorieva, A. S. Petrenko and S. A. Petrenko, "Development of Secure Software Based on the New Devsecops Technology," 2024 Conference of Young Researchers in Electrical and Electronic Engineering (ElCon), Saint Petersburg, Russian Federation, 2024, pp. 158-161.

A. K. Reddy, Venkat, S. Thota, C. S. Ravi, and M. Bonam, “DevSecOps: Integrating Security into the DevOps Pipeline for Cloud-Native Applications,” Journal of Artificial Intelligence Research and Applications, vol. 1, no. 2, pp. 89–114.

H. Chen, J. Chen, J. Chen, S. Yin, Y. Wu and J. Xu, "An Automatic Vulnerability Scanner for Web Applications," 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Guangzhou, China, 2020, pp. 1519-1524.