International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2019): 7.583

DevSecOps Automation: SAST/DAST Integration in GitLab CI/CD with Semgrep, OWASP ZAP, and Dependency-Check

Sandhya Guduru

Masters in Information Systems Security, Software Engineer - Technical Lead

Abstract: ***As software development accelerates, integrating security into continuous integration and continuous deployment (CI/CD) pipelines is essential. This paper explores the automation of security testing in GitLab CI/CD by embedding Static Application Security Testing (SAST) with Semgrep, Dynamic Application Security Testing (DAST) with OWASP ZAP, and Software Bill of Materials (SBOM) generation with Dependency-Check. These tools enable early vulnerability detection, reducing security risks in production. The implementation of SLSA scorecards is also examined to assess software supply chain security and Kubernetes admission controllers to enforce security policies by blocking vulnerable builds. Automating these security measures can enhance application security without compromising development speed. This paper highlights best practices for securing DevSecOps pipelines effectively.***

Keywords: DevSecOps Automation, CI/CD Security, GitLab CI/CD, SBOM analysis, SAST and DAST, OWASP ZAP, Dependency-Check, Kubernetes admission controllers

1. Introduction

Modern software development increasingly adopts DevSecOps automation practices, integrating security into the development lifecycle rather than treating it as a separate phase. This shift ensures vulnerabilities are identified and mitigated early, reducing risks and improving overall software security. As organizations rely on GitLab CI/CD pipelines to streamline software delivery, embedding CI/CD security automation becomes essential to prevent security flaws from reaching production.

Traditional security assessments, such as manual code reviews and penetration testing, struggle to keep pace with rapid development cycles. SAST and DAST offer automated solutions, but integrating them seamlessly into CI/CD pipelines presents challenges. Ensuring comprehensive security coverage without introducing significant delays requires careful selection and configuration of security tools. Additionally, securing software supply chain security has become a priority, with recent cyber threats targeting dependencies and building processes.

This paper explores the security testing automation in GitLab CI/CD by integrating Semgrep for SAST, OWASP ZAP for DAST, and Dependency-Check for Software Bill of Materials (SBOM) generation. It also examines the implementation of SLSA scorecards to assess software supply chain security and Kubernetes admission controllers to enforce security policies. By embedding these mechanisms into CI/CD security pipelines, this research aims to enhance security while maintaining development efficiency.

2. Literature Review

Research on DevSecOps automation has significantly expanded in recent years, emphasizing the integration of security within software development workflows. Traditional security models often treated security as a separate phase, leading to delayed vulnerability detection and costly remediation efforts. Studies have shown that embedding security into CI/CD pipelines improves software resilience by catching vulnerabilities earlier in the development cycle [1].

This shift has driven the adoption of static application security testing (SAST), dynamic application security testing (DAST), and software bill of materials (SBOM) generation, which provide multi-layered security analysis. SAST tools, such as Semgrep, analyze source code before execution, identifying security flaws without running the application. These tools are valuable for detecting common coding mistakes and compliance violations but often generate false positives, requiring fine-tuned rules [2].

In contrast, DAST tools, such as OWASP ZAP, perform security testing on live applications, simulating real-world attacks to uncover vulnerabilities in authentication, session management, and API endpoints. While DAST scans provide runtime validation, they may struggle with authentication challenges and require properly configured test environments to be effective [3]. The role of SBOM in software security has gained prominence due to increasing concerns about supply chain attacks. Dependency-Check and similar tools help identify vulnerable third-party dependencies by cross-referencing known vulnerability databases, allowing organizations to manage risk exposure proactively [4].

The table below compares these security approaches based on key attributes.

Table 1: Security Tool Comparison for GitLab CI/CD Integration

Security Tool	Type	Strengths	Limitations
Semgrep	SAST	Fast scans, customizable rules, detects vulnerabilities early	High false positive rate, rule complexity
OWASP ZAP	DAST	Real-time attack simulation, API security testing	False negatives, authentication issues
Dependency-Check	SBOM	Identifies vulnerable dependencies, enhances supply chain security	Limited coverage of proprietary libraries

Several studies have explored the integration of security tools within GitLab CI/CD pipelines, demonstrating how automated security checks can enhance software security while maintaining developer productivity [5]. Implementing SAST, DAST, and SBOM analysis at different stages ensures a comprehensive security assessment before deployment. The following code snippets illustrate how these security tools can be embedded within GitLab CI/CD configurations.

A common method for implementing SAST involves integrating Semgrep into the CI/CD pipeline. Semgrep is a lightweight static analysis tool that enables customizable rule-based scanning, making it effective for detecting security vulnerabilities early in the development process. The following GitLab CI/CD configuration shows how Semgrep can be executed as a pipeline job.

```
stages:
  - security_scan

semgrep_scan:
  stage: security_scan
  image: returntocorp/semgrep
  script:
    - semgrep --config=auto --json > semgrep-report.json
  artifacts:
    paths:
      - semgrep-report.json
  only:
    - merge_requests
    - branches
```

Figure 1: A GitLab CI/CD pipeline configuration integrating Semgrep for SAST

While SAST is effective for identifying vulnerabilities in source code, it does not assess how the application behaves at runtime. To complement this, DAST tools like OWASP ZAP perform real-world security testing by attacking the application in a controlled environment. The following pipeline configuration integrates OWASP ZAP to scan an application's API endpoints.

```
zap_scan:
  stage: security_scan
  image: owasp/zap2docker-stable
  script:
    - zap.sh -quickurl http://example.com -quickout zap-report.html
  artifacts:
    paths:
      - zap-report.html
```

Figure 2: OWASP ZAP integration in GitLab CI/CD for DAST-based API security testing

Another critical component of DevSecOps automation is SBOM analysis, which helps track software dependencies and detect vulnerable third-party libraries. Dependency-Check is a widely used tool for generating SBOMs and identifying security flaws within dependencies. The following GitLab CI/CD configuration demonstrates how to incorporate Dependency-Check into a security pipeline.

```
dependency_check:
  stage: security_scan
  image: owasp/dependency-check
  script:
    - dependency-check.sh --project my-app --out dependency-report.html
  artifacts:
    paths:
      - dependency-report.html
```

Figure 3: Dependency-Check implementation in GitLab CI/CD to generate SBOM and detect vulnerabilities in dependencies.

Beyond vulnerability detection, researchers have explored enforcement mechanisms to strengthen CI/CD pipeline security. Studies using empirical vulnerability models, such as those analyzing OpenSSL, demonstrate how software supply chain risks can be identified and predicted -reinforcing the need for structured frameworks like SLSA to evaluate build provenance and ensure artifact integrity [6].

Similarly, Kubernetes admission controllers act as a gatekeeper for production environments, blocking deployments that fail security policies. These controllers enforce security constraints, such as rejecting images with critical vulnerabilities or unapproved dependencies, preventing high-risk applications from being deployed [7]. The table below summarizes the role of these enforcement mechanisms.

Table 2: Security Enforcement Mechanisms in CI/CD Pipelines

Security Measure	Function	Benefit
SLSA Scorecards	Software artifact security assessment	Ensures build integrity and security compliance
Kubernetes Admission Controllers	Blocks deployments with vulnerabilities	Prevents insecure applications from reaching production

Despite advancements in automated security tooling, challenges remain in optimizing these solutions for performance, accuracy, and scalability. False positives in SAST scans, authentication difficulties in DAST, and incomplete vulnerability databases in SBOM analysis introduce operational complexities that organizations must address. Moreover, integrating security enforcement mechanisms such as SLSA scorecards and Kubernetes admission controllers requires fine-grained policies and continuous monitoring. Future research may explore the use of machine learning and AI-driven security analytics to improve scan accuracy and reduce developer friction.

The integration of machine learning (ML) in automated security tooling has been explored in areas like networking security, where ML techniques are used to identify anomalies and attacks. These advancements highlight the potential of AI-driven analytics to enhance the accuracy, performance, and scalability of security solutions in complex environments like DevSecOps pipelines [8].

Ultimately, combining SAST, DAST, SBOM analysis, SLSA scorecards, and Kubernetes admission controllers forms a robust security framework for modern DevSecOps practices, reinforcing the need for security automation within CI/CD pipelines [9].

3. Problem Statement

The adoption of CI/CD pipelines has transformed software development by enabling rapid and continuous deployment. However, security concerns have not kept pace with this acceleration. Traditional software development models incorporate security as a final step, but CI/CD demands a shift-left approach where security is embedded throughout the development lifecycle. Without this integration, organizations face increased risks of deploying vulnerable software, which can be exploited by malicious actors.

A key challenge in modern DevSecOps practices is balancing security and development speed. Security tools must operate efficiently within CI/CD workflows without disrupting developer productivity. Many organizations still rely on outdated security practices that do not scale with automated deployment models, leading to security gaps. Additionally, the complexity of integrating multiple security tools within a pipeline creates operational friction, making security implementation inconsistent across teams.

The following sections outline the major security challenges in CI/CD environments, including risks associated with traditional pipelines, the inefficiencies of manual security testing, difficulties in tool integration, and the need for automated enforcement of security policies.

3.1 Security Risks in Traditional CI/CD Pipelines

Modern CI/CD pipelines streamline software delivery but often lack built-in security measures, exposing applications to critical vulnerabilities. When security is not integrated into the development process, insecure code can be deployed into production, increasing the risk of breaches, data leaks, and system compromise. Threat actors target misconfigured CI/CD environments, exploiting exposed secrets, unpatched dependencies, and weak access controls. Additionally, lack of visibility into security risks across the pipeline makes it difficult to track and remediate vulnerabilities before they become exploitable.

3.2 Limitations of Manual Security Testing

Traditional security testing relies on manual assessments, which are slow, inconsistent, and unable to scale with modern development cycles. As software updates become more frequent, security teams struggle to keep pace, leading to delayed vulnerability detection and patching. Manual code reviews and penetration testing require significant expertise and effort, making them resource-intensive and impractical for fast-moving CI/CD environments. Moreover, human error introduces inconsistencies, resulting in missed vulnerabilities or wrongly prioritized risks. Developers often perceive security reviews as bottlenecks, leading to resistance in adopting security best practices.

3.3 Challenges in Integrating Security Tools Seamlessly

Integrating SAST, DAST, and SBOM tools into CI/CD pipelines introduces technical and operational challenges. SAST tools often generate excessive false positives, overwhelming developers with irrelevant security alerts. DAST scans can be slow and require fully deployed environments, making real-time vulnerability detection difficult. SBOM generation and dependency scanning help track vulnerabilities in third-party components but require accurate dependency resolution to avoid false reports. Without careful tuning and integration, security tools can slow down development and increase friction between security and engineering teams.

3.4 Need for Automated Security Policy Enforcement

Even when security tools are present, enforcing security policies consistently remains a challenge. Developers may bypass security checks if they cause workflow disruptions, leading to unsecure builds being deployed. Inconsistent enforcement of security policies results in non-compliant releases, increasing risk exposure. Additionally, organizations often lack a structured mechanism to block vulnerable artifacts from entering production, relying instead on reactive measures after deployment. Without automated policy enforcement, security remains an afterthought rather than an integral part of the software development lifecycle.

4. Proposed Solutions

To address the security challenges in CI/CD pipelines, this paper proposes a DevSecOps automation framework that integrates SAST, DAST, and SBOM generation within GitLab CI/CD. This approach ensures security is embedded throughout the development lifecycle, reducing risks associated with insecure code, vulnerable dependencies, and misconfigured deployments. Additionally, SLSA scorecards are leveraged to assess the security posture of the software supply chain, while Kubernetes admission controllers enforce security policies before deployment.

4.1 Integration of SAST, DAST, and Dependency Scanning in GitLab CI/CD

A key aspect of this solution is the seamless integration of security tools into GitLab CI/CD pipelines to automate static analysis, dynamic testing, and dependency tracking. Semgrep is an AST-based SAST tool that provides fast and customizable pattern matching for identifying security vulnerabilities in source code. Unlike traditional SAST tools that generate excessive false positives, Semgrep enables rule-based scanning to detect insecure coding patterns with minimal overhead. The following snippet demonstrates how to integrate Semgrep into a GitLab pipeline.

```
stages:
  - security

semgrep_scan:
  stage: security
  image: returntocorp/semgrep
  script:
    - semgrep --config "p/ci"
  allow_failure: true
```

Figure 4: Integrating Semgrep in GitLab CI/CD

This implementation runs Semgrep within a dedicated security stage, ensuring that code is analyzed for vulnerabilities before proceeding to further build or deployment steps.

Another critical component of security integration is OWASP ZAP, a DAST tool that performs API fuzzing and runtime security assessments. The tool scans web applications for vulnerabilities, such as injection flaws and misconfigurations, that might not be detectable through static analysis. The following GitLab pipeline job configures OWASP ZAP to perform automated security scans.

```yaml
zap_scan:
  stage: security
  image: owasp/zap2docker-stable
  script:
    - zap-baseline.py -t https://example.com -r zap_report.html
  artifacts:
    paths:
      - zap_report.html
```

Figure 5: OWASP ZAP DAST Integration in GitLab CI/CD

This configuration ensures that web applications undergo DAST scanning as part of the pipeline, generating vulnerability reports for review.

Software Bill of Materials (SBOM) generation is also critical for tracking third-party dependencies and identifying security flaws in open-source libraries. Dependency-Check, an OWASP tool, is integrated to detect known vulnerabilities in project dependencies based on NVD (National Vulnerability Database) feeds.

```yaml
dependency_scan:
  stage: security
  image: owasp/dependency-check
  script:
    - dependency-check --project "MyProject" --scan /app --format HTML
  artifacts:
    paths:
      - dependency-check-report.html
```

Figure 6: Dependency-Check SBOM Generation in GitLab CI/CD

By implementing Dependency-Check, the pipeline continuously monitors dependency risks and prevents the use of outdated or vulnerable libraries.

4.2 Enhancing Security with SLSA Scorecards

To strengthen software supply chain security, SLSA (Supply Chain Levels for Software Artifacts) scorecards provide an automated way to assess the security posture of software artifacts. Scorecard checks analyze code repositories, verifying security best practices such as branch protection, signed commits, and dependency tracking.

Automating SLSA Scorecard checks within GitLab CI/CD pipelines ensures that security policies are continuously enforced without manual intervention. These checks evaluate critical security metrics such as signed commits, dependency freshness, and branch protection. The following table summarizes key security checks performed by SLSA scorecards.

Table 3: SLSA Scorecard Security Checks

Security Check	Purpose
Signed Commits	Verifies authenticity of commits
Dependency Updates	Ensures third-party libraries are up-to-date
Branch Protection	Prevents unauthorized changes to critical code
Code Review Policies	Enforces multi-party code review before merges

4.3 Kubernetes Admission Controllers for Security Enforcement

Even with SAST, DAST, and dependency scanning, insecure builds may still reach production. Kubernetes admission controllers act as a policy enforcement layer, preventing the deployment of non-compliant or vulnerable applications. Admission controllers evaluate security policies before allowing workloads to be scheduled. Policies can be defined to block deployments based on detected vulnerabilities, missing SBOM metadata, or the presence of secrets in container images.

By integrating admission controllers with GitLab CI/CD, any image that fails security scanning is denied execution. The

following Kubernetes policy, for example, prevents the deployment of containers with critical CVEs.

```yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
spec:
  requiredDropCapabilities:
    - ALL
  forbiddenSysctls:
    - "net.*"
  runAsUser:
    rule: "MustRunAsNonRoot"
  seLinux:
    rule: "RunAsAny"
```

Figure 7: Kubernetes Admission Policy for Security Enforcement

This policy ensures that containers are deployed with restricted privileges, reducing the risk of exploitation in production environments.

By integrating SAST, DAST, SBOM tracking, SLSA scorecards, and Kubernetes admission controls, this framework automates end-to-end security enforcement in GitLab CI/CD pipelines. These measures collectively reduce vulnerabilities, improve compliance, and enhance the overall security of software development processes.

5. Conclusion

This paper explored the integration of SAST, DAST, and SBOM tracking within GitLab CI/CD pipelines to automate security enforcement. The proposed framework leverages Semgrep for static analysis, OWASP ZAP for dynamic security testing, and Dependency-Check for tracking vulnerabilities in third-party dependencies. Additionally, SLSA scorecards assess the security posture of software artifacts, while Kubernetes admission controllers enforce deployment security policies. These combined approaches ensure that security is embedded throughout the development lifecycle, reducing the risks associated with insecure code, outdated dependencies, and misconfigurations.

Automating security enforcement in CI/CD pipelines offers significant benefits, including early vulnerability detection, reduced manual effort, and improved compliance with security best practices. By integrating security tools into the development workflow, organizations can minimize security risks without disrupting development speed. The use of policy-based enforcement mechanisms, such as Kubernetes admission controllers, further strengthens security by preventing the deployment of non-compliant applications.

Future work could focus on enhancing automation through AI-driven vulnerability detection, refining false-positive reduction techniques in SAST tools, and expanding security policies for cloud-native environments. Additionally, integrating threat intelligence feeds into GitLab CI/CD could improve the real-time detection of emerging vulnerabilities. As DevSecOps practices continue to evolve, further research is needed to optimize security automation while maintaining development agility.

References

[1] Schicchi, M., Vallittu, K., Crispo, B., Sainio, P., & Virtanen, S. (Oct, 2020). *Security in DevOps: understanding the most efficient way to integrate security in the agile software development process*
[2] Ahmed, Z., & Francis, S.C. (2019). Integrating Security with DevSecOps: Techniques and Challenges. *2019 International Conference on Digitization (ICD)*, 178-182.
[3] Thulin, P. (2015). Evaluation of the applicability of security testing techniques in continuous integration environments (Dissertation). Retrieved from https://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-113753
[4] Multistakeholder, N. (Dec, 2019). Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM).
[5] Koopman, M. (2019). A framework for detecting and preventing security vulnerabilities in continuous integration/continuous delivery pipelines.
[6] Benthall, S. (2017). Assessing software supply chain risk using public data. 2017 IEEE 28th Annual Software Technology Conference (STC), 1-5.
[7] Islam Shamim, Md. S., Ahamed Bhuiyan, F., & Rahman, A. (September, 2020). Xi commandments of Kubernetes Security: A systematization of knowledge related to Kubernetes Security Practices. 2020 IEEE Secure Development (SecDev), 58–64.
[8] Herrera, J.A., & Camargo, J.E. (2019). A Survey on Machine Learning Applications for Software Defined Network Security. ACNS Workshops.
[9] Rahul, B., Kharvi, P., & Manu, M. (2019). Implementation of DevSecOps using Open-Source tools. International Journal of Advance Research, Ideas and Innovations in Technology, 5, 1050-1051.