Seminar Report

Automated Vulnerability Scanning with Trivy

Pranay Bhatia

MatrNr: 17935037

Supervisor: Jonathan Decker

Georg-August-Universität Göttingen
Institute of Computer Science

September 28, 2024

Abstract

In modern software development, *Continuous Integration* (CI) and *Continuous Deployment* (CD) pipelines play a critical role in automating the testing and deployment processes. This experiment integrates a security scanning mechanism into a GitLab CI/CD pipeline to ensure that applications remain secure throughout the development lifecycle. A custom GitLab Runner is configured to run Trivy, a vulnerability scanner, which evaluates *Yet Another Markup Language* (YAML) files and container images submitted via *Pull Request* (PR). An admission control system automatically accepts or rejects PRs based on the severity of detected vulnerabilities. This pipeline ensures that only secure code is merged, minimizing the risk of security flaws in the production environment. The integration of security scanning into the CI/CD workflow streamlines the deployment process and enhances the overall security posture of applications.

Contents

1 Introduction

In the rapidly evolving landscape of software development, the deployment and management of applications have undergone significant transformations. Traditional deployment models, which relied heavily on physical hardware and monolithic systems, often resulted in inefficient resource utilization and prolonged deployment cycles. This paved the way for virtualization, a groundbreaking technology introduced in the 1970s, which abstracted physical hardware and allowed for more flexible and scalable computing

Building on the principles of virtualization, containerization emerged as a revolutionary approach, providing a more lightweight and efficient method for application deployment. Unlike traditional virtualization, which involves running complete operating systems on virtual hardware, containers encapsulate applications with their dependencies into a single, portable unit. This makes containers exceptionally lightweight and quick to deploy, using shared operating system resources while maintaining isolated environments. [Doc20]: Docker, one of the leading containerization technologies, has become integral to the deployment strategies of major organizations such as Google and Amazon, particularly within their *Platform as a Service* (PaaS) models. [Bha23]

Despite the advantages of containerization, the rise in container usage has brought security concerns to the forefront. Open-source Docker images, while convenient, can be vulnerable to attacks if not properly secured. This was highlighted by incidents such as the discovery of Alpine Docker images in 2019, which were shipped with no password, creating a potential backdoor for attackers. [Cim19] Studies indicate that approximately 60% of organizations have encountered some form of attack due to insecure Docker images. Addressing these security vulnerabilities is crucial to maintaining the integrity and reliability of containerized applications. [SAD19]

To mitigate these risks, this project focuses on integrating security checks into the development pipeline through automated static analysis workflows. Specifically, leveraging GitLab's CI/CD capabilities to perform admission control for infrastructure and validate user-submitted YAML files, through a GitLab pipeline triggered by PRs.

When a user submits a PR that includes new YAML files, our automated pipeline initiates a validation process. This process scans the YAML files for potential defects and vulnerabilities using tools like Trivy, which is renowned for its ability to identify *Common Vulnerabilities and Exposures* (CVE)s in container images and configuration files. Based on the reports generated by Trivy or other analysis tools, the PR is either accepted or rejected, ensuring only secure and compliant configurations are deployed.

This project delivers a Git repository that contains an automated validation workflow. By incorporating Trivy into the pipeline, Report proposes a robust admission control mechanism that not only enhances the security posture but also streamlines the development process. This integration facilitates proactive detection of vulnerabilities and defects, thus reinforcing the security and reliability of containerized applications and *Infrastructure as Code* (IaC) code.

2 Background

This section provides an overview of key foundational concepts relevant to the study. It begins by examining **Docker and Docker containers**, which serve as the backbone of modern containerization technology. This is followed by a discussion on **CI/CD**, outlining the practices of continuous integration and continuous deployment, which streamline the software development process. **Container security** is also explored, emphasizing the importance of securing containerized environments. Finally, other relevant subtopics are introduced to provide additional context and support for the research.

2.1 Docker and Docker Containers

Docker is a widely used platform for developing, shipping, and running applications. By utilizing containerization technology, Docker ensures that software runs consistently across different computing environments, whether it is a developer's local machine or a production server. The platform provides a set of tools that automate the deployment of applications inside lightweight, portable containers. This allows developers to create, test, and deploy their applications faster and more efficiently without worrying about environmental discrepancies.

Docker containers encapsulate an application and its dependencies into a single, isolated unit. Unlike traditional virtual machines, containers share the host operating system's kernel but run in isolated processes, which enhances efficiency and performance. This architecture enables containers to consume fewer system resources while providing a consistent execution environment for applications, regardless of the underlying infrastructure. [Esc+16] As a result, Docker containers are highly scalable and ideal for modern cloud-native applications.

2.2 CI/CD

CI is a fundamental development practice where developers frequently integrate their code into a shared repository. Each integration is followed by an automated build and test process, allowing teams to identify and address issues early in the development cycle. By promoting early detection of bugs and inconsistencies, CI helps maintain the stability of the project and reduces the complexity of later stages of development. [Zam+21]

CD takes CI one step further by automating the process of deploying code changes to production environments once they pass integration and testing phases. With CD, new features, bug fixes, and updates can be delivered quickly and reliably to end users, enhancing agility in software delivery. Together, CI and CD form a pipeline where code changes flow seamlessly from development to production with minimal manual intervention, ensuring high-quality releases. [Zam+21]

2.3 Container Security

While containers offer numerous benefits in terms of efficiency and portability, they also present specific security challenges. Since containers share the host *operating system* (OS) kernel, improper isolation can pose security risks, especially if vulnerabilities are present in the underlying system. Additionally, the use of insecure or untrusted container images and configurations can expose applications to potential exploits. [Sax23]

To mitigate these risks, it is essential to follow container security best practices. Regularly scanning container images for vulnerabilities helps ensure that known security flaws are addressed before deployment. Using trusted and official images from reputable sources reduces the likelihood of incorporating malicious or vulnerable software components.

2.4 Other Relevant Subtopics

IaC is a pivotal concept in modern infrastructure management, where computing infrastructure is managed and provisioned through machine-readable configuration files rather than manual processes. IaC allows for greater consistency and repeatability in infrastructure changes, as configurations can be version-controlled and automated. This approach reduces the potential for human error and ensures that infrastructure is always aligned with the desired state.

Kubernetes, an open-source platform for managing containerized applications, has become a cornerstone in container orchestration. It automates the deployment, scaling, and management of containers across clusters of machines. Kubernetes offers a robust ecosystem for ensuring that containerized applications are highly available, scalable, and resilient. [Kub19]

Lastly, **GitLab CI/CD** provides a powerful tool for automating the software delivery lifecycle. Through pipelines defined in the `.gitlab-ci.yml` file, GitLab facilitates automated testing, building, and deployment of applications. By integrating CI/CD practices within GitLab, development teams can streamline the entire software delivery process, ensuring that code changes are thoroughly tested and deployed with minimal manual effort, reducing risks and accelerating the release of new features.

3 Literature Review

This section examines the mechanisms and capabilities of Trivy, a comprehensive security scanner frequently employed to identify vulnerabilities, secrets, and misconfigurations within containerized environments. The review focuses on how Trivy utilizes its internal database for efficient threat detection and its scanning behavior across various categories. Particular emphasis is placed on Trivy's approach to secret scanning, which is critical for preventing sensitive data exposure. By understanding these components, Trivy's role as a vital tool in modern security practices is highlighted.

3.1 Different types of scans

This subsection provides an overview of the different types of scans that are essential in maintaining the security and compliance of containerized environments. These scans, including secret scanning, vulnerability scanning, license scanning, and misconfiguration scanning, play a pivotal role in identifying potential risks and ensuring robust security practices throughout the software development lifecycle.

Secret Scanning

Trivy's secret scanning capabilities provide a robust solution for detecting exposed credentials, such as passwords, *Application Programming Interface* (API) keys, and tokens, within container images, file systems, and git repositories. This feature is enabled by default and is governed by a set of built-in rules, targeting common sensitive data types, including *Amazon Web Services* (AWS) access keys, *Google Cloud Platform* (GCP) service account credentials, and personal access tokens from GitHub and GitLab. By examining plaintext files, Trivy ensures that sensitive information is identified and mitigated before it can be exploited. [22c]

Vulnerability Scanning

Trivy is designed to detect known vulnerabilities in software components within various scan targets, such as container images, OS packages, language-specific packages, and Kubernetes components. Its approach involves consuming security advisories from reliable data sources specific to each type of package, such as vendor databases for OS packages.[23] This data source selection is essential to prevent false positives, as OS vendors often backport security fixes. The severity of detected vulnerabilities is also determined based on the source, providing more accurate risk assessments compared to general databases like the NVD. [24b]

Misconfiguration Scanning

In addition to vulnerability detection, Trivy also includes misconfiguration scanning to identify security and compliance issues within IaC files. It supports popular frameworks such as Kubernetes, Docker, Terraform, and CloudFormation, automatically detecting mixed types of IaC files within the specified directory and applying the relevant checks. [22b] This flexibility allows Trivy to address potential configuration flaws across a wide range of environments, enhancing security by identifying risks inherent in infrastructure setup and deployment.

License Scanning

Trivy provides a license scanning feature that analyzes container images for license files and assesses associated risks. By classifying licenses into categories such as “Forbidden,” “Restricted,” and “Permissive” based on the Google License Classification, [22a] Trivy offers a clear risk assessment for compliance purposes. It scans packages installed via popular package managers like pip, npm, and apt-get by default, and extended scanning can be enabled to inspect source code files and documents for further license violations. This ensures that organizations remain compliant with licensing requirements, avoiding legal risks associated with improper software usage.

3.2 Mechanism of Trivy

Now knowing all what Trivy do, it is important to know what is underlying mechanism, focusing on how container scanning tools operates to identify security risks. It covers key aspects such as open source database utilization, which enables efficient vulnerability detection, its behavior in scanning and detecting issues, and its specialized approach to secret scanning, which helps safeguard sensitive information

Database Utilization

Trivy operates by leveraging multiple specialized databases to provide comprehensive vulnerability scanning across various types of software components. Upon initiating a scan, Trivy automatically downloads the necessary vulnerability databases, which are cached locally for future use. The main vulnerability database, which is updated every six hours on GitHub, contains vulnerability information collected from a range of trusted data sources, including OS vendor advisories and community-maintained repositories.[24b] Trivy also uses a separate Java Index Database when scanning JAR files, allowing it to accurately identify JAR file metadata such as groupId, artifactId, and version. These databases are critical to Trivy's ability to provide accurate and up-to-date vulnerability assessments. [Tiw23]

Handling Unspecified Versions

Trivy also handles packages with unspecified versions cautiously by skipping vulnerability detection for these packages to avoid false positives. However, users can opt for the comprehensive detection priority to scan for vulnerabilities even in unspecified version ranges, where the minimum version in the range is used for detection.[R+24] While this may increase the likelihood of false positives, it serves as a proactive measure to detect vulnerabilities that could otherwise be missed in ambiguous version scenario.

Secret Scanning

In addition to vulnerability detection, Trivy is capable of scanning for secrets within file systems using predefined regular expressions (regex) to identify sensitive information such as passwords, API keys, and tokens. This secret scanning feature is enabled by default and works across container images, filesystems, and git repositories. Trivy applies built-in rules to search for common types of credentials, such as AWS access keys, GCP service account credentials, and GitHub and GitLab personal access tokens.[22c] By examining plaintext files, Trivy helps organizations detect exposed secrets early, mitigating the risk of accidental leakage of sensitive information before it is deployed or made publicly accessible. This regex-based approach ensures a wide coverage of potential security risks by identifying credentials embedded in various locations across the filesystem. [Sax23]

4 Motivation and Problem Statement

4.1 Motivation

With the rise of containerization comes a new set of security concerns. Containers, while offering a lightweight and flexible deployment model, introduce unique vulnerabilities, particularly in container images and configuration files. Misconfigurations and insecure container images are common entry points for security breaches, which can lead to significant incidents. As a result, it becomes crucial to ensure that containers are deployed securely in production environments.

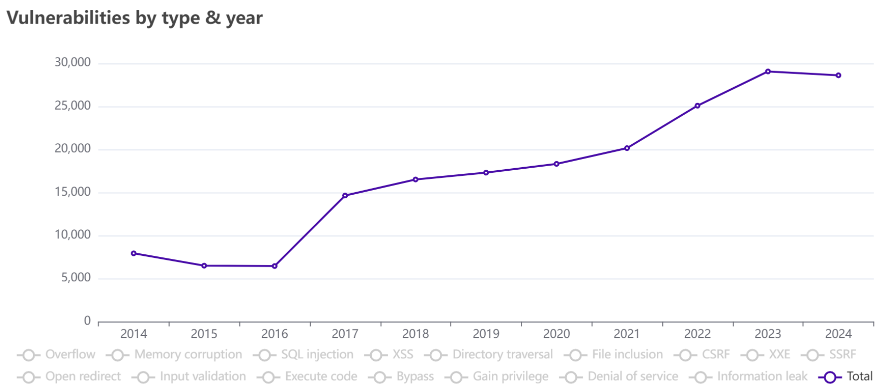


Figure 1: Graph of vulnerabilities per year Source: [24a]

Figure 1 illustrates the trend of vulnerabilities by type and year from 2014 to 2023. The graph shows a significant rise in the total number of vulnerabilities. The annual growth rate of vulnerabilities from 2014 to 2023, based on the graph, is approximately 16.65%. This indicates a substantial yearly increase in the number of reported vulnerabilities over the period of 10 years.

As development cycles shorten and deployment frequencies increase, manually conducting security checks is no longer a viable approach. To mitigate the risk of security vulnerabilities while keeping up with the fast-paced development environment, there is an increasing demand for automated security measures. Integrating automated vulnerability scanning into the CI/CD pipeline is essential for continuous enforcement of security policies. This approach allows for real-time detection and mitigation of vulnerabilities, ensuring that security is maintained consistently throughout the development lifecycle.

4.2 Problem Statement

Maintaing secure Kubernetes deployments is a significant challenge due to the dynamic and complex nature of containerized environments. The rapid proliferation of microservices and the increasing use of containers have resulted in a heightened prevalence of vulnerabilities within container images and YAML configuration files. These vulnerabilities, if left unchecked, pose serious risks to the security of deployed applications and can result in breaches, data loss, and system compromise.

This problem can be solved by an automated admission control mechanism that can perform continuous scanning and validation of container images and configuration files before they are deployed. Such a system would act as a gatekeeper, ensuring that only secure and compliant images and configurations are allowed into production environments. By preventing the deployment of vulnerable containers and configurations, organizations can significantly enhance their overall security posture.

The objective of this project is to design and implement an automated CI/CD pipeline within GitLab that incorporates vulnerability scanning tools such as Trivy. This pipeline will serve as an admission control system, automatically scanning YAML files and container images during the CI/CD process. Based on the results of these security scans, the pipeline will either approve or reject pull requests, thereby enforcing security validation before deployment.

By automating the security validation process, this project addresses the critical need for continuous and proactive security enforcement within Kubernetes environments. The proposed solution will reduce the likelihood of deploying vulnerable configurations, thereby improving the security of the application deployment process.

5 Comparison of different Tools

Based on the problem statements and motivation discussed in previous sections, there is a pressing need for a robust container security scanner to ensure a secure deployment lifecycle. Numerous tools are available for this purpose; thus, this section provides a comprehensive analysis and comparison of several popular container security solutions. 2 It examines their key features, scope of usage, and compliance capabilities, highlighting both open-source and commercial options. Each tool presents a unique approach to container security, enabling organizations to choose the most suitable solution to address their specific security challenges effectively.

aqua trivy
https://hub.docker.com/r/aquasec/trivy
clair
www.okyaytechald.com
Project QUAY
https://www.projectquay.io/
anchore
Anchore Engine:
https://catalog.redhat.com/software/container-stacks/detail/5e9872c4d4ae96b7493f08e1
Twistlock.
improves-container-security-and-compliance/
BLACKDUCK BY SYNOPSYS
https://www.linkedin.com/pulse/semi-automating-blackduck-brit-glazer/

Figure 2: Container security tools for comparison

5.1 Vulnerability Scanning Approaches

A common threat among the examined tools is their focus on vulnerability scanning. Trivy, Clair, and Quay Security Scanner represent the open-source spectrum of vulnerability scanners. These tools primarily focus on identifying known vulnerabilities in container images, with Trivy extending its capabilities to file systems and Git repositories.[Red19] While effective for basic security needs, they generally lack advanced compliance features.

In contrast, commercial solutions like Twistlock (now part of Prisma Cloud by Palo Alto Networks) and Black Duck by Synopsys offer more comprehensive vulnerability management. These tools not only scan for vulnerabilities but also provide broader security features and compliance management capabilities. Twistlock, for instance, includes runtime protection alongside vulnerability scanning, offering a more holistic approach to container security. [Whi24]

5.2 Compliance Management Features

The tools exhibit significant differences in their approach to compliance management. Open-source tools like Trivy and Clair focus primarily on vulnerability detection, which indirectly supports compliance by ensuring images are free from known security flaws. However, they lack built-in compliance enforcement features.

Anchore Engine bridges the gap between open-source and commercial solutions by offering both vulnerability scanning and compliance policy enforcement. It supports customizable compliance policies and detailed reporting, making it a versatile option for organizations with specific compliance needs. [Gup21]

At the commercial end of the spectrum, Twistlock and Black Duck offer extensive compliance features. Twistlock provides support for various industry standards such as PCI DSS and HIPAA, while Black Duck focuses on open-source license compliance and security standards. [Whi24] These tools are particularly suited for large enterprises with complex compliance requirements.

5.3 Integration Capabilities

Integration with existing development and deployment pipelines is a crucial factor in the adoption of container security tools. Most of the examined tools offer integration capabilities with CI/CD pipelines, allowing for automated scanning and policy enforcement. Trivy and Clair are noted for their ease of integration, making them popular choices in DevSecOps environments [Tiw23]. Trivy downloads Database into pipeline container before the scan start from Trivy's hosted database. Which is further used to scan git project 3

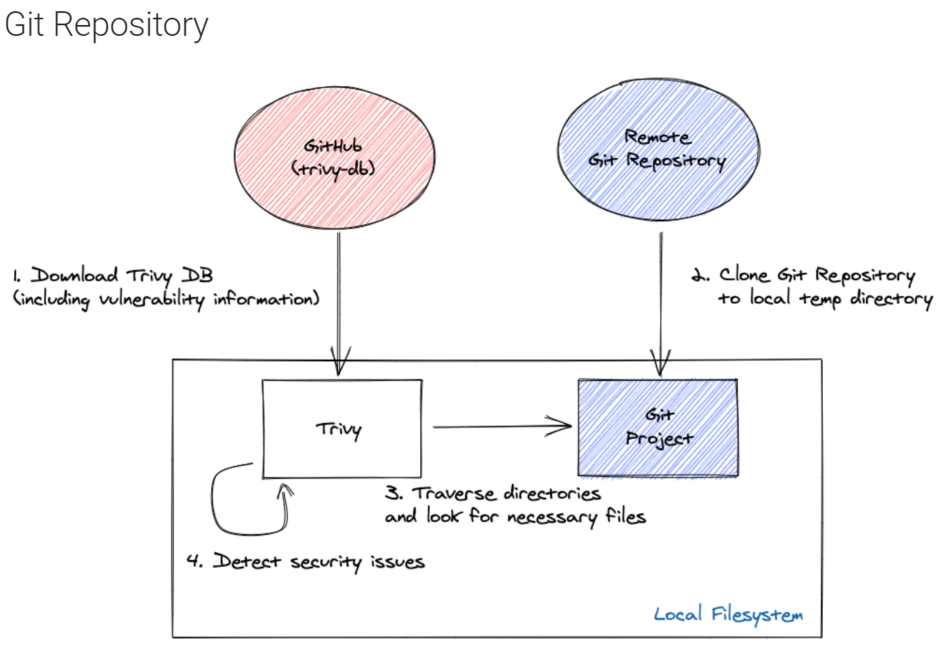

Git Repository
GitHub (trivy-db)
Remote Git Repository
1. Download Trivy DB (including vulnerability information)
2. Clone Git Repository to local temp directory
Trivy
Git Project
3. Traverse directories and look for necessary files
4. Detect security issues
Local Filesystem

Figure 3: Gitlab integration with trivy [blu21]

Quay Security Scanner takes a unique approach by integrating directly with the Quay container registry, offering seamless vulnerability scanning for users of this specific platform. Commercial solutions like Twistlock and Black Duck provide deeper integration capabilities, extending beyond CI/CD pipelines to cover various aspects of cloud-native environments.

5.4 Usage Scenarios and Target Audiences

The tools cater to different usage scenarios and target audiences. Open-source tools like Trivy, Clair, and Anchore Engine are free and user-friendly, making them suitable for organizations looking for cost-effective solutions or those just beginning to implement container security practices.

Quay Security Scanner, while free, is specifically targeted at users of the Quay registry, highlighting the importance of considering existing infrastructure when choosing security tools.

Commercial solutions like Twistlock and Black Duck are geared towards larger enterprises with more complex security and compliance needs [Com24][Whi24]. These tools offer comprehensive features but come with associated costs, making them more suitable for organizations willing to invest significantly in their security infrastructure.

5.5 Conclusion

The landscape of container security tools offers a range of options to suit various organizational needs and budgets. While open-source tools provide effective vulnerability scanning capabilities, commercial solutions offer more comprehensive security and compliance features. The choice of tool depends on factors such as the organization's size, compliance requirements, existing infrastructure, and budget constraints.

Tool	**Type**	**Compliance**	**Cost**	**Integration**
Anchore Engine	Open-source	Yes	Free	CI/CD
Clair	Open-source	Limited	Free	CI/CD
Trivy	Open-source	No	Free	CI/CD
Quay Security Scanner	Open-source	Limited	Free	Quay registry
Black Duck by Synopsys	Commercial	Yes	Paid	APIs
Twistlock (Prisma Cloud)	Commercial	Extensive	Paid	-

Table 1: Comparison of Container Security Tools

Section 6 transitions into the architecture and experimental setup that underpins the implementation of the chosen tools.

6 Architecture and Experiment

This chapter provides a detailed diagram of the workflow, along with the configuration of the GitLab CI/CD pipeline and the utilization of Docker for the GitLab Runner. Furthermore, it explores the reports generated by Trivy and the criteria for making decisions on pull requests based on scan results.

The experiment involves the development of a comprehensive GitLab CI/CD pipeline that integrates security scanning as a core component. The pipeline is designed to detecti vulnerabilities in YAML files and container images submitted in PRs. The security scans are facilitated through the integration of Trivy within the pipeline. The process begins with the setup of a GitLab Runner, which is configured to execute the jobs defined in the pipeline. The pipeline itself is configured using the '.gitlab-ci.yml' file, where the stages and jobs that trigger the security scans are outlined. The code to the experiment can be found at: https://github.com/pranay-bh/trivy

The primary focus of the workflow is to implement an admission control mechanism that determines the acceptance or rejection of PRs based on the results of the security scans. This process assesses the severity and number of vulnerabilities detected, enforcing predefined criteria for accepting or rejecting PRs. If critical vulnerabilities are found, the PR is rejected, otherwise, it can be accepted and merged.

6.1 Diagram of Workflow

The CI/CD pipeline workflow can be visualized in a diagram that illustrates the entire process, from the submission of a pull request to the final decision.

Firgure 4 highlights the key stages of the pipeline: scanning, report generation, and decision-making. The integration points for Trivy are clearly marked, showing how security scans influence the acceptance or rejection of a PR. Additionally, a PR lifecycle

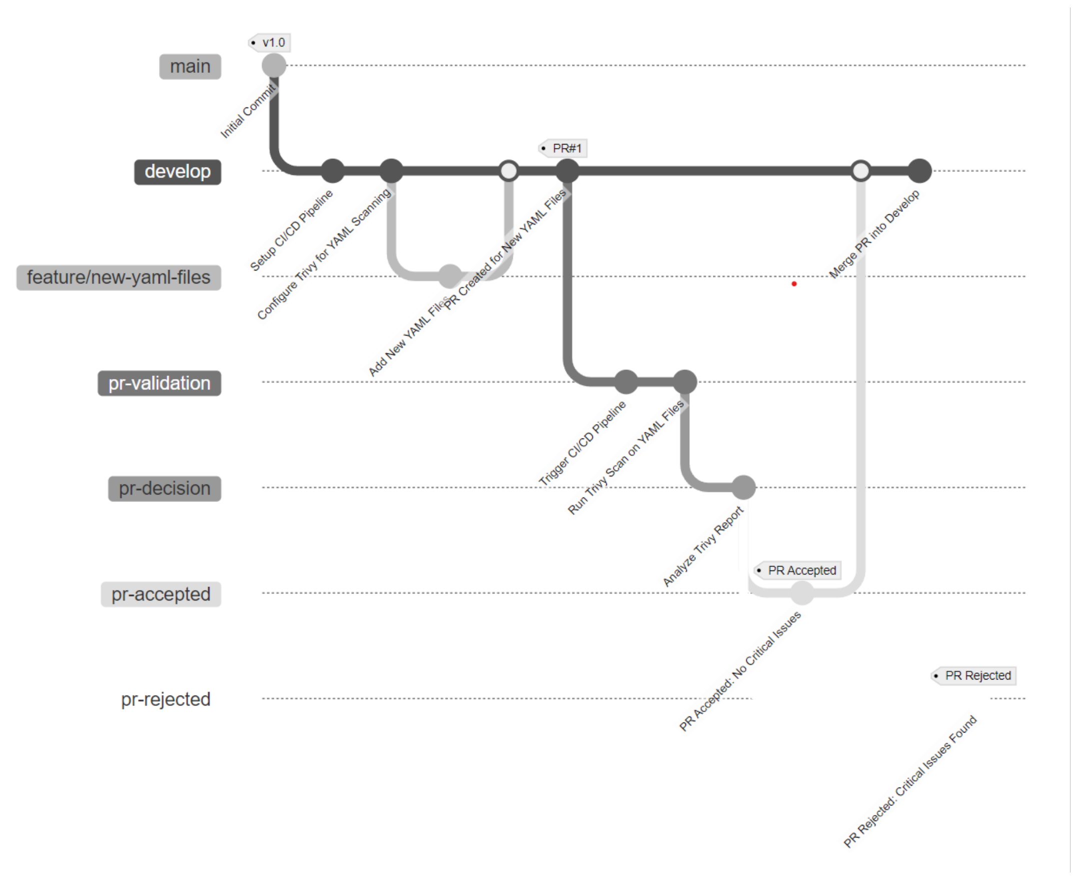


Figure 4: Merge Request Workflow

diagram is included, which depicts the flow of actions when a PR is submitted. This includes an initial scan of the submitted YAML files, generation of security reports, and automated decision-making based on the scan results. The flowchart provides possible outcomes for the PR, demonstrating the conditions under which it is accepted or rejected.

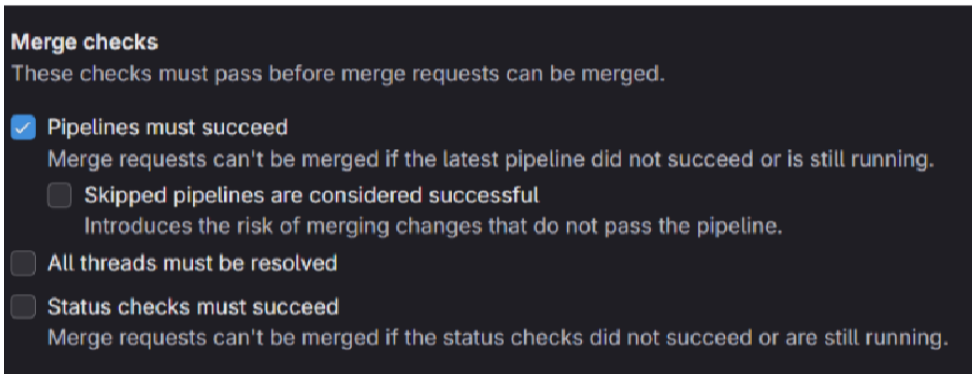
6.2 GitLab CI/CD Pipeline Configuration

The configuration of the pipeline is defined in the '.gitlab-ci.yml' file, which outlines the structure and content required to implement the security scanning process. The pipeline is divided into multiple stages, such as build, scan, report, and deploy. Each stage is executed sequentially, with the Trivy scan integrated within the pipeline to assess vulnerabilities in YAML files and container images.

The implementation of pipeline merge checks introduces a critical security control mechanism. By mandating the successful completion of all pipeline jobs as showin in Figure 5, including Trivy container security scans, prior to code merging, this system effectively prevents the integration of code with high-severity security vulnerabilities, thereby maintaining the integrity of the codebase.

6.3 GitLab Runner Dockerfile

A custom GitLab Runner is necessary to support the specific requirements of the security scanning pipeline. The GitLab Runner is responsible for executing the jobs defined in the pipeline, and a custom Dockerfile is used to configure the environment in which the runner



Merge checks
These checks must pass before merge requests can be merged.

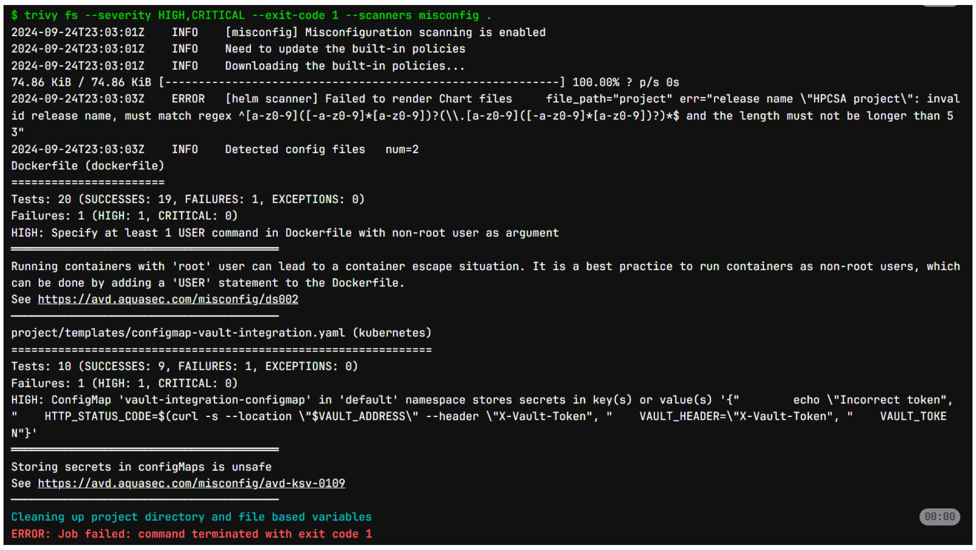
- [✓] Pipelines must succeed
  Merge requests can't be merged if the latest pipeline did not succeed or is still running.
  - [ ] Skipped pipelines are considered successful
    Introduces the risk of merging changes that do not pass the pipeline.
- [ ] All threads must be resolved
- [ ] Status checks must succeed
  Merge requests can't be merged if the status checks did not succeed or are still running.

Figure 5: Gitlab Merge Request Confiuration

operates. The Dockerfile includes the base image, dependencies, and tools required to run the Trivy scanner and handle security reports. After building the custom runner image, it is deployed to the GitLab environment, where it is registered and configured to work with the GitLab instance. This ensures that the runner can execute the security scans and report generation jobs efficiently. For detailed code and configuration refer Appendix Figure 2.

6.4 Reports Generated by Trivy

Trivy generates multiple types of reports during the security scanning process, including vulnerability and misconfiguration reports. These reports are formatted to provide detailed information on the findings, such as the severity level of detected vulnerabilities as seen in figure 6. The pipeline analyzes these reports and uses the information to make decisions regarding the PR. For instance, the severity of vulnerabilities, classified as critical, high, medium, or low, plays a key role in determining whether a PR should be accepted or rejected.



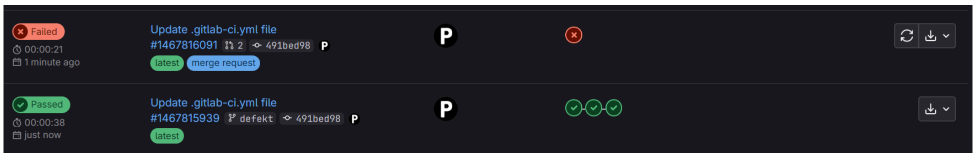
```
$ trivy fs --severity HIGH,CRITICAL --exit-code 1 --scanners misconfig .
2024-09-24T23:03:01Z    INFO    [misconfig] Misconfiguration scanning is enabled
2024-09-24T23:03:01Z    INFO    Need to update the built-in policies
2024-09-24T23:03:01Z    INFO    Downloading the built-in policies...
74.86 KiB / 74.86 KiB [----------------------------------------------------------] 100.00% ? p/s 0s
2024-09-24T23:03:03Z    ERROR   [helm scanner] Failed to render Chart files     file_path="project" err="release name \"HPCSA project\": inval
id release name, must match regex ^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$ and the length must not be longer than 5
3"
2024-09-24T23:03:03Z    INFO    Detected config files   num=2
Dockerfile (dockerfile)
=======================
Tests: 20 (SUCCESSES: 19, FAILURES: 1, EXCEPTIONS: 0)
Failures: 1 (HIGH: 1, CRITICAL: 0)
HIGH: Specify at least 1 USER command in Dockerfile with non-root user as argument
════════════════════════════════════════
Running containers with 'root' user can lead to a container escape situation. It is a best practice to run containers as non-root users, which
can be done by adding a 'USER' statement to the Dockerfile.
See https://avd.aquasec.com/misconfig/ds002
────────────────────────────────────────

project/templates/configmap-vault-integration.yaml (kubernetes)
===============================================================
Tests: 10 (SUCCESSES: 9, FAILURES: 1, EXCEPTIONS: 0)
Failures: 1 (HIGH: 1, CRITICAL: 0)
HIGH: ConfigMap 'vault-integration-configmap' in 'default' namespace stores secrets in key(s) or value(s) '{"        echo \"Incorrect token",
"    HTTP_STATUS_CODE=$(curl -s --location \"$VAULT_ADDRESS\" --header \"X-Vault-Token", "    VAULT_HEADER=\"X-Vault-Token", "    VAULT_TOKE
N"}'
════════════════════════════════════════
Storing secrets in configMaps is unsafe
See https://avd.aquasec.com/misconfig/avd-ksv-0109
────────────────────────────────────────

Cleaning up project directory and file based variables                                                                            00:00
ERROR: Job failed: command terminated with exit code 1
```

Figure 6: Report showing failures

Example reports (Figure 6) are provided to demonstrate the typical output generated by Trivy, highlighting significant findings that influence the automated decision-making process within the CI/CD pipeline.

6.5 Decision on PR Based on Scan Results

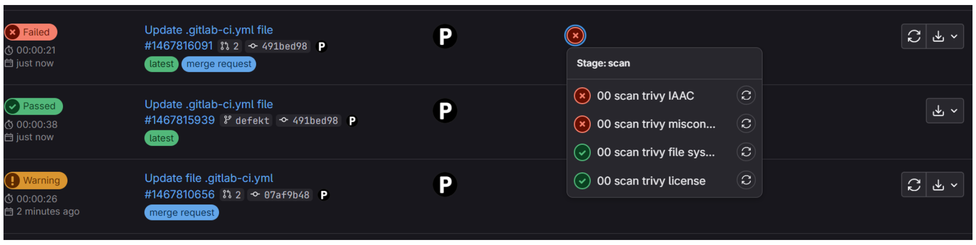
The criteria for accepting or rejecting a PR are clearly defined based on the severity and number of vulnerabilities found during the Trivy scans. Critical vulnerabilities result in the automatic rejection of the PR, while less severe findings may still allow the PR to be accepted. The automated decision-making process is integral to the pipeline, providing immediate feedback to developers by posting the scan results as comments on the PR.



Failed
00:00:21
1 minute ago
Update .gitlab-ci.yml file
#1467816091 2 491bed98 P
latest merge request

Passed
00:00:38
just now
Update .gitlab-ci.yml file
#1467815939 defekt 491bed98 P
latest

(a) Merge Request Trivy Scan Failure



Failed
00:00:21
just now
Update .gitlab-ci.yml file
#1467816091 2 491bed98 P
latest merge request

Passed
00:00:38
just now
Update .gitlab-ci.yml file
#1467815939 defekt 491bed98 P
latest

Warning
00:00:26
2 minutes ago
Update file .gitlab-ci.yml
#1467810656 2 07af9b48 P
merge request

Stage: scan
00 scan trivy IAAC
00 scan trivy miscon...
00 scan trivy file sys...
00 scan trivy license

(b) IaC & Misconfiguration Scan Failure

Figure 7: Trivy and IaC scan failures.

In the case of rejected PRs 7, actionable feedback is provided to guide the submitter in resolving the identified vulnerabilities. Continuous improvement is encouraged by regularly updating the vulnerability databases and refining the acceptance criteria, ensuring that the system remains effective in handling evolving security threats.

7 Result: Trivy Scan Overview

Result section will discuss a summary of security and misconfiguration issues detected in the project directory during a Trivy scan. The scan focused on high and critical severity issues.

The scan was executed using the following command:

```
$ trivy fs ./project \
    --severity HIGH,CRITICAL\
    --exit-code 1\
    --scanners misconfig
```

7.1 Detected Configuration Files

The following configuration files were scanned:

- templates/configmap-vault-integration.yaml

- templates/deployment-myapp.yaml
- templates/deployment-vault.yaml

7.2 Detected Issues

File	**Severity**	**Description**
templates/configmap-vault-integration.yaml **Reference:** AVD-KSV-0109	HIGH	Storing secrets in ConfigMap is unsafe.
templates/deployment-myapp.yaml **Reference:** KSV-014	HIGH	Container ‘myapp‘ should set ‘securityContext.readOnlyRootFilesystem‘ to true.
templates/deployment-myapp.yaml **Reference:** KSV-014	HIGH	Container ‘vault-init‘ should set ‘securityContext.readOnlyRootFilesystem‘ to true.
templates/deployment-myapp.yaml **Reference:** KSV-117	HIGH	Deployment ‘myapp‘ should not set ports lower than 1024.
templates/deployment-vault.yaml **Reference:** KSV-014	HIGH	Container ‘vault‘ should set ‘securityContext.readOnlyRootFilesystem‘ to true.
templates/deployment-vault.yaml **Reference:** KSV-014	HIGH	Container ‘volume-mount-hack‘ should set ‘securityContext.readOnlyRootFilesystem‘ to true.

Table 2: Issues detected by Trivy

Detected issues as shown in Table 2 can be found on the Databases of Trivy and CVE, A link is provided in the reports for reference which highlights the potential issue and suggester fix. Which is first help for developers to fix the issue.

7.3 Recommendations

Trivy scan results also provide recommendations in pipeline logs for quick fixtures. Based on the scan results, the following actions are recommended to mitigate the detected risks:

- Avoid storing secrets in ConfigMaps.
- Set ‘securityContext.readOnlyRootFilesystem‘ to true for all containers.
- Avoid using ports lower than 1024 for container communication.

7.4 References

For further details, refer to the Aqua Security documentation on misconfigurations:

- AVD-KSV-0109: ConfigMap storing secrets
- KSV-014: Immutable root file system
- KSV-117: Privileged ports

8 Conclusion

The experiment demonstrates the successful integration of security scanning into a GitLab CI/CD pipeline using Trivy. By automating the vulnerability scanning process, the pipeline effectively ensures that code submissions are thoroughly evaluated for security risks before they are merged. The admission control mechanism, based on automated decision-making, helps in maintaining a secure codebase by enforcing strict criteria for PR acceptance. Additionally, the custom GitLab Runner enables flexible execution of security scans, making the system scalable and adaptable to different development environments. The approach provides a robust and efficient method for incorporating security into the DevOps pipeline, minimizing manual intervention and reducing the risk of vulnerabilities being introduced into production. Continuous improvements, including regular updates to vulnerability databases and refinement of decision criteria, will further strengthen the pipeline's effectiveness over time.

References

[22a] *License - Trivy*. Github.io, 2022. URL: https://aquasecurity.github.io/trivy/v0.55/docs/scanner/license/ (visited on 09/26/2024).

[22b] *Misconfiguration - Trivy*. Github.io, 2022. URL: https://aquasecurity.github.io/trivy/v0.55/docs/scanner/misconfiguration/ (visited on 09/26/2024).

[22c] *Secret - Trivy*. Github.io, 2022. URL: https://aquasecurity.github.io/trivy/v0.55/docs/scanner/secret/ (visited on 09/26/2024).

[23] *Vulnerability - Trivy*. Github.io, 2023. URL: https://aquasecurity.github.io/trivy/v0.55/docs/scanner/vulnerability/ (visited on 09/26/2024).

[24a] *CVE Details*. https://www.cvedetails.com/. Accessed: 2024-09-28. 2024.

[24b] *Vulnerability Database | Aqua Security*. Aqua Vulnerability Database, 2024. URL: https://avd.aquasec.com/ (visited on 09/26/2024).

[Bha23] Preeti Bhardwaj. *Detecting Container vulnerabilities leveraging the CICD pipeline MSc Research Project Cybersecurity*. Dec. 2023. URL: https://norma.ncirl.ie/6512/1/preetibhardwaj.pdf (visited on 07/05/2024).

[blu21] bluelight. *How to Set up Trivy Scanner in GitLab CI: The Complete Guide*. https://bluelight.co/blog/how-to-set-up-trivy-scanner-in-gitlab-ci-guide. Accessed: 2024-09-28. 2021.

[Cim19] Catalin Cimpanu. "Alpine Linux Docker images ship a root account with no password". In: *ZDNet* (May 2019). Accessed: 2024-09-28. URL: https://www.zdnet.com/article/alpine-linux-docker-images-ship-a-root-account-with-no-password/.

[Com24] Synopsys Software Integrity Customer Community. *Black Duck: Introduction to Scanning*. Accessed: 2024-09-28. 2024. URL: https://community.synopsys.com/s/article/Black-Duck-Introduction-to-Scanning.

[Doc20] Inc Docker. "Docker". In: *lınea].[Junio de 2017]. Disponible en: https://www. docker. com/what-docker* (2020).

[Esc+16] Daniel Escobar et al. "Towards the understanding and evolution of monolithic applications as microservices". In: *2016 XLII Latin American Computing Conference (CLEI)*. 2016, pp. 1–11. DOI: 10.1109/CLEI.2016.7833410.

[Gup21] N. L. Gupta. *Container (Docker) Image Vulnerability Scan Using Anchore*. https://medium.com/linux-shots/container-docker-image-vulnerability-scan-using-anchore-b3a3a36bad9a. Accessed: 2024-09-28. Sept. 2021.

[Kub19] T Kubernetes. "Kubernetes". In: *Kubernetes. Retrieved May* 24 (2019), p. 2019.

[R+24] Rajyashree R et al. "An Empirical Investigation of Docker Sockets for Privilege Escalation and Defensive Strategies". In: *Procedia Computer Science* 233 (2024). 5th International Conference on Innovative Data Communication Technologies and Application (ICIDCA 2024), pp. 660–669. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2024.03.255. URL: https://www.sciencedirect.com/science/article/pii/S187705092400615X.

[Red19] Redhat. *What is Clair?* `https://www.redhat.com/en/topics/containers/what-is-clair`. Accessed: 2024-09-28. Jan. 2019.

[SAD19] Sari Sultan, Imtiaz Ahmad, and Tassos Dimitriou. “Container Security: Issues, Challenges, and the Road Ahead”. In: *IEEE Access* 7 (2019), pp. 52976–52996. DOI: `10.1109/access.2019.2911732`.

[Sax23] Prawal Saxena. “Container Image Security with Trivy and Istio Inter-Service Secure Communication in Kubernetes - NORMA@NCI Library”. In: *Ncirl.ie* (Jan. 2023). DOI: `https://norma.ncirl.ie/6491/1/prawalsaxena.pdf`. URL: `https://norma.ncirl.ie/6491/` (visited on 09/26/2024).

[Tiw23] Himanshu Tiwari. *Enhancing Container Security Through Automated Vulnerability Scanning and Remediation with Trivy.* Insights2Techinfo, Oct. 2023. URL: `https://insights2techinfo.com/enhancing-container-security-through-automated-vulnerability-scanning-and-remediation-with-trivy/`.

[Whi24] W. Whitmore. *Unit 42 Incident Response Retainers Enhance Organizational Resilience.* `https://www.paloaltonetworks.com/blog/2024/09/unit-42-incident-response-retainers-enhance-organizational-resilience/`. Accessed: 2024-09-28. Sept. 2024.

[Zam+21] Fiorella Zampetti et al. “CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study”. In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2021, pp. 471–482. DOI: `10.1109/ICSME52107.2021.00048`.

Abbreviations and Acronyms

PaaS	*Platform as a Service*
CI	*Continuous Integration*
CD	*Continuous Deployment*
K8s	*Kubernetes*
PR	*Pull Request*
CVE	*Common Vulnerabilities and Exposures*
IaC	*Infrastructure as Code*
DevSecOps	*Development, Security, and Operations*
YAML	*Yet Another Markup Language*
GCP	*Google Cloud Platform*
API	*Application Programming Interface*
AWS	*Amazon Web Services*
devOps	*development operations*
OS	*operating system*

List of Figures

Listings

A Appendix 1: Gitlab-ci.yaml Code

```
stages:
  - scan
  - test
  - build
  - deploy

.trivy-scan-template:
  image: aquasec/trivy:latest
  stage: scan
  allow_failure: false
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'

00 scan trivy misconfig:
   extends: .trivy-scan-template
   script:
     - trivy fs --severity HIGH,CRITICAL --exit-code 1 --scanners
        ↪ misconfig ./project

00 scan trivy license:
   extends: .trivy-scan-template
   script:
     - trivy fs --severity HIGH,CRITICAL --exit-code 1 --scanners
        ↪ license ./project

00 scan trivy file system:
   extends: .trivy-scan-template
   script:
     - trivy fs --severity HIGH,CRITICAL --exit-code 1 --scanners
        ↪ vuln ./project

00 scan trivy IAAC:
   extends: .trivy-scan-template
   script:
     - trivy conf --severity HIGH,CRITICAL --exit-code 1 ./project


test:
  stage: test
  script:
    - echo "Hello world!"

build:
  stage: build
  script:
    - echo "Building project..."

deploy:
  stage: deploy
  script:
    - echo "Deploying project..."
```

Listing 1: .gitlab-ci.yaml file

B Appendix 2: Dockerfile

```
FROM alpine:3.18
ENV TRIVY_VERSION=v0.18.3
RUN apk add --no-cache \
    curl \
    && curl -sfL https://raw.githubusercontent.com/aquasecurity
    /trivy/main/contrib/install.sh | sh -s -- -b /usr/local/bin
    ${TRIVY_VERSION} \
    && apk del curl
WORKDIR /app
# Copy scripts if any
# COPY script.sh /app/

ENTRYPOINT ["trivy"]
CMD ["--help"]
```

Listing 2: Dockerfile