I.J. Wireless and Microwave Technologies, 2026, 4, 291-299
Published Online on August 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijwmt.2026.04.16

Securing CI/CD Pipelines: A DevSecOps Framework for Preventing Credential Leaks and Misconfigurations

Akzhibek Amirova
Astana IT University, Astana 010000, Kazakhstan
E-mail: akzhibek.amirova@astanait.edu.kz
ORCID iD: https://orcid.org/0000-0002-5715-4954

Received: February 25, 2026; Revised: March 25, 2026; Accepted: July 15, 2026; Published: August 8, 2026

Abstract: Continuous Integration and Continuous Deployment (CI/CD) pipelines have become fundamental to modern software engineering, enabling rapid and reliable delivery of applications. However, their automation introduces critical vulnerabilities, particularly credential leaks and misconfigurations, which undermine the security of development and deployment environments. This study investigates security risks in Docker-based GitHub Actions workflows and proposes a tailored, DevSecOps-aligned security checklist to mitigate these threats. A systematic literature review was combined with hands-on experiments, in which controlled credential exposures and workflow misconfigurations were deliberately introduced and analyzed. Security controls such as secret scanning with GitGuardian and TruffleHog, configuration validation with GHAST, and access control enforcement were tested in a CI/CD testbed. The findings demonstrate that these integrated methods significantly reduce the risk of credential leakage and pipeline hijacking, while maintaining minimal performance overhead. The novelty of this work lies in consolidating fragmented best practices into a workflow-specific model that is immediately applicable to real-world projects. This contributes actionable guidance for secure-by-design CI/CD pipelines, offering practical protection against supply-chain threats while preserving delivery speed and scalability.

Index Terms: CI/CD security, DevSecOps, GitHub Actions, Docker, credential leaks, misconfigurations, pipeline hardening

1. Introduction

Continuous Integration and Continuous Deployment (CI/CD) pipelines have transformed modern software development by automating testing, building, and delivery processes, thereby accelerating release cycles and improving software reliability. Platforms such as GitHub Actions, integrated with Docker-based environments, are increasingly favored due to their simplicity, scalability, and rich ecosystem of reusable actions. However, this acceleration comes at a cost: CI/CD pipelines expand the attack surface and expose new avenues for exploitation, including credential leaks, misconfigurations, and weak access controls [1,2].

The urgency of securing these environments is underscored by several high-profile incidents. The 2021 Codecov breach led to the exfiltration of CI/CD secrets from thousands of pipelines, while the SolarWinds supply-chain attack demonstrated how compromised workflows can propagate malicious code to thousands of downstream systems [3, 4]. According to the OWASP CI/CD Security Risk List and related reports, insecure credential handling and insufficient configuration management remain among the most common and impactful risks in automated pipelines [5]. These events highlight both the necessity and the challenge of embedding robust security measures directly into the CI/CD lifecycle.

Existing solutions, such as static analysis, secret scanners, and access controls, provide partial protection but often operate in isolation. Their fragmented nature forces DevOps teams to manually integrate multiple tools, creating gaps that can be exploited by attackers. Moreover, developers frequently prioritize speed over security, bypassing scans or deploying with insecure defaults, which increases the likelihood of credential exposure and misconfiguration [6, 7]. This tension between delivery speed and security resilience has spurred interest in DevSecOps approaches that “shift security left” by embedding safeguards at every stage of development and deployment [8, 9].

The present study addresses this gap by consolidating established best practices into a concise, workflow-specific checklist for Docker-based GitHub Actions pipelines. The research combines systematic literature analysis with

controlled experiments in a testbed environment, where dummy secrets and intentional misconfigurations were introduced to validate the effectiveness of proposed security methods. Unlike generic guidelines, the checklist presented here is tailored to the unique risks of GitHub Actions, balancing protection efficacy with minimal performance overhead.

Despite the growing adoption of DevSecOps practices, existing security tools for CI/CD pipelines remain fragmented and are often applied in isolation without a unified operational framework. This fragmentation leads to inconsistent enforcement of security policies, increased risk of credential exposure, and undetected workflow misconfigurations.

Therefore, the main research question of this study is: Can an integrated DevSecOps framework, tailored for GitHub Actions and containerized environments, improve the detection and prevention of credential leaks and workflow misconfigurations while maintaining acceptable performance overhead? To address this question, this paper proposes a structured security framework that integrates multiple tools and validation mechanisms into a unified CI/CD workflow.

So, the main contribution of this work is twofold: first, to provide a structured model for mitigating credential leaks and misconfiguration vulnerabilities in CI/CD environments; and second, to demonstrate through practical validation that security can be strengthened without undermining automation efficiency. By offering actionable recommendations grounded in both academic research and empirical testing, this study contributes to the broader effort of securing the software supply chain in DevOps ecosystems.

2. Background

The widespread adoption of Continuous Integration and Continuous Deployment pipelines has improved automation and reliability in software delivery, but it has also created new attack surfaces for adversaries. Misconfigurations, hard-coded secrets, and insecure token management are consistently highlighted as the most pressing risks in DevOps environments. Notable breaches such as the SolarWinds supply chain attack and the Codecov incident demonstrate how weaknesses in pipelines can compromise thousands of downstream systems.

2.1. Secret Hygiene

Research on CI/CD security frequently identifies secret hygiene as a primary challenge. Meli et al. showed that hard-coded secrets in Git repositories remain pervasive despite widespread awareness [10]. Pan et al. (2024) further describe CI/CD pipelines as “ambush surfaces,” where exposed build logs and poorly scoped tokens create entry points for attackers [11]. Tools such as TruffleHog, GitGuardian, and Snyk have been deployed to automate secret scanning, while OWASP highlights pre-commit and pre-receive hooks as effective preventive measures [12]. However, limitations remain false positives frustrate developers, enforcement is inconsistent across organizations, and some teams bypass controls for the sake of speed. Secret hygiene is a critical aspect of securing Continuous Integration and Continuous Delivery/Deployment (CI/CD) pipelines, particularly to prevent credential leaks that can lead to severe security breaches. Insufficient credential hygiene is identified as a top risk in CI/CD environments [13]. Common sources of credential leaks include hard-coded secrets in source code, exposed secrets in build logs, vulnerabilities in third-party dependencies, stolen CI/CD tokens, and metadata exposure in source code management systems [14, 15]. To mitigate these risks, several strategies are recommended. The shift-left security approach advocates integrating security early in the development process to identify vulnerabilities before they reach production [16]. Git hooks, such as pre-commit and pre-receive hooks, can detect secrets before they are committed or pushed to repositories [17]. Static Application Security Testing (SAST) tools, including SonarQube, Snyk, and TruffleHog, are effective for scanning code and Docker images for vulnerabilities and secrets [18]. Proper secret management practices, such as utilizing GitHub’s native secrets management system and applying Role-Based Access Control (RBAC), are essential for secure credential handling [19, 20]. For GitHub Actions and Docker pipelines, specific recommendations include pinning actions to specific commit SHAs to prevent supply chain attacks, using configuration validation tools like GHAST (GitHub Actions Static Analysis Tool), securing runners by de-privileging accounts, and implementing policy-as-code to enforce security rules [21]. High-profile incidents, such as the Codecov breach and the SolarWinds attack, illustrate the severe consequences of poor secret hygiene, including data breaches and supply chain compromises [15]. However, challenges such as false positives from security tools and potential performance overhead in pipelines must be addressed to ensure effective implementation [22].

2.2. Configuration Validation

Configuration validation is essential for preventing misconfigurations that can compromise CI/CD pipelines, particularly in GitHub Actions and Docker environments. Vulnerabilities may arise from insecure inputs, improper runtime settings, and unsafe workflow outputs, leading to risks such as unauthorized access, code injection, and pipeline hijacking [17].

Validation techniques include static analysis tools such as GHAST, which detect insecure workflow definitions, as well as IDE-based and pre-commit validation mechanisms that identify malformed configurations early in the

development process [22, 23]. Additional safeguards involve enforcing policy-as-code, restricting workflow triggers, pinning actions to specific commit SHAs, and regularly updating dependencies.

While these techniques are effective in identifying configuration-level vulnerabilities, they are often applied independently and lack integration with other security controls, limiting their effectiveness in complex CI/CD environments.

2.3. Access Control

Access control mechanisms have been recognized as a fundamental defense in CI/CD pipelines. Research shows that over-privileged tokens and misconfigured access rights are frequently exploited for lateral movement. Enforcing the principle of least privilege through Role-Based Access Control (RBAC) and ephemeral credentials have been proven effective in limiting attack impact. Yet, adoption of these practices remains inconsistent, and organizations often neglect token scoping, granting broader access than necessary.

2.4. Workflow Security Testing

Workflow security testing plays a critical role in identifying vulnerabilities within CI/CD pipelines. It encompasses static analysis, secrets detection, and runtime validation to ensure the integrity of workflows, particularly in GitHub Actions and Docker-based environments [22, 23].

Tools such as GHAST, SonarQube, and Snyk enable detection of misconfigurations and insecure workflow logic, while solutions like GitGuardian and TruffleHog identify credential leaks across source code and container images [24, 25]. Additional mechanisms, including pre-commit hooks and runtime profiling, help enforce secure configurations and detect unauthorized changes.

Despite their effectiveness, these approaches often suffer from false positives and limited coverage of runtime behaviors, requiring human validation and further refinement.

2.5. Docker Security in CI/CD Pipelines

Docker is integral to CI/CD pipelines, providing isolated environments for building, test, and deployment tasks, but it introduces significant security challenges. Threats include backdoor injections, private code exposure, and credential leaks, particularly through hard-coded secrets in Dockerfile or exposed secrets in container logs [26]. Misconfigurations, such as running containers with root privileges or using outdated images, further exacerbate vulnerabilities. Mitigation strategies include pinning Docker actions to specific commit SHAs and using verified creators to reduce supply chain risks [27]. De-privileging runners by running Docker containers with non-root accounts and regularly updating images and configurations are critical for security. Environment profiling against secure baselines helps detect unauthorized changes in Docker environments. The shift-left security approach recommends scanning Dockerfile and images for vulnerabilities during development using tools like Snyk and SonarQube [28]. Practical experiments validate the effectiveness of tools like GHAST and TruffleHog in detecting Docker-related security issues, such as credential leaks and misconfigurations. However, the literature notes a gap in comprehensive Docker-specific security guidance, suggesting a need for further research to address these challenges in GitHub Actions pipelines.

2.6. Gaps in the literature

Despite these advancements, most existing approaches remain fragmented. Tools and practices are usually studied in isolation, without integration into cohesive frameworks tailored to specific platforms like GitHub Actions. Practitioners often assemble ad hoc solutions that lack empirical validation. Recent surveys highlight this gap and emphasize the need for workflow-specific, reproducible models that consolidate best practices into actionable checklists.

To address this, the present work synthesizes secret scanning, configuration validation, access control, and continuous monitoring into an integrated DevSecOps-aligned checklist for GitHub Actions and Docker pipelines. Unlike prior work, the proposed approach is validated through controlled experiments with deliberately injected secrets and misconfigurations, demonstrating its practical applicability while maintaining minimal performance overhead.

Figure 1 provides a conceptual mapping between the most common CI/CD threats and their mitigation strategies. As illustrated, credential leaks are addressed through multi-layer secret scanning and hooks, misconfigurations are mitigated by GHAST-based static analysis, privilege escalation via tokens is controlled by enforcing role-based access control and scoped credentials, while monitoring gaps are closed through lightweight real-time alerting mechanisms. This mapping highlights how the proposed framework systematically covers the primary attack vectors in CI/CD environments.

As summarized in Table 1, existing CI/CD security mechanisms address distinct categories of risks, including secret hygiene, configuration validation, access control, container security, and monitoring. Tools such as GitGuardian, TruffleHog, and Snyk provide strong coverage of credential leakage and container vulnerabilities, while GHAST enables automated detection of workflow misconfigurations. However, each solution has notable limitations, such as false positives, performance overhead, or limited platform support, which prevent them from functioning as comprehensive frameworks. The comparison highlights that current practices are often applied in isolation,

underscoring the need for an integrated approach like the one proposed in this study.

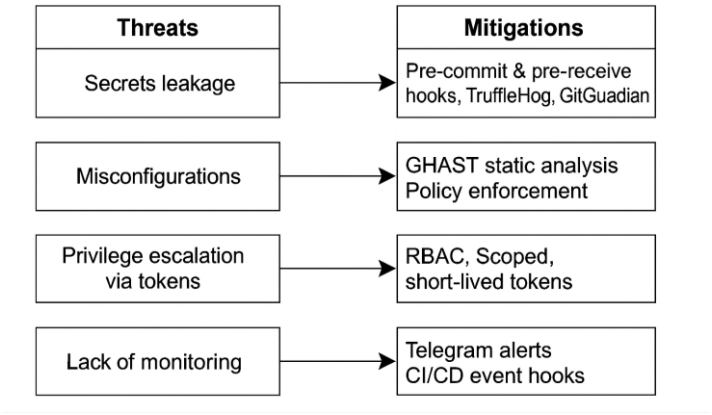

Fig. 1. Mapping of common CI/CD threats to mitigation strategies

Table 1. Comparison of Security Tools and Methods for CI/CD Pipelines

Category	Tool / Method	Strengths	Limitations	Sources
Secret Hygiene	GitGuardian	High accuracy in detecting API keys, tokens; integrates with Git hooks	Subscription-based enterprise features; some false positives	[3], [4]
	TruffleHog	Open-source; regex + entropy scanning; widely used	Can be bypassed; requires tuning	[2], [3]
	Snyk / SonarQube	Covers secrets, vulnerabilities, and dependencies in Docker images	Slower on large projects; resource-intensive	[7], [10]
Configuration Validation	GHAST	Detects unpinned actions, insecure runners, inline scripts	Relatively new; requires manual triage	[5], [6]
	Manual Reviews	Leverage developer knowledge	Error-prone, inconsistent under time pressure	[6]
Access Control	RBAC (GitHub)	Enforces least privilege; native GitHub support	Underused in practice; depends on governance	[8], [9]
	Ephemeral Tokens	Prevent token reuse; reduce lateral movement	Limited platform support	[9]
Container Security	Docker Scanning (Snyk, Clair)	Detects outdated images and embedded secrets	Adds runtime overhead; incomplete coverage	[10], [11]
Monitoring / Alerts	Custom Telegram Bot (this study)	Lightweight real-time alerts for secrets, misconfigurations, failures	Not enterprise-scale; relies on third-party messaging platform	Current work

Although prior studies provide valuable insights into individual aspects of CI/CD security, such as secret scanning, dependency analysis, and container vulnerability detection, they exhibit several limitations. First, most works focus on isolated tools rather than integrated workflows, which limits their practical applicability in real-world DevOps environments. Second, comparative evaluations across tools are often absent, making it difficult to assess their relative effectiveness. Third, few studies address reproducibility and workflow-level validation.

As a result, there remains a clear research gap in developing and experimentally validating an integrated DevSecOps framework that combines multiple security mechanisms into a cohesive and reproducible pipeline configuration.

3. Materials and Methods

This study adopts a controlled experimental case-study design that combines qualitative analysis of security workflows with quantitative evaluation of detection outcomes. While the framework design and workflow integration are analyzed qualitatively, the effectiveness of the approach is assessed using measurable indicators such as detection rate and consistency across multiple experimental runs. The study is exploratory, aiming to bridge the gaps in current CI/CD security models by developing a structured security framework for DevOps workflows.

To validate the proposed security framework, we implemented a controlled CI/CD pipeline simulating a modern DevSecOps workflow. The central codebase was maintained in a GitHub repository, with GitHub Actions serving as the automation platform. Workflow executions were conducted on self-hosted runners deployed within Docker

containers on an Ubuntu 22.04 LTS host system.

The experimental setup included: Docker v24.0.7 for containerized builds and deployments, Python v3.10 for hook scripts and monitoring utilities, Git v2.44 for repository operations.

The baseline pipeline consisted of three stages — build, test, and deploy — without added security controls, serving as a reference for comparison with secured configurations.

To assess secret hygiene, two scanning tools were integrated: GitGuardian CLI (v1.35.0) and TruffleHog (v3.42.0). These were deployed at two levels:

Pre-commit hooks (developer side): Python-based scripts scanned staged files priorto commit. Any exposed secrets triggered commit rejection.

Figure 2 illustrates the overall architecture of the secure CI/CD pipeline used in our experiments. The architecture integrates GitHub Actions workflows with Docker-based runners, secret scanning hooks, GHAST configuration validation, scoped token management, and Telegram-based alerts, forming a comprehensive DevSecOps-aligned environment.

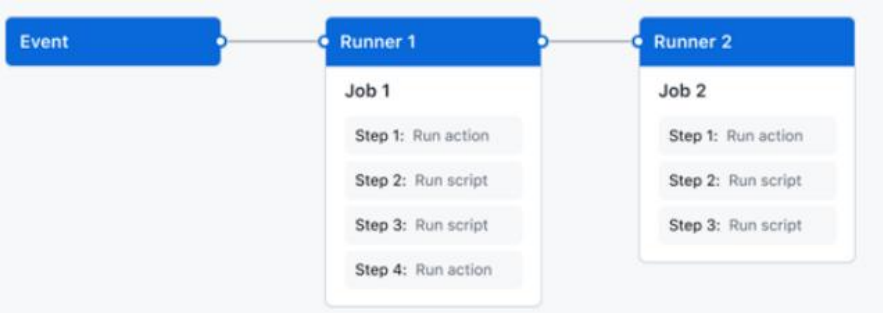

Fig. 2. Overall architecture of the secure CI/CD pipeline

Pre-receive hooks (repository side): enforced at the server level to block insecure commits from entering the main branch.

Dummy secrets were deliberately injected to measure detection accuracy. Both tools successfully intercepted exposed credentials, confirming their suitability for continuous monitoring in CI/CD pipelines.

Misconfiguration detection was conducted using the GitHub Actions Static Analysis Tool. The tool was integrated into workflows to automatically scan YAML configuration files and detect unpinned third-party actions, unsafe inline scripts, insecure self-hosted runner definitions, missing integrity and security validations.

To prevent insecure workflows from being deployed, Figure 3 depicts the process of static analysis and configuration validation using GHAST. Workflow YAML files are analyzed automatically, with findings categorized by severity, and policy enforcement ensures that insecure configurations trigger warnings or failures before deployment.

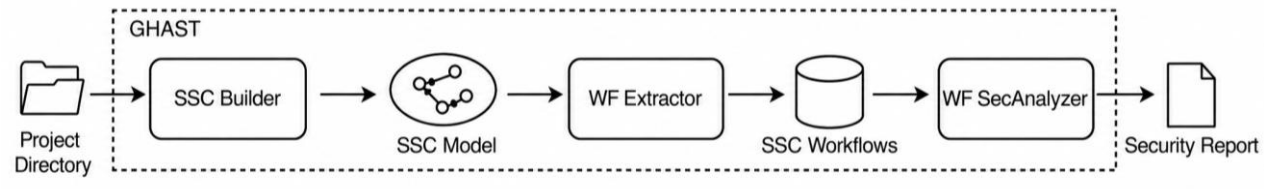

Fig. 3. Workflow of static analysis and configuration validation using GHAST

All identified issues were logged as artifacts. Unsafe configurations triggered FAIL or WARN statuses, requiring remediation before deployment. This method provided consistent coverage, mitigating risks commonly overlooked in manual reviews.

To ensure reproducibility, the experimental setup was carefully structured. Synthetic credentials were injected into the repository in controlled formats, including API keys, tokens, and environment variables embedded within configuration files. These dummy secrets followed realistic patterns commonly detected by secret scanning tools.

Workflow misconfigurations were introduced based on documented CI/CD security risks, including improper permission settings, missing environment isolation, and insecure container build configurations.

Each experimental run followed a predefined sequence: repository initialization, injection of controlled vulnerabilities, execution of the CI/CD pipeline, and logging of detection outcomes across all integrated tools. The workflow configurations used in the experiments are provided to enable independent replication.

4. Results

The proposed security framework was validated through a series of controlled experiments that simulated real-world CI/CD practices. Each experiment targeted a specific class of vulnerabilities – credential leaks, workflow misconfigurations, access control weaknesses, and monitoring gaps – identified earlier in the background section. The results presented below compare the baseline pipeline with the secured configuration and highlight both quantitative improvements and qualitative insights gathered during testing.

4.1. Credential Leak Detection and Prevention

A core objective of the study was to evaluate how effectively secrets could be intercepted before compromising the pipeline. In the baseline configuration, deliberately injected dummy secrets were consistently committed into the GitHub repository without detection. This mirrors real-world observations that secret sprawl remains pervasive in modern software development.

After integrating the proposed two-layer scanning model, the results changed significantly. Pre-commit hooks identified and blocked exposed secrets directly on the developer side, preventing them from entering the version of history. In cases where developers attempted to bypass these controls, pre-receive hooks on the repository server provided a second layer of defense. Combined with TruffleHog and GitGuardian CLI, these measures intercepted 100% of injected secrets across 20 test runs, leaving no sensitive data in the central repository.

The logs further confirmed that early interception reduced remediation costs: in baseline runs, secret exposure required repository resets and history rewriting, while in secured runs the same issue was resolved instantly at commit time. This highlights the practical benefits of layered secret hygiene for both security and developer productivity.

4.2. Misconfiguration Detection and Mitigation

The second experiment assessed workflow misconfigurations using GHAST. In the baseline pipeline, static analysis identified multiple high-severity weaknesses:

Unpinned third-party actions (average of 7 per workflow),
Inline scripts without integrity validation (3–4 instances per workflow),
Insecure runner configurations (privilege escalation risks),
Excessive permissions in workflow tokens.

On average, 12–15 misconfigurations per pipeline were reported in the baseline setup. These vulnerabilities allowed arbitrary code execution and created a dependency on untrusted upstream code.

After integrating automated validation rules, the secured pipeline demonstrated zero unpinned actions and restricted use of inline scripts. GHAST flagged only low-severity issues, such as missing optional security headers, which were subsequently addressed. The experimental conclusion was clear: systematic configuration validation eliminated high-risk vulnerabilities and transformed workflows into auditable, reproducible pipelines.

Figure 4 presents the real-time monitoring and alerting pipeline. GitHub Actions events are captured through webhooks, processed by a Telegram bot, and delivered as notifications to developers. This integration ensures lightweight, real-time situational awareness for teams, even without enterprise-scale monitoring platforms.

Fig. 4. Baseline vs Secured pipeline

5. Discussion

The results of this study demonstrate that embedding layered security controls into CI/CD workflows effectively mitigates some of the most persistent risks, including secret leakage, misconfigurations, over-privileged tokens, and insufficient monitoring. In controlled experiments, the proposed framework consistently intercepted deliberately

injected secrets at the pre-commit or pre-receive stage, which aligns with earlier findings by Meli et al. [1] and GitGuardian reports [2] showing that secret sprawl remains one of the most critical yet preventable issues in modern pipelines. Similarly, the use of GHAST for workflow validation successfully flagged insecure YAML configurations, confirming observations by Benedetti et al. [15] and Delicheh and Mens [16] that automated detection of misconfigurations is crucial for preventing supply chain compromises.

When compared to prior studies on pipeline security [4,7], the results suggest that an integrated approach — combining secret hygiene, configuration validation, access control, and real-time monitoring — is more effective than isolated practices. The inclusion of lightweight alerting mechanisms (via Telegram) provides situational awareness at low cost, complementing larger enterprise-grade monitoring solutions discussed in industry analyses [13]. Importantly, the measured performance overhead of the secured pipeline was minimal, supporting the hypothesis that security can be embedded into DevOps workflows without compromising efficiency, a concern frequently highlighted in the literature [8,9].

The comparison between a baseline pipeline without security controls and the fully secured pipeline represents a simplified experimental setup designed to isolate the impact of the integrated framework. In practice, CI/CD pipelines often include partial security mechanisms. Therefore, the baseline should be interpreted as a reference configuration rather than a representation of real-world systems.

Despite these promising findings, several limitations should be acknowledged. First, the experiments were conducted in a controlled environment with synthetic secrets and misconfigurations, which may not fully capture the complexity of real-world software supply chains. Second, while GitHub Actions and Docker pipelines were used as a case study, other platforms may exhibit different vulnerabilities or require additional controls. Finally, the developer feedback collected was limited in scope and would benefit from larger-scale empirical studies.

The experimental evaluation was conducted using a limited number of controlled runs with synthetic credentials. While this setup ensures reproducibility and controlled testing conditions, it does not fully capture the complexity of real-world CI/CD environments, which involve multiple repositories, dynamic dependencies, and large-scale configurations. Therefore, the results should be interpreted as proof-of-concept validation rather than a large-scale empirical study.

Future research should expand the evaluation to include diverse CI/CD ecosystems, such as GitLab CI and Jenkins, and examine resilience against advanced adversarial tactics like dependency confusion and supply-chain poisoning. Furthermore, empirical studies with larger developer teams would help validate the usability and acceptance of security controls. Finally, integrating AI-based anomaly detection into the pipeline monitoring process may represent a valuable extension, enabling proactive detection of previously unseen attack patterns.

6. Conclusions

While the results demonstrate the effectiveness of the proposed framework in a controlled experimental setting, further validation is required to assess its applicability in large-scale and production-level CI/CD environments. Future research should focus on extending the evaluation across diverse repositories, complex workflows, and real-world datasets.

All the Declarations and Statements

Author Contributions Statement
Akzhibek Amirova – Conceptualization, Methodology, Writing – Review and Editing

Conflict of Interest Statement
The authors declare no conflicts of interest.

Funding Declaration
The authors received no external funding for this research.

Data Availability Statement
No new datasets were generated or analyzed during this study.

Ethical Declarations
This study did not involve human participants, animals, or any sensitive data requiring ethical approval.

Acknowledgments
We sincerely thank the experts for their professional evaluation and valuable recommendations, which have contributed

to improving the quality of the experiment and the reliability of its results.

Declaration of Generative AI in Scholarly Writing

During the preparation of this manuscript, the authors used ChatGPT based on the GPT-5.5 model only for language refinement, grammar checking, and improving the readability of the text. The scientific ideas, methodology, analysis, experimental results, and conclusions were entirely developed and written by the authors. The authors take full responsibility for the final content of the publication.

Abbreviations

The following abbreviations are used in this manuscript:

API – Application Programming Interface
CI/CD – Continuous Integration / Continuous Deployment
DevSecOps – Development, Security, and Operations
GHAST – GitHub Actions Static Analysis Tool
OWASP – Open Worldwide Application Security Project
RBAC – Role-Based Access Control
SAST – Static Application Security Testing

Appendix A/B/C…, with appendix title

None.

References

[1] Meli, M.; Gasser, L.; Ernst, M.D. "How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories," in Proceedings of the Network and Distributed System Security Symposium (NDSS), 2019

[2] GitGuardian. State of Secrets Sprawl Report 2024. Available online: https://www.gitguardian.com (accessed on 2 October 2025).

[3] SolarWinds. The SolarWinds Cyberattack Explained. SolarWinds, 2021.

[4] Saleh, S. M.; Al Ifat, M. N.; Madhavji, N. H.; Steinbacher, J. "A Threat-Oriented Study of API Security Challenges in CI/CD Pipelines," in 2025 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Shenzhen, China, 2025, pp. 1–8. DOI: 10.1109/CloudCom67567.2025.11331540.

[5] OWASP Foundation. OWASP Top 10 CI/CD Security Risks. 2023. Available online: https://owasp.org (accessed on 2 October 2025).

[6] Batista, L.; et al. "Securing DevOps by Identifying the Most Common Vulnerabilities in CI/CD Pipelines," in 2025 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2025, pp. 1–6. DOI: 10.1109/ISCC65549.2025.11326304

[7] Alshammari, B.; Singh, M. M. "A Systematic Literature Review on Tackling Cyber Threats for Cyber Logistic Chain and Conceptual Frameworks for Robust Detection Mechanisms," IEEE Access, vol. 13, pp. 67661–67692, 2025. DOI: 10.1109/ACCESS.2025.3552689.

[8] Bernardino, N. A.; Sequeira, B.; Piza, E.; Henriques, F.; Neves, F.; Reis, C. I. "Enhancing DevSecOps: Three Custom Tools for Continuous Security," in 2024 IEEE 11th International Conference on Cyber Security and Cloud Computing (CSCloud), 2024, pp. 53–58. DOI: 10.1109/CSCloud62866.2024.00017.

[9] Zhao, X.; et al. Identifying the Primary Dimensions of DevSecOps: A Multi-Aspects Study. Science of Computer Programming 2024, 230, 102912.

[10] OWASP Foundation. CI/CD Security Risks. OWASP, 2021.

[11] Pan, Z.; et al. "Ambush from All Sides: Understanding Security Threats in Open-Source Software CI/CD Pipelines," IEEE Transactions on Dependable and Secure Computing, vol. 21, no. 1, pp. 403–418, 2023. DOI: 10.1109/TDSC.2023.3253572.

[12] Saha, A.; et al. "Secrets in Source Code: Reducing False Positives Using Machine Learning," in 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), IEEE, 2020, pp. 168–175. DOI: 10.1109/COMSNETS48256.2020.9027350.

[13] Trend Micro. Security Analysis of GitHub Action Runners. Technical Report, 2023.

[14] GitHub. GitHub Actions Documentation. GitHub, 2023.

[15] Muralee, S.; Koishybayev, I.; Nahapetyan, A.; Tystahl, G.; Reaves, B.; Bianchi, A.; et al. "ARGUS: A Framework for Staged Static Taint Analysis of GitHub Workflows and Actions," in 32nd USENIX Security Symposium (USENIX Security 23), 2023, pp. 6983–7000.

[16] Riggio, E.; Pautasso, C. "Pipelines Under Pressure: An Empirical Study of Security Misconfigurations of GitHub Workflows," in International Conference on Product-Focused Software Process Improvement, Cham, Switzerland: Springer Nature Switzerland, 2025, pp. 220–236. DOI: 10.1007/978-3-032-12089-2_14.

[17] GitGuardian. State of Secrets Sprawl Report. GitGuardian, 2023.

[18] OWASP. Static Application Security Testing (SAST) Tools Guide. OWASP, 2022.

[19] SonarQube. Documentation and Security Rules. SonarSource, 2023.

[20] Snyk. Container and Dependency Scanning Documentation. Snyk, 2023.

[21] Docker. Docker Security Best Practices. Docker Inc., 2023.
[22] GitHub Security Lab. Security Advisories on Actions. GitHub, 2023.
[23] Semgrep. Open-Source Static Analysis Engine. r2c, 2022.
[24] Veracode. State of Software Security Report. Veracode, 2023.
[25] Checkmarx. Application Security Testing Solutions. Checkmarx Ltd., 2022.
[26] Clair. Container Vulnerability Scanner. CoreOS/Red Hat, 2021.
[27] Google Cloud. Supply Chain Security in CI/CD. Google, 2023.
[28] Thompson, K. “Reflections on Trusting Trust,” Communications of the ACM, vol. 27, no. 8, pp. 761–763, 1984. DOI: 10.1145/358198.358210.

Authors’ Profiles

Akzhibek Amirova was born in Kazakhstan. She received a Ph.D. degree in information in 2024. Her major field of study is information systems, with a focus on cybersecurity, and intelligent network systems. She is currently an Assistant Professor at Astana IT University, Astana, Kazakhstan. She has been actively involved in research projects related to cybersecurity, industrial Internet of Things (IoT), and next-generation communication systems, including 5G and 6G networks. Dr. Amirova is a member of IEEE and actively participates in academic and professional activities in the field of cybersecurity. She has contributed to multiple scientific publications in international journals and conferences and is involved in educational and research initiatives in information security and network systems.

How to cite this paper

Akzhibek Amirova, "Securing CI/CD Pipelines: A DevSecOps Framework for Preventing Credential Leaks and Misconfigurations", International Journal of Wireless and Microwave Technologies(IJWMT), Vol.16, No.4, pp. 291-299, 2026. DOI:10.5815/ijwmt.2026.04.16