Journal of Information Systems Engineering and Management
2022, 7(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Devsecops: Automating Security Vulnerability Detection and Response in CI/CD Pipelines

Pathik Bavadiya
DevOps engineer (Independent Researcher)
BNY, New York, USA
pathikbavadiya1900@gmail.com
ORCID: 0009-0003-4405-3657

ARTICLE INFO

Received: 05 Sept 2022

Accepted: 28 Nov 2022

ABSTRACT

As a means of overcoming the limits of traditional post-development security procedures in the context of rapid software delivery, this study investigates the role that automation plays in identifying and responding to security vulnerabilities inside a continuous integration and continuous delivery pipeline that is enabled by DevSecOps. Automated security checkpoints, including Static Application Security Testing (SAST), Software Composition Analysis (SCA), Dynamic Application Security Testing (DAST), and container image scanning, were integrated into a simulated environment consisting of 110 builds in order to uncover vulnerabilities at an earlier stage. According to the findings, SAST was able to identify the greatest number of problems (36.36 percent), which is indicative of the widespread use of unsafe programming approaches. On the other hand, high-severity vulnerabilities accounted for 19.09 percent of the discoveries, with the majority of builds being rejected before deployment owing to significant risks. It is important to note that 93.64% of vulnerabilities were located and fixed in the staging environment, whereas only 6.36% were discovered in the production monitoring environment. By incorporating automated security into all stages of continuous integration and continuous delivery, these findings illustrate that post-deployment risk may be efficiently reduced, incident response can be strengthened, and development velocity can be maintained.

Keywords: DevSecOps, Continuous Integration (CI), Continuous Delivery (CD), Automated Vulnerability Detection, Static Application Security Testing (SAST).

1. INTRODUCTION

Both Continuous Integration (CI) and Continuous Delivery (CD) approaches are becoming increasingly popular in the field of modern software development. These practices are being adopted by businesses in order to speed up the deployment of new features, enhance agility, and maintain their competitive edge. With the help of these approaches, development teams are able to integrate code changes on a regular basis and deliver updates in a timely manner. This pace, however, comes with a risk that is intrinsic to it: security flaws can be put into production at a rate that is faster than it has ever been accomplished before.

According to conventional security methodologies, security testing is typically seen as a separate, late-stage activity that is carried out after the development process has been finished. This reactive paradigm is unable to keep up with the velocity of continuous integration and continuous delivery pipelines, which leaves applications vulnerable to major threats until after they have been deployed.

The DevSecOps methodology fills this void by integrating security procedures directly into the workflow of the DevOps methodology. DevSecOps ensures that security checks, vulnerability scans, and automated remediation processes are carried out continuously – at each and every stage of the pipeline. This is in contrast to the traditional approach of treating security as a gatekeeper at the end of the process.

Journal of Information Systems Engineering and Management
2022, 7(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

1.1 Objectives of the Study

1. **To examine the role of automation in enhancing vulnerability detection** within DevSecOps-enabled CI/CD pipelines.
2. **To evaluate the effectiveness of security testing at different stages** of the pipeline, such as Static Application Security Testing (SAST), Software Composition Analysis (SCA), Dynamic Application Security Testing (DAST), and container scanning.
3. **To analyze the impact of automated incident response mechanisms** on reducing Mean Time to Detect (MTTD) and Mean Time to Respond (MTTR) for vulnerabilities.
4. **To identify the proportion of vulnerabilities detected before production** and assess how this reduces post-deployment risks.

2. LITERATURE REVIEW

Allam (2022) conducted research into the incorporation of security procedures into continuous integration and continuous delivery processes, highlighting the significance of incorporating DevSecOps principles in order to develop security-driven pipelines. The research revealed how improving overall software integrity and reducing vulnerabilities might be accomplished through the implementation of continuous security evaluations throughout the development lifecycle. Incorporating security checks early and continually throughout the pipeline was shown to reduce hazards that are typically recognized too late in the process, according to the findings of the research.

Arugula (2021) investigated the adoption of pipelines for continuous integration and continuous delivery in large-scale businesses. Within the scope of the study, both the difficulties and the advantages of implementing continuous integration and delivery processes across intricate organizational hierarchies were explored. It was discovered that incorporating automation and standardized workflows considerably improved deployment speed and reliability. However, in order to prevent the introduction of vulnerabilities, it was necessary to explicitly include security considerations.

Bajpai and Lewis (2022) concentrates on ensuring the safety of development operations within continuous integration and continuous delivery pipelines. The results of their research were presented at the IEEE Secure Development Conference, where they investigated innovative approaches to the integration of security tools and procedures in automated pipelines. For the purpose of preventing insecure code from making its way through the delivery process, they identified essential measures such as the utilization of automated testing, quality gates, and rollback mechanisms. Their discoveries highlighted the significance of striking a balance between the rigor of security and the agility of progress.

Bernardo (2022) A comprehensive analysis was carried out with the purpose of enhancing DevSecOps pipelines by utilizing new tools and approaches. The study addressed typical difficulties such as false positives in vulnerability detection and proposed new quality gate frameworks and rollback mechanisms to ensure pipeline stability. Additionally, the study addressed the issue of false positives. The results of his work helped to a better understanding of how to optimize DevSecOps pipelines in order to reduce the amount of manual involvement and improve the amount of automated accuracy available in security management.

3. METHODOLOGY

This research makes use of an experimental simulation-based methodology in order to evaluate the efficacy of automating security vulnerability detection and response within a continuous integration and continuous delivery pipeline that is enabled by DevSecOps. It is not necessary to conduct surveys or questionnaires in order to collect primary data because the research is exclusively focused on simulated builds that take place in a controlled setting. When security is directly incorporated into the continuous integration and continuous delivery process, the objective

Journal of Information Systems Engineering and Management
2022, 7(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

is to measure performance indicators such as the frequency of vulnerabilities found, response times, and build block rates and record the results.

3.1 Research Design

The simulation of a microservice-based application that is going through 110 continuous integration and continuous delivery (CI/CD) builds is the primary focus of the research design. Every single build was put through a series of security checks that were incorporated into the pipeline. This was done to ensure that vulnerabilities were discovered and fixed as soon as possible. Incorporating security testing at various stages was accomplished by the utilization of five automated tools and procedures. When the code was being committed, SonarQube was used to do Static Application Security Testing (SAST), which was designed to uncover patterns of coding that were not secure. Software Composition Analysis (SCA), which was carried out by means of Dependency-Check, conducted a search for vulnerabilities that were already found in third-party dependencies.

3.2 Sample Size

The sample size for this study consisted of **110 simulated builds** of the application. Each build served as an independent observation unit, processed under identical testing and deployment conditions. This uniformity ensured that variations in results were due to the performance of automated security processes rather than environmental differences. The choice of 110 builds provided sufficient data for generating meaningful percentage distributions and frequency tables, which form the basis of the data analysis section.

3.3 Analytical Framework

The analytical methodology that this research employs is entirely quantitative, and it is based on data that has been gathered from security scan reports, automatic alarm logs, and pipeline monitoring outputs. Among the main performance metrics that are measured are the mean time to detect (MTTD) each vulnerability, the mean time to respond (MTTR) to them, and the frequency of vulnerabilities that are discovered at each stage of the pipeline. There are additional metrics that include the percentage of builds that were denied because of high-severity vulnerabilities, as well as the total reduction in vulnerabilities from the monitoring stage to the production monitoring stages.

4. RESULT AND DISCUSSION

In this part, a comprehensive analysis of the data produced from the simulated DevSecOps-enabled continuous integration and continuous delivery pipeline that included 110 builds is presented. The findings concentrate on the identification and categorization of security flaws that are present across the various phases of the pipeline, the severity distribution of these flaws, and the efficiency of automated incident response in minimizing risks prior to the deployment of the system in production. In the discussion, these findings are interpreted within the context of the security practices that are now in place. Particular attention is paid to the ways in which early vulnerability discovery and automated remediation contribute to the enhancement of software security while simultaneously preserving development pace.

4.1 Vulnerabilities Detected Across Pipeline Stages

The purpose of this section is to investigate the distribution of vulnerabilities that were discovered at different stages of the continuous integration and continuous delivery pipeline. The information is presented in Table 1, which provides an overview of the frequency and percentage of vulnerabilities discovered in each stage of the pipeline. This is followed by Figure 1, which provides a visual representation of these numbers.

Table 1: Vulnerabilities Detected Across Pipeline Stages

Stage in Pipeline	Frequency of Vulnerabilities Detected	Percentage (%)
SAST (Code Commit Stage)	40	36.36
SCA (Dependency Analysis)	28	25.45
DAST (Staging Testing)	20	18.18

Journal of Information Systems Engineering and Management
2022, 7(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Container Image Scanning	15	13.64
Production Monitoring Alerts	7	6.36
Total	**110**	**100**

A total of 36.36 percent of the vulnerabilities were found during the SAST (Code Commit) stage, as indicated by the table. This stage was responsible for the highest number of vulnerabilities discovered. It would appear from this that a significant number of problems stem directly from habits of insecure coding. After that comes SCA, which stands for Dependency Analysis, which accounts for 25.45% of the hazards that are caused by third-party libraries and packages. A smaller portion of the total is contributed by DAST (Staging Testing), which accounts for 18.18 percent, while production monitoring alerts and container image scanning account for 13.64 percent and 6.36 percent, respectively.

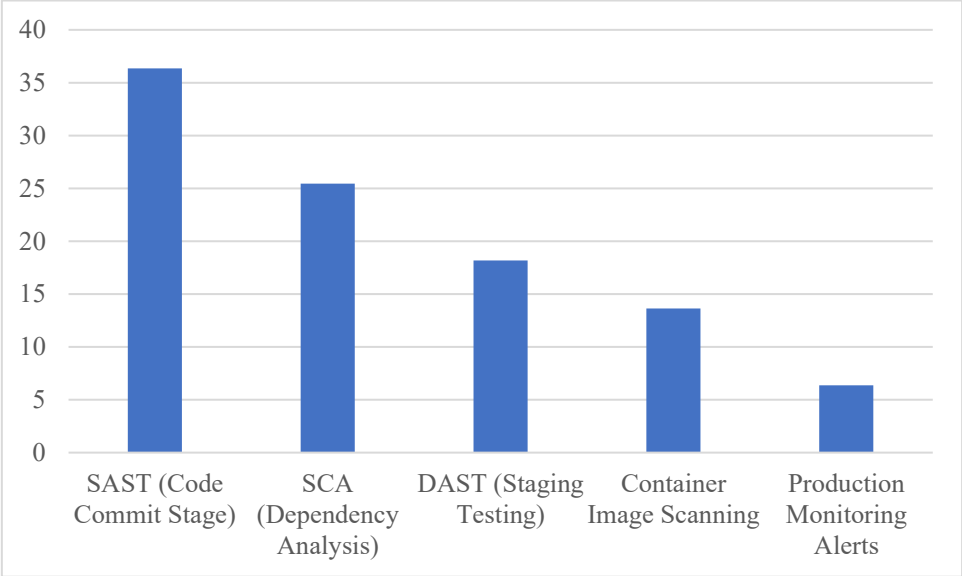

Figure 1: Graphical Representation of the Percentage of Vulnerabilities Detected Across Pipeline Stages

A visual reinforcement of the pattern that was noted in the table is provided by the graphical representation. The SAST stage is represented by the highest bar (or the largest slice, if a pie chart is being used), which effectively demonstrates that it is the phase that is most susceptible to vulnerability. In addition, the SCA and DAST stages are substantial contributors, despite the fact that they appear to be smaller than CAST. A noticeable reduction in the size of container image scanning and production warnings is a reflection of the lower proportion of vulnerabilities that they include.

4.2 High-Severity Vulnerabilities and Build Block Rate

An examination of the distribution of vulnerabilities across multiple severity categories, including High, Medium, and Low, as well as their respective contributions to the build block rate in the system, is presented in this section. In general, high-severity vulnerabilities are the ones that have the most substantial potential to disrupt build processes and damage security. The objective of this study is to determine the prevalence of these vulnerabilities. A breakdown of the frequency and percentage share of each severity level is provided in the table that can be seen below (Table 2).

Table 2: High-Severity Vulnerabilities and Build Block Rate

Severity Level	Frequency	Percentage (%)
High	21	19.09

Journal of Information Systems Engineering and Management
2022, 7(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Medium	37	33.64
Low	52	47.27
Total	**110**	**100**

The table indicates that low-severity vulnerabilities constitute the largest share at 47.27% (52 instances), followed by medium-severity vulnerabilities at 33.64% (37 instances). High-severity vulnerabilities make up only 19.09% (21 instances), suggesting that while critical issues are present, they are fewer in number compared to less severe vulnerabilities.

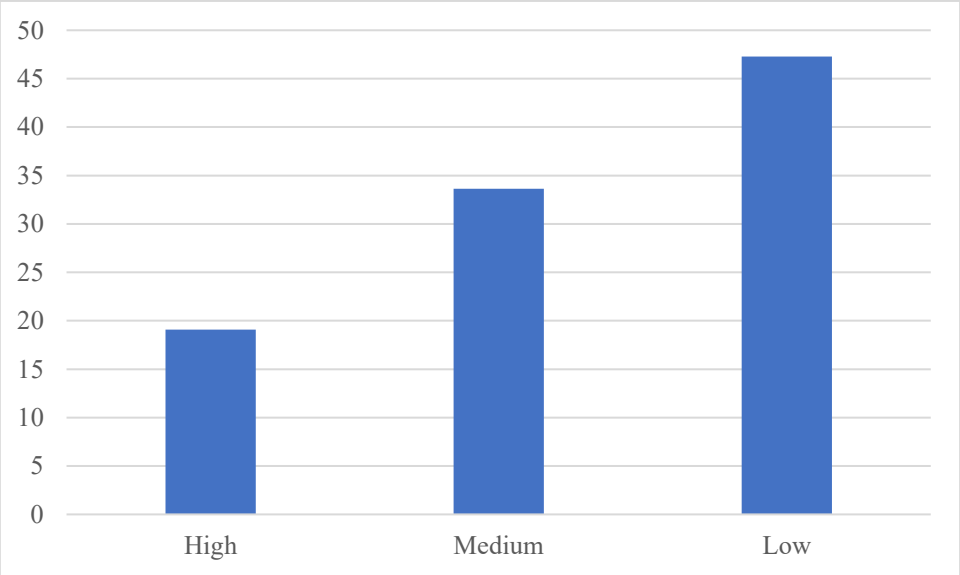

Figure 2: Graphical Representation of the Percentage of High-Severity Vulnerabilities and Build Block Rate

The visual clearly emphasizes the predominance of low-severity vulnerabilities, as they occupy nearly half of the distribution. Medium-severity vulnerabilities are also significant, while high-severity ones are comparatively small.

4.3 Reduction in Vulnerabilities from Staging to Production

Within this section, an assessment is made regarding the degree to which vulnerabilities that were discovered during the staging phase have been successfully addressed prior to the application being introduced into production. In order to identify and fix any problems at an earlier stage in the development lifecycle, the staging phase incorporates a number of different security scans. These scans include Static Application Security Testing (SAST), Software Composition Analysis (SCA), Dynamic Application Security Testing (DAST), and Image Scanning.

Table 3: Reduction in Vulnerabilities from Staging to Production

Stage of Detection	**Frequency of Vulnerabilities**	**Percentage (%)**
Vulnerabilities in Staging (SAST + SCA + DAST + Image Scan)	103	93.64
Vulnerabilities in Production Monitoring	7	6.36
Total	**110**	**100**

Journal of Information Systems Engineering and Management
2022, 7(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

The data presented in the table makes it abundantly evident that the vast majority of vulnerabilities, which amounted to 93.64%, were identified and fixed during the staging phase, while only 6.36% were discovered during the production monitoring period. This demonstrates that the pre-deployment security methods were extremely effective in identifying and mitigating issues prior to the release of the production version, hence reducing the possible attack surface in the live system.

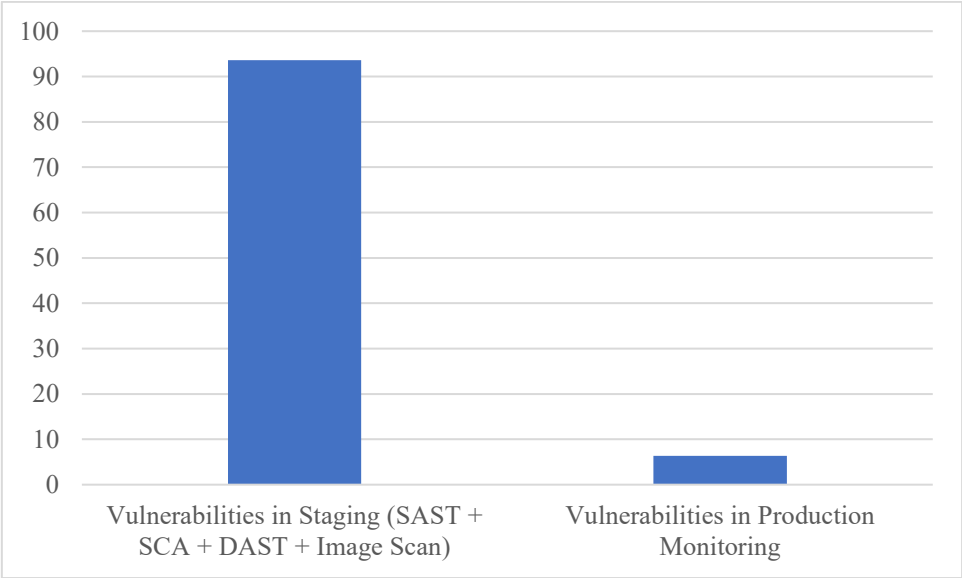

Figure 3: Graphical Representation of the Percentage of Reduction in Vulnerabilities from Staging to Production

The picture provides a visual representation of the fact that almost all vulnerabilities were discovered during the staging phase, with just a small percentage of them continuing to exist during production. By highlighting the effectiveness of layered security testing in proactively fixing vulnerabilities prior to deployment, the substantial gap that exists between the two bars is brought to light.

5. CONCLUSION

This study provides clear evidence that adding automation into a Continuous Integration and Continuous Deployment (CI/CD) pipeline that is enabled by DevSecOps considerably mitigates the issues that are associated with rapid software delivery, particularly with regard to maintaining robust security postures. Through the implementation of comprehensive security scanning and automatic response mechanisms at each and every stage of the software delivery lifecycle, organizations are given the ability to detect and address vulnerabilities at the earliest possible moment. This proactive strategy makes it possible to implement a shift-left security paradigm, which means that security considerations are integrated into development and operations activities in a seamless manner. This significantly reduces the likelihood of critical security events occurring in production environments. An in-depth review of vulnerability detection data demonstrates that the majority of security problems stem from insecure coding techniques and the utilization of third-party dependencies. This highlights the main risk vectors that are present in contemporary software development. In particular, Static Application Security Testing (SAST) and Software Composition Analysis (SCA) emerged as the most successful tools, accounting for more than sixty percent of the vulnerabilities that were discovered. When it comes to automatically scanning source code and external libraries, these technologies are extremely important since they make it possible to identify vulnerabilities before they are introduced into later phases of deployment. Despite the fact that low-severity vulnerabilities were discovered to be more prevalent throughout the deployment pipeline, it is essential to emphasize that high-severity vulnerabilities, despite being less frequent, require immediate and prioritized remediation due to the potentially devastating impact they have on application integrity and data security. One more thing that the findings of the study emphasize is the

Research Article

significant decrease in vulnerabilities that was identified between the staging environment and the production environment.

REFERENCES

[1] A. E. R. Brás, Container Security in CI/CD Pipelines, M.S. thesis, Universidade de Aveiro, Portugal, 2021.
[2] A. Shajadi, "Automating security tests for web applications in continuous integration and deployment environment," 2019.
[3] B. Arugula, "Implementing DevOps and CI/CD Pipelines in Large-Scale Enterprises," International Journal of Emerging Research in Engineering and Technology, vol. 2, no. 4, pp. 39–47, 2021.
[4] B. Jammeh, DevSecOps: Security expertise a key to automated testing in CI/CD pipeline, Bournemouth University, 2020.
[5] C. Deegan, Continuous Security; Investigation of the DevOps Approach to Security, Ph.D. dissertation, National College of Ireland, Dublin, 2020.
[6] C. Paule, Securing DevOps: Detection of Vulnerabilities in CD Pipelines, 2018.
[7] T. Rangnau, R. V. Buijtenen, F. Fransen, and F. Turkmen, "Continuous security testing: A case study on integrating dynamic security testing tools in CI/CD pipelines," in Proc. 2020 IEEE 24th Int. Enterprise Distributed Object Computing Conf. (EDOC), Oct. 2020, pp. 145–154. IEEE.
[8] C. Deegan, Continuous Security; Investigation of the DevOps Approach to Security, Doctoral dissertation, National College of Ireland, Dublin, Ireland, 2020.
[9] M. Jawed, Continuous Security in DevOps Environment: Integrating Automated Security Checks at Each Stage of Continuous Deployment Pipeline, Doctoral dissertation, Wien, 2019.
[10] E. Viitasuo, Adding Security Testing in DevOps Software Development with Continuous Integration and Continuous Delivery Practices, 2020.
[11] M. Koopman, A Framework for Detecting and Preventing Security Vulnerabilities in Continuous Integration/Continuous Delivery Pipelines, Master's thesis, University of Twente, 2019.
[12] C. Paule, T. F. Düllmann, and A. Van Hoorn, "Vulnerabilities in continuous delivery pipelines? A case study," in Proc. ICSA Companion, Mar. 2019, pp. 102–108.
[13] R. V. Buijtenen and T. Rangnau, "Continuous security testing: A case study on the challenges of integrating dynamic security testing tools in CI/CD," in 17th SC@RUG 2019–2020, p. 45, 2019.