Integrating Automated Security Checks into CI/CD Pipelines for Enhanced Vulnerability Detection and Remediation

Forename Surname

Student ID

Programme

National College of Ireland

**Abstract**

Rising popularization of Continuous Integration and Continuous Delivery (CI/CD) pipelines has served to speed up the delivery of software but has also increased the risks to security as any vulnerability or misconfiguration may quickly spread across the automated process. The study addresses the process of integrating automated security checks into the software development life cycle using CI/CD pipelines to help make the application more vulnerable to remediation. Expanding on the DevSecOps paradigm, the research will conduct an analysis of capabilities and limitations of discussed key tools, such as static application security testing (SAST), software composition analysis (SCA), and infrastructure-as-code (IaC) scanning. The study will answer what trade-offs exist both between security depth and pipeline performance and the formulation of best practices in relation to selection of tools, their configuration, and integration. The research will be conducted by integrating literature review, case study, and experimental testing on synthetic environments to provide actionable principles that can help balance the velocity of developing software with strong security in contemporary software delivery pipelines.

**Keywords:** CI/CD Security, DevSecOps, Secure Software Development, Vulnerability Scanning, Continuous Integration.

1. Introduction

Such practices as Continuous Integration and Continuous Delivery (CI/CD) have transformed the software development industry in recent years allowing teams to go through the cycle of defeat-testing and deployment much faster and more efficiently yet. Such contemporary DevOps processes have enabled the possibility to speed up the provision of the digital solutions and simultaneously guarantee their continuous advancement. But with this amplified pace, new problems have emerged the biggest one being the world of application security. With code modifications often being deployed to production, the conventional mode of viewing security as a design activity that is divorced coupled with the manner of conducting it as an end-of-the-line activity is no longer good. There is a higher likelihood of having deployed vulnerable or amateurly configured software due to lack of early-stage security controls and the consequences of this may result in breaches, loss of data or fines as far as regulation laws are concerned.

The research problem of the paper in question is that there are no automated security checks in CI/CD pipelines that are effectively integrated into the overall process. Despite the DevSecOps movement, most organizations are contending with tool compatibility, high rate of false positives, slowdowns in the pipeline, and poor coverage of security across a variety of environments. Unless there is a systematic process of integration, security is usually seen as an after-thought which opens systems to unnecessary risks.

The problem is of importance to the world of Secure Software Development, especially. With the increasing adoption of agile approaches and cloud-native architecture by more enterprises, automated security enforcement mechanisms and mechanisms that scale, operate in real-time, and enable an automated approach to security enforcement become essential. Ugwueze and Chukwunweike (2024), Yang (2025), and Baitha et al. (2024) mention DevSecOps and security automation as a transformative phenomenon, but at the same time refer to the lack of tool adoption and to trade-offs management and to workflow alignment. According to these studies, striking the right chords of good security and developer productivity is essential and this study aims at mitigating this aspect.

The core research question guiding this work is:

*How can automated security checks be effectively integrated into CI/CD pipelines to improve the detection and remediation of vulnerabilities and misconfigurations throughout the software development lifecycle?*

To address the problem, this paper suggests the following solution: a systematic study of existing security tools, such as Static Application Security Testing (SAST), Software Composition Analysis (SCA), and Infrastructure-as-Code (IaC) scan, and the creation of an efficient model of incorporating the tools into CI/CD pipelines. Quantifiable measures such as effectiveness, usability, and trade-offs of the chosen tools shall be conducted experimentally and by reference to literature.

This study will benefit science and practice as it will recognize the best practices, assess the practice implementation scenario, and produce a model of the implementation of security within CI/CD processes. It will be delivered within synthetic environments with the help of open-source tools to enable an ethical approach, prevent real-world vulnerabilities disclosure, and repeatability.

In this proposal the following organizational pattern is applied:

* A critical review of related literature is described in section 2.2, categorized under themes namely, current practices, tools, trade-offs and case studies.
* Section 2.3 defines the general research methodology or the proposed solution, steps of the project, evaluation approach, tools to be used and ethical issues to be addressed.
* Section 2.4 gives a full list of all academic sources that have been used in this proposal.

1. Literature Review

*CI/CD Security: Current Practices and Gaps*

The nature of Modern DevOps pipelines is that they are prone to security lapses because of their pace and automation. According to (None Oluwatosin Oluwatimileyin Abiona et al., 2024), although CI/CD helps introduce agility, it is its pace that usually leaves manual security behind, introducing risk hotspots in such things as the third-party dependency area and cloud infrastructure. On the same note, (Jani, 2023) state that most CI/CD implementations require the developers to implement security policies manually, which creates inconsistency and human error. Daniels and the U.S. Bureau of Labor Statistics both state that in order to fix this, automation is the answer but differ in where to implement it.

According to (Gopinath Kathiresan, 2022), it can be considered to resolve such issues as the involvement of developers in the training process and stronger policies related to the rules of governance. On the other hand, emphasize the direct introduction of automation into the pipeline involving DevSecOps practices. The distinction points out one of the main differences between the practices; although training can be helpful, it is not as scaling as automation. This contrast is important to our research because it states that tool integration into the CI/CD system ought to become automated (rather than just policy or awareness).

Such gap creates a logical transition to the following question, what machines products could be the most appropriate and how they could be embedded into the development processes.

*Tools for Security Automation in CI/CD Pipelines*

Automatic security-testing tools fall into three different broad categories, namely Static Application Security Testing (SAST), Software Composition Analysis (SCA) and Infrastructure-as-Code (IaC) scanners. Tools such as SonarQube are Static Analysis Source Tool (SAST) that are run on uncoded source code in search of bugs and security vulnerability. (Lenarduzzi et al., 2023) indicate that SonarQube enables remediating vulnerabilities early, yet it can have a significant number of false positives to the extreme that they are frustrating developers. OWASP Dependency-Check (SCA), on the contrary, targets to identify and flag the known vulnerabilities in the open-source libraries; as (Jani, 2023) claims, it provides a high degree of software integrity, but cannot detect either zero-day or logic bugs that have been overlooked during the static analysis.

This represents the importance of tool diversity and balance to find fewer solutions instead of a sole solution. Moreover, (Wadhams, Izurieta and Reinhold, 2024) comparatively criticize SCA and SAST tools, illustrating how integration issues, i.e., SCA has been demonstrated to have a large build latency and limited IDE (integrated development environment) support, frequently prevent integration. Such shortcomings are reasons why the work we have presented here should not only limit its performance to study accuracy of detection, but should also consider real world usability, and trade-offs.

Therefore, the subsequent step should be the assessment of the performance, effectiveness, and adoption cost of these tools within the real CI/CD adoption processes.

*Effectiveness vs. Developer Experience: The Trade-Off Dilemma*

Security checks that are automated are often faulted because they influence the pipeline performance and the productivity of the developer. The researchers revealed that adding IaC scanners such as Checkov resulted in 30 percent build delays in containerized settings (War et al., 2023). At the same time, claim that the overabundance of false-positives would cause developers to dismiss the results of scans completely, which is called an alert fatigue. This trade-off plays the role of a deterrent when it comes to DevSecOps adoption.

Clark, Smalling and Moses (2023), in contrast, discovered that the inserting of custom severity thresholds, as well as the filtering of the code in context, augmented the applicability of the scans by 45% with teams using Snyk. This implies that the configuration of the tools and not only the tool selection could affect usability. In our study, it means that automating checks does not only entail the level of efficiency of the tools involved but also the ways in which they are tuned and incorporated into the current workflows.

In order to understand better how successful teams can approach such a balance, it is of value to consider actual implementation of cases within context which will be measured in the following section.

*Case Studies and Patterns in DevSecOps Implementation*

Practice possibilities may be detected through real-world deployments to show theory in action. An example of GitLab case study (2023) implementing SAST, DAST, and container scans on its internal DevSecOps implementation demonstrated that this integration could be achieved without any meaningful slowdown in the build time through parallel pipelines and caching. On the other hand, one of the efforts in that regard by HashiCorp to implement IaC checks in their Terraform-based delivery pipeline (Rajapakse et al., 2022) became unsuccessful at first due to an incompatible toolchain and developer adoption. Such opposite results point to the fact that the organizational parameters, such as team training, pipeline design, and tool maturity may be considered as important as technical configuration.

In these examples, one could draw a common pattern: effective DevSecOps teams view security as a shared mission and adopt tools in an incremental way. This proves the not-so-radical notion that successful integration relates more to conscious steps towards implementation, feedback iterations and support culture to the proposed approach in this study.

*Identifying the Research Niche and Contribution*

The literature that has been analyzed points to the reported progress in the automation in CI/CD security but also indicates flaws in comprehensive system integration structures, test standards, or effectiveness investigations. No cohesively defined recommendations exist on how to choose which tool, how to handle trade-offs, and how to scale to the DevOps teams at varying levels of maturity. Although prior literature is helpful in understanding the specific tools and alignments, they do not cover an evidence-based, end-to-end, integration model, which considers tradeoffs between coverage of security and pipeline performance and developer experience.

The current work will address that gap by providing a practical handbook on automatically integrating security tools into CI/CD pipelines through the literature review as well as a practical experience. The deliverable will include the list of best practices and an assessment framework that could be replicated and modified in accordance with the infrastructure and risk level of teams.

1. Research Method and Specification

*Proposed Solution and Research Approach*

To answer the research question — *“How can automated security checks be effectively integrated into CI/CD pipelines to improve the detection and remediation of vulnerabilities and misconfigurations throughout the software development lifecycle?”* — this study proposes the design, integration, and evaluation of a prototype DevSecOps pipeline incorporating key automated security tools. The approach is applied, experimental, and engineering-based.

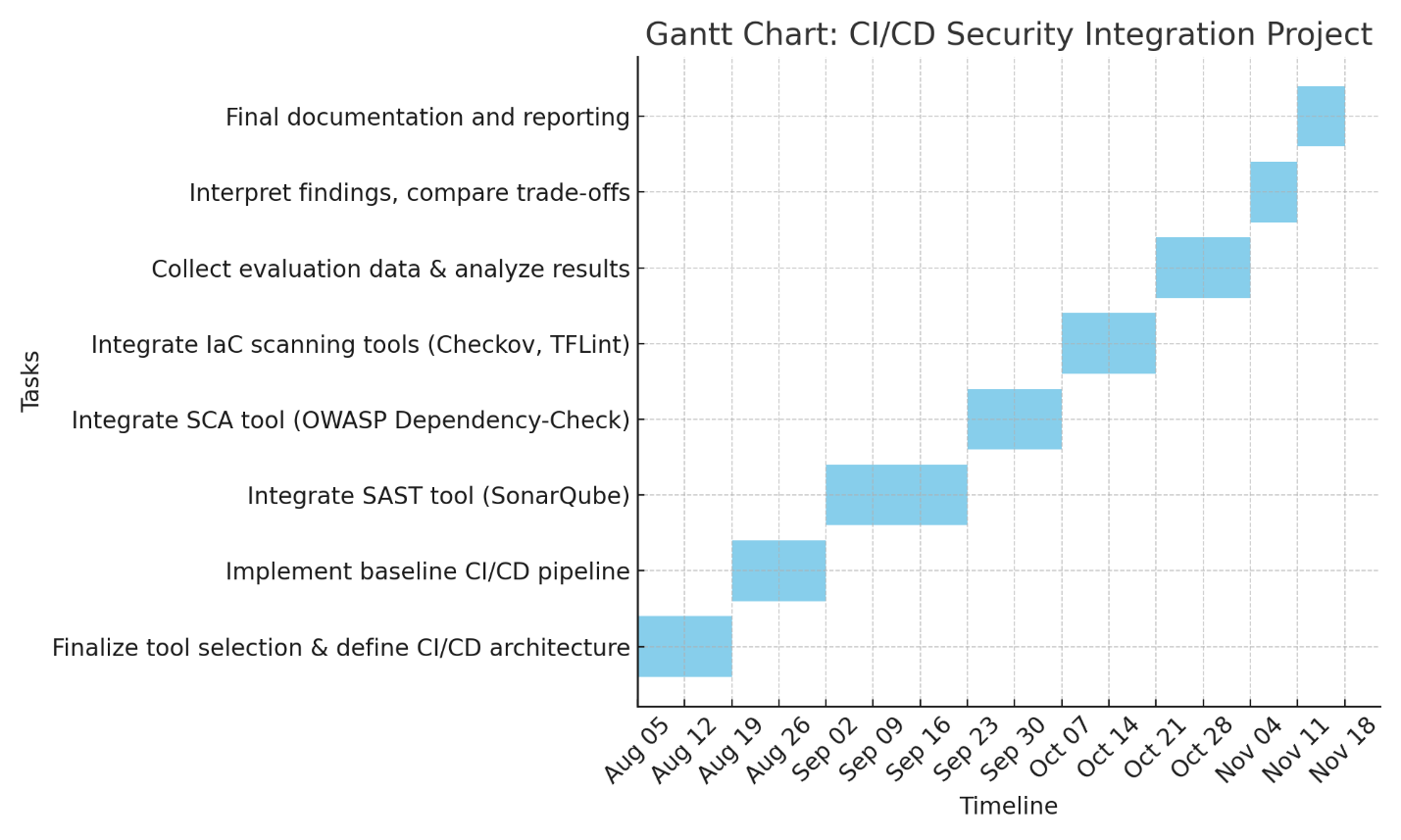
The recommendation is to incorporate a toolchain of automated scanners (SAST, SCA, and IaC tools) into a CI/CD pipeline that would be built upon an open-source DevOps stack. The tools will be compared based on factors like detection rate of vulnerabilities, false positives, ease of integration and effect on build time. The pipeline is going to be deployed in phases, where it would be modularly evaluated and iterated.

This approach combines:

* Descriptive research to explore existing practices and tools.
* Design science methodology to develop the prototype.
* Empirical experimentation to validate the effectiveness of the solution.

*Research Plan and Expected Activities*

The project will follow a structured, milestone-based methodology aligned with the Capstone schedule. Below are the key stages:



A **Gantt chart** will visually represent this timeline with dependencies and progress milestones.

*Tools, Platforms, and Test Data*

Tools:

* CI/CD platform: GitLab CI/CD (with optional Jenkins comparison)
* SAST tool: SonarQube (open-source edition)
* SCA tool: OWASP Dependency-Check
* IaC scanning: Checkov (for Terraform), TFLint (optional)
* Container scanning: Trivy
* Repository: GitHub/GitLab private repository
* Build environment: Docker + Linux VMs
* Data visualization: Grafana + Prometheus (for metrics collection)

Test Data and Applications:

* Sample microservice applications (e.g., PetClinic or open-source Flask/Django apps)
* Deliberately vulnerable IaC templates (e.g., from Vulnado, Damn Vulnerable Terraform)
* Public CVE databases for benchmarking SCA tool results

*Evaluation Strategy*

The approach will be evaluated based on quantitative and qualitative evidence. The primary metrics will include:

* Detection Rate: Number and severity of vulnerabilities identified by each tool.
* False Positives: Manual review of flagged issues to assess accuracy.
* Pipeline Performance: Build time before vs. after integration.
* Integration Complexity: Time/effort required to implement each tool.
* Developer Experience Impact: Measured through simulated delays and usability notes.

Results will be compared across stages (no tools, single tool, full stack) to establish the trade-offs between security and efficiency. The effectiveness of the solution will be defined by the ability to maintain acceptable build performance while significantly enhancing vulnerability detection.

*Ethical Considerations*

This research will not expose real user data, private repositories, or proprietary applications. Instead, it uses open-source sample projects and synthetic test cases. The following ethical safeguards will be observed:

* All test applications will be scanned in a local, isolated environment to prevent unintentional exposure.
* Tools used (e.g., SonarQube, OWASP scanners) are open-source and approved for academic and community use.
* No private or confidential data will be processed.
* If vulnerability reports are shared publicly (e.g., in documentation), they will be anonymized.
* The research adheres to NCI’s ethical framework, and all necessary declaration forms will be submitted with this proposal.

Any future deployment of this solution within real-world environments will require organizational consent and access control, and will be governed by standard DevSecOps practices.

# References

Clark, B., Smalling, E. and Moses, J. (2023). *Critical WebP 0-day security CVE-2023-4863 impacts wider software ecosystem*. [online] Snyk. Available at: https://snyk.io/blog/critical-webp-0-day-cve-2023-4863/ [Accessed 1 Aug. 2025].

Gopinath Kathiresan (2022). Evaluating the impact of Devsecops on software quality: A systematic review and empirical study. *World Journal of Advanced Research and Reviews*, [online] 14(1), pp.644–653. doi:https://doi.org/10.30574/wjarr.2022.14.1.0267.

Jani, Y. (2023). Implementing Continuous Integration and Continuous Deployment (CI/CD) in Modern Software Development. *International Journal of Science and Research (IJSR)*, 12(6), pp.2984–2987. doi:https://doi.org/10.21275/sr24716120535.

Lenarduzzi, V., Pecorelli, F., Saarimaki, N., Lujan, S. and Palomba, F. (2023). A critical comparison on six static analysis tools: Detection, agreement, and precision. *Journal of Systems and Software*, [online] 198, p.111575. doi:https://doi.org/10.1016/j.jss.2022.111575.

None Oluwatosin Oluwatimileyin Abiona, Oladapo, J., Oluwole, N., None Oyekunle Claudius Oyeniran, Okechukwu, A. and Abiola, N. (2024). The emergence and importance of DevSecOps: Integrating and reviewing security practices within the DevOps pipeline. *World Journal of Advanced Engineering Technology and Sciences*, 11(2), pp.127–133. doi:https://doi.org/10.30574/wjaets.2024.11.2.0093.

Rajapakse, R.N., Zahedi, M., Babar, M.A. and Shen, H. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology*, 141, p.106700. doi:https://doi.org/10.1016/j.infsof.2021.106700.

Wadhams, Z.D., Izurieta, C. and Reinhold, A.M. (2024). Barriers to Using Static Application Security Testing (SAST) Tools: A Literature Review. *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering Workshops*, [online] pp.161–166. doi:https://doi.org/10.1145/3691621.3694947.

War, A., Habib, A., Diallo, A., Klein, J. and Bissyandé, T.F. (2023). Security Vulnerabilities in Infrastructure as Code: What, How Many, and Who? *Research Square (Research Square)*. [online] doi:https://doi.org/10.21203/rs.3.rs-3600645/v1.