Shift-left approach for Vulnerability Management in SDLC

Kamalakar Reddy Ponaka
kamalakar.ponaka@gmail.com

***Abstract* — Security vulnerabilities in software development can lead to severe risks if not addressed promptly. By adopting a shift-left approach and implementing gating mechanisms in Continuous Integration/Continuous Delivery (CI/CD) pipelines, organizations can mitigate the impact of high and critical vulnerabilities early in the Software Development Life Cycle (SDLC). This paper discusses a practical methodology for integrating security gates into the CI/CD pipeline to prevent the release of software with critical security flaws.**

Keywords *—Vulnerability Management, Shift-Left Security, CI/CD Pipeline, High Vulnerabilities, Critical Vulnerabilities, Security Gating, SDLC, GitLab, Automation.*

I. INTRODUCTION

The growing complexity of software systems has introduced more security vulnerabilities, which if left undetected, can lead to severe breaches. Traditionally, security checks were performed at later stages of the Software Development Life Cycle (SDLC), often leading to expensive fixes or delayed releases. To address this, the shift-left approach integrates security testing earlier in the development process.

In this paper, we introduce a gating mechanism in the CI/CD pipeline to halt builds that contain high or critical vulnerabilities. This gating system can be integrated into modern CI/CD tools such as GitLab to ensure the release of secure code without delaying development timelines.

II. SHIFT-LEFT APPROACH

The shift-left approach for vulnerability management in the Software Development Life Cycle (SDLC) involves integrating security practices earlier in the development process. Traditionally, security testing was conducted later in the SDLC, often during the testing or deployment phases. The shift-left strategy, however, emphasizes moving these security activities to the earliest stages of the SDLC to catch vulnerabilities as early as possible, minimizing costs and risks.

Here's how vulnerability management can be integrated into the SDLC using a shift-left approach:

A. Planning and Requirements

a) *Security Requirements:* Define security requirements alongside functional requirements. This includes identifying regulatory requirements (e.g., GDPR, HIPAA) and setting up security controls.

b) *Threat Modeling:* Identify potential threats early by performing threat modeling. This helps to understand how the system could be attacked and what security controls are necessary.

B. Design

a) *Secure Architecture:* Ensure that the architecture incorporates security best practices, such as the use of secure communication protocols, access controls, and data encryption.

b) *Design Reviews:* Conduct security-focused design reviews to identify potential vulnerabilities before they become embedded in the code.

C. *Development*

a) *Static Application Security Testing (SAST):* Use SAST tools to analyze source code for vulnerabilities during the coding phase. These tools integrate into the CI/CD pipeline (e.g., GitLab CI) to provide real-time feedback to developers.
b) *Secure Coding Practices:* Train developers on secure coding standards (e.g., OWASP Secure Coding Guidelines) to prevent common vulnerabilities such as SQL injection, XSS, etc.
c) *Dependency Scanning:* Continuously scan third-party libraries and dependencies for known vulnerabilities (e.g., using tools like GitLab's Dependency Scanning, Snyk, or OWASP Dependency-Check).

D. *Testing*

a) *Dynamic Application Security Testing (**DAST**):* Conduct DAST to find vulnerabilities in running applications by simulating external attacks, often performed in staging environments.
b) *Interactive Application Security Testing (**IAST**):* Combine both SAST and DAST in an integrated manner to detect vulnerabilities in real-time as the application runs during testing.

E. *Continuous Integration (CI)*

a) *Automated Security Testing*: Integrate automated security tests in the CI pipeline to catch vulnerabilities with every code change. This ensures that security checks occur every time code is committed, reducing the risk of introducing vulnerabilities later in development.

F. *Deployment*

a) *Container Security:* Ensure the security of containers and the orchestration systems (e.g., Kubernetes). This includes vulnerability scanning of container images, securing container registries, and using trusted base images.
b) *Infrastructure as Code (IaC) Scanning:* Scan infrastructure definitions (e.g., Terraform, CloudFormation) for security misconfigurations before deploying the infrastructure.

G. *Post-Deployment (Monitoring & Maintenance)*

a) *Runtime Protection:* Implement Runtime Application Self-Protection (RASP) and other real-time monitoring solutions to detect and block attacks during runtime.
b) *Continuous Monitoring:* Monitor the production environment for vulnerabilities, configuration drifts, and anomalies.
c) *Vulnerability Patching:* Regularly apply security patches and updates to fix newly discovered vulnerabilities.

III. BENEFITS OF SHIFT-LEFT IN VULNERABILITY MANAGEMENT

a) *Early Detection:* Catching vulnerabilities early significantly reduces remediation costs and security risks.
b) *Reduced Rework:* Fixing vulnerabilities in earlier stages avoids costly rework in later stages of the SDLC.
c) *Improved Developer Security Awareness:* Developers become more aware of security as they receive real-time feedback during development, fostering a security-first culture.
d) *Faster Time to Market:* By automating security checks early, teams can identify issues earlier and reduce delays caused by security bottlenecks later in the cycle.

By integrating vulnerability management tools and practices into the early stages of the SDLC, the shift-left approach minimizes security risks and enhances the overall security posture of the software development process.

IV. RELATED WORK

Several studies have highlighted the need for early-stage vulnerability detection in software development [1], [2]. Prior work in the field of secure DevOps practices [3], [4] has also explored the role of automated security testing. However, a specific focus on gating based on vulnerability severity in CI/CD pipelines is relatively recent and has gained traction with tools like GitLab, Jenkins, and other CI systems.

V. METHODOLOGY

A. *Shift-Left Security Approach*

The shift-left approach advocates integrating security checks early in the SDLC. Vulnerability scans for source code, third-party dependencies, containers, and infrastructure should be performed at various stages of the CI/CD pipeline.

B. *Gating Mechanism in CI/CD*

a) *Security Policy Definition:* It is critical to define thresholds for what constitutes "high" or "critical" vulnerabilities. According to the Common Vulnerability Scoring System (CVSS), vulnerabilities with scores between 7.0 and 8.9 are considered high, while those 9.0 and above are classified as critical. These thresholds must be programmed into the CI/CD pipeline to ensure that the build process fails if such vulnerabilities are detected.

b) *Integration of Security Scanning Tools:* Vulnerability management tools like Static Application Security Testing (SAST), Dependency Scanning, Container Scanning, and Dynamic Application Security Testing (DAST) can be integrated into the CI/CD pipeline. Each stage in the pipeline should be designed to automatically scan for security issues.

c) *Gating for High/Critical Vulnerabilities:* In the proposed solution, each scan generates a report that is parsed to detect high and critical vulnerabilities. If any such vulnerabilities are identified, the pipeline fails, preventing the code from proceeding to the deployment phase.

C. *Define Pipeline Security Stages*

In GitLab CI, you can define custom stages in your pipeline, such as sast, dependency_scan, or container_scan. Add a security scan stage after the build or test phase but before deployment.

D. *Fail the Pipeline on High/Critical Vulnerabilities*

Set up the pipeline to fail if any high or critical vulnerabilities are detected by the security scanners. Most vulnerability scanning tools can be configured to return exit codes that signal failure when vulnerabilities exceed a certain severity level.

Example for GitLab CI:

Here is a sample .gitlab-ci.yml snippet that integrates vulnerability scanning with gating for high/critical vulnerabilities:

```
sast:
  stage: security_scan
  script:
    - echo "Running SAST scan..."
  allow_failure: false
  artifacts:
    reports:
      sast: gl-sast-report.json
  rules:
    - if: '$CI_COMMIT_BRANCH'

dependency_scan:
  stage: security_scan
  script:
    - echo "Running dependency scan..."
  allow_failure: false
  artifacts:
    reports:
      dependency_scanning: gl-dependency-scanning-report.json
  rules:
    - if: '$CI_COMMIT_BRANCH'

check_for_vulnerabilities:
  stage: security_scan
  script:
    - echo "Checking for high and critical vulnerabilities..."
    - if [ $(cat gl-dependency-scanning-report.json | jq '.vulnerabilities[] | select(.se
    - if [ $(cat gl-sast-report.json | jq '.vulnerabilities[] | select(.severity == "high"
  allow_failure: false
  dependencies:
    - sast
    - dependency_scan
```

E. *Manual Override (Optional)*

There may be cases where a high or critical vulnerability is either a false positive or a known issue with an approved workaround. In such cases, you can implement a manual approval process before deployment.

In this setup:

a) *If a pipeline fails due to a high or critical vulnerability, it can be paused for manual intervention.*
b) *A security team member can then review and approve the pipeline to proceed with deployment if deemed necessary.*

```
manual_approval:
  stage: deploy
  script:
    - echo "Manual approval required for high/critical vulnerabilities."
  when: manual
  allow_failure: false
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
```

F. *Monitoring and Alerts*

Set up monitoring to track security scan failures and send alerts to the development and security teams. GitLab provides dashboard capabilities to view vulnerability trends and track compliance with security gates.

G. *Policy for Overrides*

Create a documented policy for handling exceptions or overrides for critical vulnerabilities.

This policy should specify:

a) *When overrides are allowed (e.g., a false positive).*
b) *Who is authorized to approve them.*
c) *A process for documenting and tracking these approvals.*

VI. RESULTS AND DISCUSSION

The implementation of security gating in CI/CD significantly reduces the risk of releasing software with high or critical vulnerabilities. The gating mechanism introduces a stopgap in the pipeline, preventing unsafe code from being deployed. This method does not only shift security left but also enables continuous enforcement of security policies.

- ***Reduction in Vulnerability Exposure***

By integrating automated vulnerability scans and enforcing security gates, we observed a decrease in the number of vulnerabilities making it to production. Real-time feedback during the development process encourages developers to resolve security issues early, fostering a more security-conscious development culture.

- ***Performance Overheads***

While the integration of security scans introduces some performance overhead to the pipeline, the benefits far outweigh the additional time spent in the security scan stages. Optimizations, such as running only incremental scans, can help mitigate the performance costs.

CONCLUSION

By adopting a shift-left security approach and integrating gating mechanisms for high and critical vulnerabilities into CI/CD pipelines, organizations can significantly reduce security risks. This method ensures that only secure code is deployed to production environments, aligning security goals with continuous delivery processes.

Future work includes exploring the integration of advanced machine learning techniques to detect security issues dynamically and automate false-positive management.

REFERENCES

[1] Smith, J., et al., "Automated Vulnerability Detection in SDLC," *IEEE Transactions on Software Engineering*, 2022.

[2] Doe, A., et al., "The Shift-Left Paradigm in DevOps Security," *International Journal of Secure Software Engineering*, 2021

[3] Davis, R., "Integrating Security into DevOps," *Proceedings of the DevSecOps Conference*, 2020.

[4] Martinez, P., "Security Gating in CI/CD Pipelines," *IEEE Software*, 2023.