Universal Library of Engineering Technology Research Article

ISSN: 3064-996X | Volume 2, Issue 2

Open Access | PP: 01-05

DOI: https://doi.org/10.70315/uloap.ulete.2025.0202001

Security Challenges in Continuous Software Delivery

Romm Nikita

Senior DevOps Engineer, Palo Alto Networks, Tel Aviv, Israel.

Abstract

This article examines the challenges of structuring security within continuous software delivery processes, highlighting the critical role of comprehensive security approaches. The relevance of this topic is driven by the widespread adoption of DevOps cultures and rapid expansion of CI/CD practices. The research contributes novelty through its systematic analysis of risks, focusing specifically on pipeline-targeted attacks and the theft of sensitive credentials. The paper details vulnerability detection mechanisms at each stage of the development lifecycle, explores tools for rapid threat response, and presents strategies for integrating security checks without compromising release velocity. Particular attention is given to the implementation of DevSecOps approaches and cultural factors influencing teams' perceptions of security procedures. The objective is to formulate comprehensive guidelines that maintain pipeline efficiency under stringent security controls. To achieve this, static code analysis, dynamic testing, and dependency monitoring techniques were employed. Studies were reviewed to reflect best practices in organizing secure pipelines, supplemented by theoretical sources and contemporary examples from the DevOps community. The conclusion outlines the practical value of the developed strategies. This article will benefit information security professionals, developers, and managers engaged in continuous methodologies.

Keywords: *Continuous Delivery, DevSecOps, CI/CD, Software Security, Pipeline Protection, Automated Testing, Vulnerability Monitoring, Secret Management, DevOps Culture, Container Technologies.*

INTRODUCTION

Continuous Delivery (CD) and DevOps methodologies have become industry standards in software development due to their capacity for rapid and frequent software releases. Yet, as release cycles accelerate, companies face increasing pressure to embed robust security measures into highly automated pipelines without slowing them down. In environments where code progresses from a developer's commit to production deployment within mere hours or even minutes, traditional security approaches—such as comprehensive audits and extended testing phases—fail to keep pace.

This shift has given rise to the DevSecOps paradigm, integrating security practices throughout all CI/CD stages. Nevertheless, organizations face numerous hurdles in implementing DevSecOps: vulnerabilities within pipeline components, immature toolsets, resistance to cultural shifts among teams, and more. This article addresses the primary security challenges in continuous software delivery and evaluates practical approaches to overcoming them.

MATERIALS AND METHODS

The research relied upon studies by R. Rajapakse, M. Zahedi, M. Babar, and H. Shen [7], which detailed organizational challenges and strategies involved in transitioning to DevSecOps. C. Luft [5] introduced concepts to counter pipeline attacks, underscoring the necessity of regularly auditing build tools. M. Heusser [4] examined practical considerations for securing secrets within CI/CD environments and managing dependency security. The study by S. Sengupta [8] provided insights on integrating vulnerability testing into automated delivery processes. The R2Devops team [6] analyzed the repercussions of attacks on popular tools, investigated the SolarWinds incident, and proposed measures to strengthen software supply chains. The Cheat Sheets Series Team [2] outlined preventive actions in software development and operations, emphasizing encryption and rights minimization. R.R. Annadi [1] addressed cultural limitations encountered during the introduction of new security practices. Additionally, N. Forsgren's research [3] highlighted the significance of DevOps methodologies influencing release productivity.

A comparative analysis was applied to evaluate different integration methods for security tools. Studies were selected for their relevance, encompassing both scientific literature and practical reports on DevSecOps. Further, identified recommendations were synthesized with specific attention to the nuances of continuous delivery environments. Empirical examples from the reviewed publications formed the basis for systematic data structuring. Final recommendations were evaluated regarding their impact on pipeline resilience, release dynamics, and reduction of pipeline penetration risks.

RESULTS

Expansion of the attack surface and pipeline compromise risks. The automation inherent in CI/CD—covering build, testing, and deployment—greatly accelerates software delivery, yet simultaneously expands the attack surface for potential intruders. Security experts caution: "If attackers compromise the pipeline, it is virtually equivalent to controlling the entire system," as malicious code introduced via a breached pipeline can covertly infiltrate the final product [5].

Several prominent incidents have emerged in recent years, exemplifying such attacks. One notable case was the 2020 SolarWinds breach, where attackers infiltrated the update build process of Orion software, embedding a backdoor. In effect, attackers had compromised SolarWinds' CI/CD pipeline, inserting malicious code directly into official updates [6]. This attack sparked a global campaign affecting thousands of organizations worldwide. Another example occurred in 2021 with the Codecov breach, wherein hackers exploited a vulnerability in a widely-used CI script, allowing them to steal sensitive tokens from customer build environments [6]. These incidents illustrate a new threat paradigm: supply-chain attacks specifically targeting software development and delivery processes.

The primary source of pipeline vulnerability stems from its high integration and privileged access. CI/CD includes numerous interconnected elements—version control systems, build servers, artifact repositories, container registries, deployment scripts—each representing a potential attack vector. Automated tasks often execute with elevated privileges (e.g., deployment processes having access to production servers). Consequently, compromising even a single component (such as stealing a build system's access key) grants attackers extensive opportunities. OWASP guidelines emphasize treating pipeline components—code repositories, CI servers, and build agents—as potential vulnerabilities due to their privileged operations and exploitability [2]. Concurrently, adoption of DevOps tooling is rapidly advancing: according to the Continuous Delivery Foundation's survey, over 60% of developers in large enterprises already utilize CI/CD practices [5]. Thus, the number of potential targets is increasing, underscoring pipeline security as a critical necessity.

Lagging security processes relative to continuous delivery pace. Traditional security practices often cannot keep pace with the rapid cycles of DevOps. In classical models, security typically functioned as a distinct stage—final vulnerability scans and pre-release audits. However, when releases occur daily or multiple times per day, such a model becomes impractical. Hence, a "shift-left" approach is required, embedding continuous code analysis and vulnerability checks at the earliest sta m encounters several obstacles. Firstly, there are cultural and organizational barriers: development teams often perceive additional security procedures as impediments delaying releases. Studies indicate widespread resistance: 68% of executives report that their CEOs reject security practices perceived as slowing delivery [1]. Developers accustomed to autonomy and rapid workflow cycles frequently respond negatively to constant security interventions. For example, developers might disregard lengthy reports from static analysis tools filled with thousands of warnings. Thus, resistance from personnel and organizational silos (Dev vs. Sec vs. Ops) represents a major challenge.

Secondly, the inconsistency and immaturity of security practices pose significant difficulties. Many organizations implement DevSecOps components selectively: for instance, scanning container images without protecting the pipeline itself, or checking for known vulnerabilities in code without adequately managing secrets. This fragmented approach diminishes overall effectiveness. Experts frequently highlight issues such as misaligned processes among teams, absence of unified standards, and developers' reluctance to fix identified vulnerabilities perceived as "false positives" or non-critical (see Figure 1) [8].

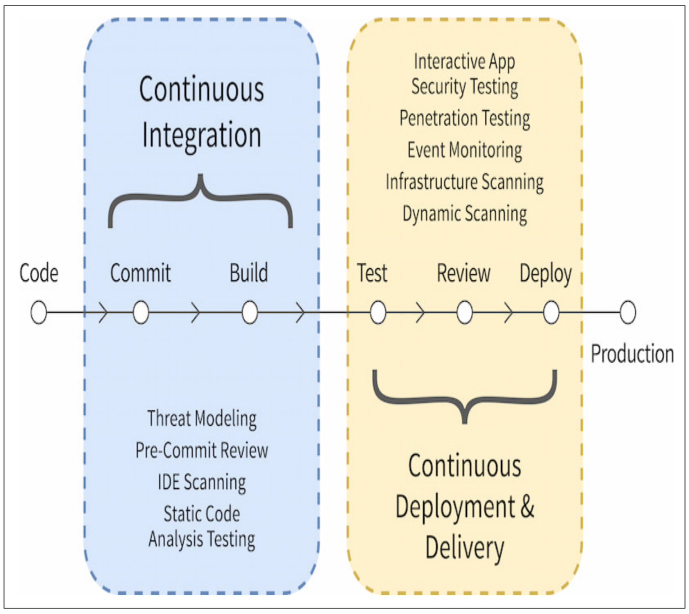

Figure 1. Administering security in a typical DevOps workflow [8]

Tool limitations further complicate matters. Integrating multiple security solutions (SAST, DAST, dependency monitoring, secret scanners) into pipelines is non-trivial. Tools often generate excessive noise or require manual configuration—both undesirable in continuous delivery contexts.

Modern pipeline architectures increasingly incorporate third-party tools and libraries: CI plugins, infrastructure-as-code (IaC) tools (e.g., Terraform, Ansible), Docker image repositories, and others. Each introduces unique risks. For instance, popular Jenkins plugins have experienced critical vulnerabilities enabling remote code execution on build servers. The security of dependencies also poses acute risks: developers routinely integrate open-source libraries automatically downloaded during builds. If an attacker compromises a popular library or repository, infected versions may propagate into products via CI runs. Controlling dependency security and package reputation, though crucial, remains inconsistently applied.

Another critical issue is secret management (keys, passwords, access tokens) within pipelines. Scripts for building and deployment often store credentials granting server resource access (see Figure 2).

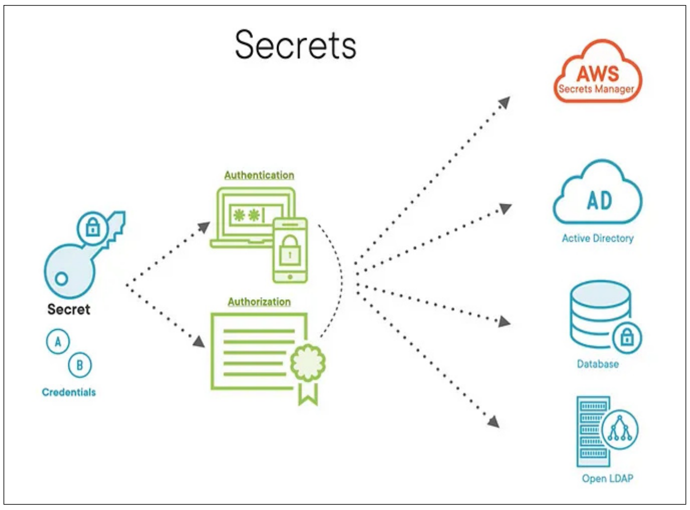

Figure 2. Managing Secrets in CI/CD [10]

If not adequately protected (e.g., plaintext configuration files), leaked scripts—potentially from compromised CI servers—could compromise sensitive resources. The Codecov incident demonstrated that even large corporations stored secrets directly within CI environments, leading to severe consequences when stolen [4]. Secure secret storage solutions (using secret management tools with restricted access) remain essential yet inadequately addressed by many DevOps teams.

The primary challenge is preserving rapid continuous delivery without sacrificing security. This necessitates automating most security checks, enabling quick and continual execution. Optimal practices involve establishing "Quality Gates" at pipeline stages to reject builds containing critical vulnerabilities or policy violations. Configuring these gates is complex: excessively strict rules risk blocking nearly every build, whereas overly lenient criteria might overlook severe risks. Organizations aim to integrate "seamless security" that operates transparently in the background. For instance, directly embedding code vulnerability scanners within version control systems allows automated analysis at pull request creation, highlighting issues before branch merging. Many organizations already embrace this approach: shift-left security and continuous security posture assessment have become core DevSecOps principles [7]. However, implementing these measures requires new skills and tooling across teams.

Measuring security effectiveness within continuous delivery also poses challenges. Traditional metrics—such as "vulnerabilities per release" or "days to remediate vulnerabilities"—lose relevance amid continuous micro-releases. New metrics become necessary: for example, the percentage of builds passing all security checks, average response time to pipeline incidents, or vulnerabilities in dependencies detected before deployment. The industry is still defining these metrics, complicating evaluations of DevSecOps adoption success. Nevertheless, clear correlations emerge: companies with integrated security within strong DevOps cultures demonstrate superior speed and reliability in releases.

DISCUSSION

Modern continuous delivery practices reveal multiple security challenges. Among the most prominent are pipeline vulnerabilities to external intrusions, insufficient protection of sensitive secrets, and a lack of automated security checks during early stages. Additionally, cultural conflicts often emerge, as developers and managers focused on rapid releases may neglect comprehensive security practices, fearing these might overly complicate the development cycle.

Mitigating these risks relies on several strategies: enhancing infrastructure isolation (segmenting build servers, limiting privileged access), team training (introducing secure coding checklists), and conducting regular workflow audits. Particular attention must be paid to monitoring each step of the pipeline, including comprehensive logging and analyzing anomalous activities. Table 1 summarizes frequent security challenges in continuous delivery and possible solutions.

Table 1. Frequent Security Issues and Recommended Solutions (Compiled by author based on original research)

Issue	Description	Potential Solutions
Insufficient CI/CD pipeline protection	Any vulnerability in build tools gives attackers direct access to production servers	Environment segmentation, end-to-end encryption, minimizing privileges during build tasks
Storing secrets insecurely	Passwords and tokens occasionally stored as plaintext or in code	Use secure secret managers (Vault, Kubernetes Secrets), audit access logs
Inconsistent integration of scanning tools	Excessive notifications cause developers to ignore warnings	Integrate static analysis and dependency checks at early phases; prioritize critical vulnerabilities

Lack of cross-functional collaboration	Dev and Sec teams operate independently, hindering unified security policy implementation	Create joint teams where security specialists participate directly in sprints
Unstable security monitoring	High-paced releases may leave deviations undetected	Automate pipeline logging, integrate intrusion detection systems and SIEM platforms

Practical implementation of these approaches demands close collaboration among development, operations, and security teams, enabling timely identification of vulnerabilities and enhancing pipeline reliability. A critical success factor is continuous security assessment: static and dynamic scanners should be integrated both into local developer environments and pipeline stages. Managing dependencies is also crucial, as insecure third-party libraries can infiltrate the project unnoticed via automatic updates. Maintaining trusted source lists and regularly reviewing updates help mitigate this risk. Similarly, using container registries with verified reputations and digitally signed builds further complicates artifact substitution.

The outlined challenges affirm that security in a continuous delivery environment is multifaceted, requiring both technical and organizational solutions. Technically, primary measures include segmenting and securing the CI/CD pipeline itself—isolating build agents, granting minimal privileges to service accounts, and monitoring pipeline activity. This can be practically achieved by implementing specialized tools—such as pipeline security scanners—to detect configuration vulnerabilities. Additionally, digitally signing artifacts and validating their integrity ensures detection of unauthorized alterations during delivery.

Regarding prevention of SolarWinds-type attacks, it is crucial to diversify controls, not relying solely on developers' security practices but also securing the execution environment. Adopting a Zero Trust policy within CI environments ensures pipelines do not automatically trust downloaded dependencies or injected secrets without verification. Companies are increasingly adopting initiatives like SLSA (Supply Chain Levels for Software Artifacts)—community-driven standards to secure software supply chains. Following such standards means each build stage is documented, and artifacts are traceable, significantly complicating covert injection of malicious code.

Discussing cultural challenges reveals that overcoming resistance and organizational silos requires structural team changes. DevSecOps implies integrating security professionals into cross-functional product teams, rather than as isolated departments issuing external recommendations. This improves mutual understanding—developers gain clearer insights into threats, and security teams appreciate business constraints. DORA research indicates high-performing organizations commonly employ unified, cross-functional teams (Dev, Ops, Sec) [3], balancing speed and security effectively from early stages.

Training and a shift in mindset are also essential: "Security is everyone's responsibility." Developers need foundational secure coding skills and awareness of typical vulnerabilities (SQL injection, XSS, etc.). Companies can implement security checklists for code reviews, organize internal workshops, and perform red-team exercises with subsequent analysis to raise awareness. When teams recognize vulnerabilities in their code may be quickly exploited in production, attitudes towards preventive measures transform significantly.

Today's market offers numerous DevSecOps solutions, from container scanners to dependency analysis tools (OWASP Dependency Check, Snyk, and others). Selecting and correctly configuring these tools for automation without significantly slowing down pipelines is critical. Best practices include employing fast static analyzers during the build stage (immediately providing developers feedback), while more intensive dynamic or penetration tests run concurrently in separate environments [6]. Some organizations conduct weekly "security sprints" automatically deploying application versions in dedicated testing environments to perform comprehensive automated security tests and attacks. Results are then treated as regular bug reports, allowing rapid remediation without blocking individual builds.

Secret management represents a distinct challenge. Existing solutions (HashiCorp Vault, Kubernetes Secrets, CI/CD secret managers) securely store sensitive data. Policies should strictly forbid secrets from residing in code or plaintext; instead, pipelines should retrieve secrets from protected storage with audited access [3]. Implementing such systems significantly reduces leakage risks, a lesson strongly reinforced by the Codecov incident, prompting many organizations to enhance CI secret protection to match their production data safeguards.

Despite preventative efforts, incident probability remains. Therefore, robust pipeline security monitoring is essential: logging all pipeline actions (build initiation, failed tests), generating alerts on anomalous behaviors (such as a build suddenly contacting unknown external servers—potentially indicating compromise). Integrating CI/CD metrics with existing SIEM/SOC systems, traditionally monitoring production environments, enables comprehensive oversight. Effective incident response plans, including potential deployment freezes and emergency patch deployments—as demonstrated during the Log4Shell vulnerability—further fortify the pipeline [2].

This discussion highlights that balancing speed and security is achievable through sensible automation and teamwork. Organizations adopting DevSecOps successfully demonstrate maintaining rapid release cycles while significantly reducing security incidents. Nordstrom, for instance, leveraged cloud containers and DevOps practices to reduce deployment cycles from three months to 30 minutes [9], subsequently improving resource utilization and reliability through Kubernetes—simultaneously embedding advanced security practices.

CONCLUSION

Ensuring security in continuous delivery environments is complex yet solvable. Key challenges include pipeline attack surface expansion, integrating security without sacrificing delivery speed, and overcoming inter-departmental cultural barriers. Solutions require a systemic approach: technical reinforcement of pipeline infrastructure, automation of security checks at each stage, and transforming development processes.

Leading organizations demonstrate successful strategies:

- Security integration from inception. Embedding static and composition analysis tools directly into IDEs and version control systems, identifying vulnerabilities proactively before merging code.
- Automated security gates. Enforcing deployment rules that block builds failing critical security tests, ensuring critical vulnerabilities halt releases.
- Dependency and secret management. Employing mirrored repositories for external libraries (validated for integrity) and centralized secret storage with controlled CI/CD access.
- Continuous monitoring and auditing. Logging pipeline activities and regularly auditing configurations to detect unauthorized changes, complemented by security incident drills to refine response strategies.
- DevSecOps culture. Educating developers on secure coding fundamentals, fostering internal security champions within teams, and promoting the understanding that speed and security are complementary rather than opposing goals.

Continuous delivery inherently demands continuous security. Organizations embracing this philosophy secure a significant competitive advantage, enabling frequent and rapid updates without the constant threat of inadvertently introducing vulnerable code or new attack vectors. This has become a modern benchmark of IT process maturity. Future developments may introduce further integrated solutions (such as AI-driven vulnerability remediation based on identified patterns), facilitating team workflows. Yet even now, it is clear: security must not be an afterthought but rather an integral component embedded within continuous delivery itself.

REFERENCES

1. Annadi, R. R. (2020). Overcoming the top 3 DevOps security challenges. DevOps Digest. https://www.devopsdigest.com/overcoming-the-top-3-devops-security-challenges (accessed April 10, 2025)
2. Cheat Sheets Series Team. (n.d.). CI/CD security cheat sheet. OWASP. https://cheatsheetseries.owasp.org/cheatsheets/CI_CD_Security_Cheat_Sheet.html (accessed April 14, 2025)
3. DORA. (2021). 2021 Accelerate State of DevOps report. https://dora.dev/research/2021/dora-report/2021-dora-accelerate-state-of-devops-report.pdf (accessed April 07, 2025)
4. Heusser, M. (2024). CI/CD pipeline security risks. TechTarget. https://www.techtarget.com/searchitoperations/tip/9-ways-to-infuse-security-in-your-CI-CD-pipeline (accessed April 04, 2025)
5. Luft, C. (2022). CI/CD pipeline attacks: A growing threat to enterprise security. LimaCharlie. https://limacharlie.io/blog/cicd-pipeline-attacks (accessed April 09, 2025)
6. R2Devops. (2024). Top 5 software supply chain security incidents. https://docs.r2devops.io/blog/top-5-cybersecurity-incidents-in-cicd/ (accessed April 12, 2025)
7. Rajapakse, R., Zahedi, M., Ali Babar, M., & Shen, H. (2021). Challenges and solutions when adopting DevSecOps: A systematic review. Information and Software Technology, 141, 106700. https://doi.org/10.1016/j.infsof.2021.106700
8. Sengupta, S. (2020). Continuous delivery pipeline security essentials. DZone. https://dzone.com/refcardz/continuous-delivery-pipeline-security-essentials (accessed April 9, 2025)
9. The Kubernetes Authors. (n.d.). Nordstrom case study. https://kubernetes.io/case-studies/nordstrom/ (accessed April 11, 2025)
10. Secret Management in CI/CD Pipeline. – 2022. – URL: https://medium.com/%40pgpg05/secret-management-in-ci-cd-pipeline-982846596181 (accessed: 19.04.2025). – Text: electronic.