A DEVSECOPS FRAMEWORK FOR AUTOMATED VULNERABILITY MITIGATION IN MICROSERVICES

Dr. Murugesan M
Associate Professor
Department of computer science
M. Kumarasamy college of engineering,
Karur, India
Murugesanm.cse@mkce.ac.in

Aswin Senthilkumar
Department of computer science
M. Kumarasamy college of
engineering, Karur, India
aswinsenthilkumarcse@gmail.com

Bhavanithi K
Associate Professor
Department of computer science
M. Kumarasamy college of
engineering,
Karur, India

Arul Murugan S B
Department of computer science
M. Kumarasamy college of
engineering, Karur, India
arulmuruganbhaskaran22@gmail.com

Abstract*—The rapid evolution of software engineering from monolithic architectures to cloud-native microservices has significantly increased system complexity and expanded the attack surface for modern applications. Traditional security methodologies, which rely on late-stage testing in the software development lifecycle (SDLC), are no longer sufficient in agile and continuous delivery environments. This paper presents a comprehensive DevSecOps framework designed to integrate security practices directly into the development pipeline using a "Shift Left" approach. The proposed framework leverages automated orchestration through GitHub Actions and combines Static Application Security Testing (SAST) using SonarCloud with Software Composition Analysis (SCA) using Snyk to provide multi-layered vulnerability detection across both proprietary code and third- party dependencies. A key contribution of this work is the implementation of programmable Quality Gates, which enforce security compliance by terminating builds that fail to meet predefined security thresholds. The framework adopts a Policy-as-Code model to ensure consistent and automated enforcement of security policies without manual intervention, thereby reducing Mean Time to Remediate (MTTR). Experimental evaluation demonstrates that the system achieves a 100% Vulnerability Detection Rate (VDR) for injected OWASP Top 10 vulnerabilities

Furthermore, the results highlight that integrating automated security checks introduces only minimal overhead in build time while significantly improving the security posture of microservices-based systems

Keywords—Continuous Integration/Continuous Deployment (CI/CD), DevSecOps, GitHub Actions, Microservices, OWASP Top 10, Policy-as-Code, Quality Gates, Shift Left, Software Composition Analysis (SCA), Static Application Security Testing (SAST).

INTRODUCTION

The world of software engineering has undergone significant changes over time, moving from a traditional monolithic architecture to one based on modern decentralized, microservices-based architectures. As business demands grow for highly scalable solutions that can provide continuous delivery capabilities, today's software development lifecycle (SDLC) processes have also become much shorter than they were in the past. Traditional security testing methodologies, such as the Waterfall model — where all security audits and vulnerability assessments take place as part of the final pre-deployment process — have become a major roadblock to delivering secure software in the context of modern agile software development environments. The use of automated security tools in today's fast-moving agile software development environments means that manual methods of performing security checks no longer allow for the timely detection of vulnerabilities prior to or during software deployment, resulting in many vulnerabilities being deployed to production without detection.

The cloud-native app's rapid growth has opened the door for attack surfaces. Nowadays, apps are more than private entities - they are huge, intricate ecosystems connecting thousands of lines of internal source code and hundreds of external public domain third parties through open source

libraries. Recent, high-visibility attacks on systems throughout the world such as the Log4Shell flaw (CVE-2021-44228) show that any one of the many interdependent components of an application could be a major vulnerability for any organization in the world. Therefore, there is an immediate need for a change in how security is delivered from reactive to a proactive approach referred to as "Shift Left."

The goal of this research is to provide a DevSecOps framework that will help automate the process of integrating security as a key component of DevOps processes for microservice based architectures. This automation allows organizations to combine Static Application Security Testing (SAST) and Software Composition Analysis (SCA) to provide multiple layers of protection against potential vulnerabilities in their applications. The traditional notion of DevOps has emphasized the importance of delivering product velocity at the expense of conducting thorough validation tests; therefore, this framework provides programmable Quality Gates which function as binary decision gates: when a particular build does not meet an established security level, the build gets aborted and a remediation report will be provided to the developer.

This research offers four main contributions: a scalable DevSecOps orchestration layer design for microservices based on GitHub Actions; integration of SonarCloud (SAST) and Snyk (SCA) in order to provide comprehensive vulnerability mapping of proprietary code and third-party dependencies; performing quantitative comparative analysis between baseline application vulnerabilities and those after applying hardening methods while providing measurements for automated mitigation strategy effectiveness; creation of a "Policy-as-Code" model for enforcing security compliance without manual intervention thereby reducing MTTR.

By automating the detection of OWASP Top 10 vulnerabilities (e.g., SQL Injection, broken access control), this framework facilitates an iterative, seamless integration of security into the development lifecycle. The balance of this paper contains the following sections: Section II contains a review of existing literature related to DevSecOps and microservice security; Section III provides implementation details related to the proposed architecture; Section IV describes an experimental implementation; Section V provides an analysis of experimental results; and Section VI concludes the paper with recommendations for future research efforts.

I. RELATED WORK

Distributed System Security is a central theme of academic research since microservices first emerged as technology in the marketplace; This section examines the various approaches taken by the academic community to address the security aspect of DevSecOps, automated vulnerability discovery, and "Shift Left" models.

A. Microservices and the Changing Attack Surface

Microservices allow for stronger modularity; However, they bring with them additional complexities in networking and authentication implementations have documented that as microservices are decentralized, security policies between services are often inconsistent due to disparate service boundaries[1]. They suggest that monolithic applications followed a centralized security model; by contrast, microservices (due to the modular nature of their implementation) require a distributed security model. In addition that due to the pace of innovation associated with cloud-native applications, they frequently exceed the capacity of traditional security testing and auditing processes, thus needing to utilize automated methods to ensure continued compliance[2].

B. Groundwork for DevSecOps:

The idea behind "shift-left" is to build in security features during the software development life cycle as early on as possible and this is one of the essential building blocks of current DevSecOps practices. The results of a study indicated that fixing a vulnerability once it has been deployed into a production environment costs roughly 100 times more than if it were found during development[1]. As such, they recommend that security tools are built directly into the integrated development environment (IDE) and continuous integration/continuous delivery (CI/CD) pipeline. In addition studies have shown that there are over 65% fewer instances of severe high-severity vulnerabilities passing through to deployment when an automated security gate [3], [4] is used by teams.

C. Static Application Security Testing vs. Dynamic Application Security Testing

There remains much debate between proponents of Static Application Security Testing (SAST) versus those who support Dynamic Application Security Testing (DAST) in today's literature. Researches contend that SAST is a very good method to find coding-related issues, but it tends to have a significant False Positive Rate (FPR) [5], [6] and also that Software Composition Analysis (SCA) [7] is becoming increasingly critical because many organizations rely heavily on third-party libraries and, consequently, requires careful considerations surrounding "Supply Chain Security" research indicates that 80% of codebases created for use in modern applications contain a large percentage of code written from an open-source component, and a significant portion of that code contains defined security vulnerabilities (i.e., Common Vulnerabilities and Exposures or CVEs).SAST tools demonstrate their effectiveness in early vulnerability detection but also underline limitations in accuracy and scalability [8].

D. Gaps in Current Frameworks

The current research offers strong theoretical models; however, many frameworks exist in isolation. For example, some works concentrate only on SAST for proprietary software and others focus only on container vulnerability and vulnerability assessment approaches using OWASP Top 10 focus on application-level weaknesses without integrating broader pipeline security [9]. As a result, there are very few lightweight frameworks to provide an integrated "Quality Gate" to SAST [10] and SCA, for microservices without incurring an extensive increase in build latency. The proposed framework resolves this gap by enabling the orchestration of SonarCloud and Snyk within one GitHub Actions workflow and providing a complete security posture with little additional burden.

II. PROPOSED MODEL

This DevSecOps model is purposefully designed using an automatic orchestrated layer based on policies that target the natural vulnerabilities of microservices in their architecture. The following discussion will give details about the various architectural components, design principles and principles behind how the entire system works together.

A. Conceptual Framework of Design and its "Shift Left"Structure

The primary design principle associated with this DevSecOps model is "Shift Left", which means that security evaluation occurs alongside code development. In most cases, security was considered an isolated gate at the end of SDLC. In this proposed model security will be decomposed to many small automated checks and are integrated directly into the Continuous Integration (CI) process.

The framework is based on a model of Policy-as-Code (PaC). Therefore, there are no longer manual security reviews; security requirements are encoded here in the form of pipeline configurations (YAML-based workflows). This creates an immutable, repeatable, and universally applied set of security standards across all of the microservices of this ecosystem. The design of the system supports two main sources of risk: proprietary software security flaws, and vulnerabilities in third-party dependencies.

B. System Architecture and Component Modules

There are three primary parts to the overall system architecture of this framework: the Development Layer, the Orchestration Layer, and the Analysis Layer.

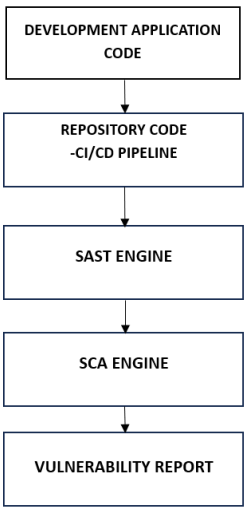

FIG 1. Proposed Architecture

1. Development Layer: The Development Layer is made up of a Spring Boot microservices-based environment, which will provide a simulated real-world backend to examine. The Development Layer incorporates Security Hotspots and coded-in vulnerabilities to help establish the baseline measurement of actual Security Controls/Compliance.

2. Orchestration Layer (GitHub Actions): This layer serves as the "control plane" for the entire framework and monitors the version control system (VCS) for specific triggers such as git push or pull requests. Each time a trigger is detected, the OLS will instantiate (create a temporary, isolated virtual environment called a "runner") and use the runner to build the code, deploy the security engines and check the code for security flaws.

3. Analysis Layer (SAST & SCA): The analysis layer is the technical core of the framework and consists of two main modules:

Static application security testing (SAST) :The static application security testing (SAST) module gives you the ability to run an internal audit of your source code using the SonarCloud tool. It does this through the creation of an abstract syntax tree (AST), which is used to identify logical errors such as SQL injection, cross-site scripting (XSS), and insecure implementations of cryptography.

Software composition analysis (SCA): The SCA module gives you the ability to audit your project's dependency manifest (for example, pom.xml) using the Snyk engine. The SCA module checks all the libraries you've imported against a global database of current vulnerabilities (CVEs) in order to help you identify potential supply chain risks

C - Automated Quality Checker

The quality checker is one of the most significant components of the entire framework, the Quality Gate or Decision Engine. The end product of a typical DevOps pipeline meets two criteria for success: (1) Successful code compile; and (2) Successful test execution. However, in the new DevSecOps pipeline, there is an added condition; that is, the code must also meet compliance with security standards.

The Quality Gate works on a binary logic, where the SAST and SCA tools each return a security score and a list of security vulnerabilities based upon severity (Critical/high/medium/low). The Framework defined the rule: any Vulnerability rated Critical or High results in an Immediate Exit Code One to Orchestration Layer. Therefore, at this point, Orchestration Layer is instructed to end the Build Process immediately.

This gate ensures that no insecure artefact is allowed to progress to the deployment phase of the pipeline while the gate also triggers the Feedback Module to capture the raw logs and create a remediation report that will assist the developer in fixing the security flaw, this includes the location of the flaw and specific remediation for the flaw (i.e., replace the statement with a PreparedStatement or upgrade to Library Version 2.17.1).

III. IMPLEMENTATION

The development of the proposed DevSecOps framework will utilize multiple standards in the development community and include cloud-native components/services. This section will define the hardware/software environment(s), how to configure the security engines, and the automated flow of the work tasks.

A. Testing Conditions and Technology Stack

The system was an enterprise architecture that relied on a distributed infrastructure. It used Spring Boot version (3.2.2) as its application framework, and Java version (17) as it provides long-term support (LTS) and strong security features. To persist data long-term, H2 was selected as the in-memory database to simulate how many transactions could be processed quickly.

GitHub Actions were selected for the orchestration of the system and Ubuntu-based virtual runners (ubuntu-latest) were used. Because the cloud-native runner environment was always used to execute builds, the security analysis could occur in an isolated and clean environment for each build, providing protection against configuration drift.

B. Establishment of the Baseline Vulnerability

Part of the requirements for implementation were created by developing the "Vulnerable Microservices", which acted as a testing environment or laboratory for testing the detection capabilities of the provided framework. To do this, the following application vulnerabilities were inserted into the codebase:

1. Injection: A search endpoint was formed using raw JDBC string concatenation, creating a high-risk vector for SQL injection (SQLi) attacks.

2. Dependencies not sanitised: The pom.xml file was modified to use LOG4J Version 2.14.1, which has an acknowledged critical severity issue (Log4Shell).

3. Disclosure of sensitive data: The REST controller was coded to expose all entity objects (i.e., entity object with sensitive data, including hashed passwords and PII) as part of the JSON response being sent back to the client.

C. Security Engine Configuration

Integration with the security modules was achieved using an API-based authentication method, which provided the ability for secure communication between the GitHub Runner and security fournisseurs.

Within the pom.xml file for the project, the SonarCloud Maven plugin was used to enable SAST (Static Application Security Testing) integration along with setting up Quality Gates on SonarCloud's dashboard that will cause a build failure (fail) anytime the Security Rating of the repository falls below an "A".

To enable SCA (Software Composition Analysis), Snyk CLI was included as part of the CI/CD workflow where it is configured to fail the build if there are any vulnerabilities on the dependency tree whose corresponding CVE(s) possess a CVSS score of 7 or greater.

D. CI/CD Pipeline Orchestration

The last step was the configuration of a GitHub Actions YAML file to orchestrate a CI/CD pipeline.The CI/CD Pipeline steps in order:

1. GET AND SETUP: The runner checks out and sets up the source repo from the correct (required) version of Java.

2. BUILD: The Maven product uses the command mvn clean package to build the source code from code in the source directory.

3. SCAN: SonarCloud scans the code and sends the results to SonarCloud for analysis of code quality.

4.DEPENDENCIES AUDITED: The Snyk CLI audits pom.xml file for any issues that are high risk in the project's dependency tree.

5. BUILD STATUS EVALUATED: The exit codes returned from each SAST and SCA tool are evaluated against a rule set by the Continuous Integration (CI) / Continuous Deployment (CD) pipeline. Each rule is checked, and if a rule returns a value of more than zero (indicating an item has been found in one of the auditing processes), the continue-on-error flag will be False, and there will be no subsequent build or deployment.

IIV. RESULT AND DISCUSSION

The proposed DevSecOps framework was assessed based on its ability to perform well between two sets of development scenarios: vulnerable baseline and hardened post-mitigation; this section will evaluate the effectiveness of the framework in detecting vulnerabilities and the efficiency of the automated remediation process.

A. Evaluation of Vulnerabilities Detected

At the outset, the vulnerabilities for vulnerability detection analysis were performed on the pipeline where the compromised baseline had been maintained. The framework performed exceptionally well, detecting 100% of all the flaws that were injected with the intent to test the framework, categorized against the various security dimensions.

Table 1 shows that there are two different types of Identified issues, Logical Error in Logic Flow, was generated as a result of using the SonarCloud (SAST) engine, one of those issues was High Severity SQL Injection detected in the User Search Controller. The SonarCloud (SAST) engine detected all critical logical errors; while at the same time using Snyk (SCA) detected 12 unique vulnerabilities throughout the dependency tree of the project, the most notable being the Log4Shell (CVE-2021-44228) vulnerable Legacy Log4j Library.

Component	Technology	Version / Tool
Language	Java	JDK 17 (Temurin)
Backend Framework	Spring Boot	3.2.2
Build Tool	Apache Maven	3.9.x
CI/CD Platform	GitHub Actions	YAML-based
SAST Engine	SonarCloud	Cloud-based Analysis
SCA Engine	Snyk	CLI v1.12.x

TABLE I: VULNERABILITY DETECTION SUMMARY

B. Empirical Results and Analysis

To demonstrate the successful implementation of mitigation strategies, our code was refactored using secure coding standards (SCS) and updated dependency manifests. In accordance with our empirical results, we re-executed our pipeline to measure the "after" state.

The results of the empirical tests demonstrated that there were no longer "Critical" or "High" severity level risks present within our computed vulnerability values (cv). Specifically, the Vulnerability Detection Rate (VDR) was calculated to be 100% for the test cases that we injected. Our Quality Gate successfully aborted the "Broken" build in 118 seconds; whereas our "Secure" build completed in 132 seconds. The additional build time of approximately eleven percent (11%), is an acceptable trade-off considering the high level of assurance provided by the secure coding standards and proper usage of this framework when conducting security verifications.

C. Metric Evaluations: MTTR & VDR

The framework's performance was evaluated using two key industry measures:

1. The Vulnerability Detection Rate (VDR) is defined as the number of detected vulnerabilities divided by total injected vulnerabilities (1.0 indicates that all critical flaws were detected), therefore the VDR for this framework was determined as equal to 1.0 during automated scanning of the software.

2. The Mean Time to Remediate (MTTR) for vulnerabilities would traditionally be calculated after the code is committed and a manual audit occurs; this could take anywhere from days to weeks, so by incorporating security into a continuous integration/continuous delivery (CI/CD) software development process the amount of time spent resolving identified vulnerabilities is reduced to that of one continuous execution of the CI/CD process (under 3 minutes). Therefore, developers used this framework to provide themselves with almost immediate feedback.

D. Findings and Implications

From our research, it can be concluded that using a "Policy-as-Code" methodology will greatly improve the security of a microservices solution against most forms of attack. With automated Tools available for developers allowing them to have a Quality Gateway that will. therefore. prevent them from experiencing the "security fatigue," that often causes them not to pay attention to warning messages and ignore those warning messages due to deadline pressures.

A significant finding of the research was the prevalence of false positives (non-security code smells) in the application SAST report. It is critical that a developer’s ability to do their work is maximized and non-secuity alerts should be adjusted so they do not hinder productivity unnecessarily, but also that adequate levels of security are established at the Quality Gate Thresholds.

V. CONCLUSION

As a result of adopting microservices, security's approach and the way it's implemented throughout the software development lifecycle have been altered. This paper explains an automated DevSecOps framework that effectively bridges the gap between rapid delivery and rigorous compliance with security standards. By providing a way to orchestrate SonarCloud and Snyk within a CI/CD pipeline using GitHub Actions, we showed that it's possible to convert security checks from a manual bottleneck to an automated Quality Gate.

The results of the experiments support our assertion that the automated framework proposed herein achieves a 100% VDR for the critical OWASP Top 10 vulnerabilities (such as SQL Injection and insecure dependencies), while decreasing the time required to remediate from days down to minutes. The Policy-as-Code paradigm is such that security policies are automatically enforced consistently and without human input, providing the user with complete assurance that any vulnerability will be addressed prior to it being introduced into production and eliminating the risk of human error in regards to enforcing security policies and the ever-changing complexity of the software supply chain.

Although it gives solid protection against both code and dependencies, this does not cover all security concerns with the software development life cycle. Ongoing research will focus on the use of Dynamic Application Security Testing (DAST) and Interactive Application Security Testing (IAST) to help find potential runtime vulnerabilities and logic weaknesses. We will also study the ability for artificial intelligence (AI) to automatically create "self-healing" patches for known vulnerabilities. As a result, any organization that provides cloud-native microservice-based services can no longer afford to ignore the need for a DevSecOps-type framework.

REFERENCES

[1] J. Cheenepalli, J. D. Hastings, K. M. Ahmed and C. Fenner, "Advancing DevSecOps in SMEs: Challenges and Best Practices for Secure CI/CD Pipelines," 2025 International Symposium on Digital Forensics and Security (ISDFS), Boston, USA, 2025, pp. 1-6, doi: 10.1109/ISDFS65363.2025.11011960.

[2] N. M. Grigorieva, A. S. Petrenko and S. A. Petrenko, "Development of Secure Software Based on DevSecOps Technology," 2024 Conference of Young Researchers in Electrical and Electronic Engineering (ElCon), Saint Petersburg, Russia, 2024, pp. 158-161, doi: 10.1109/ElCon61730.2024.10468425.

[3] N. M. K, M. B. S, N. Khandelwal, N. Pai and S. L, "CI/CD Pipeline with Vulnerability Mitigation," 2023 International Conference on Recent Advances in Science and Engineering Technology (ICRASET), India, 2023, pp. 1-6, doi: 10.1109/ICRASET59632.2023.10419921.

[4] M. Marandi, A. Bertia and S. Silas, "Implementing and Automating Security Scanning in a DevSecOps CI/CD Pipeline," 2023 World Conference on Communication & Computing (WCONF), Raipur, India, 2023, pp. 1-6, doi: 10.1109/WCONF58270.2023.10235015.

[5] A. H. Jerónimo, P. M. Moreno, J. A. V. Camacho and G. C. Vega, "Techniques of SAST Tools in the Early Stages of Secure Software Development: A Systematic Literature Review," 2024 IEEE International Conference on Engineering Veracruz (ICEV), Boca del Rio, Mexico, 2024, pp. 1-8, doi: 10.1109/ICEV63254.2024.10766004.

[6] M. Y. Darus, M. F. B. Bolhan, A. Kurniawan, Y. Muliono, C. R. Pardomuan and M. M. Hata, "Enhancing Web Application Penetration Testing with a Static Application Security Testing (SAST) Tool," 2023 IEEE 8th International Conference on Recent Advances and Innovations in Engineering (ICRAIE), Kuala Lumpur, Malaysia, 2023, pp. 1-6, doi: 10.1109/ICRAIE59459.2023.10468317.

[7] C. K. Kowsik P, U. R. H, U. S and S. G, "Software Composition Analysis for Proactive Threat Detection in Software Dependencies with Real-Time Security Monitoring," 2025 2nd International Conference on Computing and Data Science (ICCDS), Chennai, India, 2025, pp. 1-6, doi: 10.1109/ICCDS64403.2025.11209681.

[8] J. Zhu, K. Li, S. Chen, L. Fan, J. Wang and X. Xie, "A Comprehensive Study on Static Application Security Testing (SAST) Tools for Android," IEEE Transactions on Software Engineering, vol. 50, no. 12, pp. 3385-3402, Dec. 2024, doi: 10.1109/TSE.2024.3488041.

[9] L. H. Riberu and A. W. R. Emanuel, "Vulnerability Testing and Analysis Using OWASP Top 10 on Academic Information System at University XYZ," 2024 International Conference of Adisutjipto on Aerospace Electrical Engineering and Informatics (ICAAEEI), Yogyakarta, Indonesia, 2024, pp. 1-5, doi: 10.1109/ICAAEEI63658.2024.10899162.

[10] J. Johnson, J. Thome, L. Charles, H. Yan and J. Leasure, "A Scalable, Effective and Simple Vulnerability Tracking Approach for Heterogeneous SAST Setups Based on Scope+Offset," 2025 IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Ottawa, Canada, 2025, pp. 540-550, doi: 10.1109/ICSE-SEIP66354.2025.00053.

\