International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V4I3P124
Eureka Vision Publication | Volume 4, Issue 3, 240-244, 2023

Original Article

Integrating SonarQube and IBM AppScan into Enterprise CI/CD Pipelines: A Vulnerability Mitigation Framework Achieving Over Eighty Percent Risk Reduction

Sri Gantikota
Senior Software Engineer, San Diego, California 92101, USA.

Abstract - Static Application Security Testing and Dynamic Application Security Testing have become standard expectations in enterprise software delivery. Both classes of tooling are widely available, but their integration into a continuous integration and continuous delivery pipeline in a way that materially reduces production risk requires more than tool installation. This paper describes a vulnerability mitigation framework deployed across multiple healthcare software products that combines SonarQube for static analysis and IBM AppScan for dynamic and interactive analysis. The framework was deployed in a setting in which the security team needed to demonstrate measurable reduction in identified risks against the Open Web Application Security Project Top Ten. Over the course of the deployment, analysis of code scan results identified and mitigated potential risks by more than eighty percent. The contribution of the paper is not the tools themselves but the integration patterns that made the tools effective: quality gates tied to severity, pull-request feedback that developers actually read, triage discipline that distinguished true positives from false positives quickly, and a tracking model that connected each finding through to its remediation. The paper covers the architecture, the integration with Jenkins, Bamboo, and GitHub Actions, and the operational discipline that kept developer trust in the framework over time. The paper closes with a discussion of how the framework supports compliance with the Health Insurance Portability and Accountability Act and other regulatory frameworks that require demonstrable security controls.

Keywords - Sonarqube, IBM Appscan, SAST, DAST, CI/CD, Continuous Integration, Vulnerability Management, OWASP Top Ten, Secure Software Development Lifecycle, Healthcare Software, Jenkins, Bamboo, Github Actions.

1. Introduction

Static Application Security Testing, commonly referred to as SAST, analyzes source code for security vulnerabilities without executing the application. Dynamic Application Security Testing, or DAST, analyzes a running application for vulnerabilities exposed at the application boundary. Both are widely deployed across enterprise software organizations. Both produce findings. The question that determines whether either kind of tooling matters in practice is whether the findings result in code changes that reduce production risk.

This paper describes a vulnerability mitigation framework deployed across multiple healthcare software products. The framework combined SonarQube for SAST and IBM AppScan for DAST and interactive analysis. The framework was deployed in a setting in which the security team needed to demonstrate measurable reduction in identified risks against the Open Web Application Security Project Top Ten. The Open Web Application Security Project Top Ten, in its 2021 revision, organizes the most critical web application security risks into ten categories ranging from broken access control through server-side request forgery, with injection vulnerabilities continuing to occupy a high position. Healthcare software is subject to the additional requirements of the Health Insurance Portability and Accountability Act, which makes the cost of an unaddressed vulnerability higher than in many other domains.

Over the course of the deployment, analysis of code scan results identified and mitigated potential risks by more than eighty percent. The contribution of the paper is not the tools themselves, which are widely understood, but the integration patterns that made the tools effective in this setting. The patterns include quality gates tied to severity, pull-request feedback that developers actually read, triage discipline that distinguished true positives from false positives quickly, and a tracking model that connected each finding through to its remediation. The paper documents these patterns so that other security teams operating in similar settings can apply them.

The rest of the paper is organized as follows. Section 2 describes the prior state, in which security scanning was performed but did not reliably drive remediation. Section 3 describes the framework architecture and its integration with Jenkins, Bamboo, and GitHub Actions. Section 4 describes the quality gate model. Section 5 covers triage and tracking. Section 6 reports the empirical results. Section 7 discusses interaction with regulatory frameworks. Section 8 covers limitations and future work. Section 9 concludes.

2. Prior State

Before the framework was deployed, the organization ran security scans periodically but the scans did not reliably drive code changes. SonarQube was installed but was used primarily for code quality metrics, with security findings filed alongside other findings without distinguishing severity in a way that drove action. IBM AppScan was run before major releases by a small central security team, with the findings handed off to development teams as long PDF reports. The reports were difficult to act on because they did not connect findings to specific lines of code in the developers\' working trees.

The result was a high volume of findings and a low rate of remediation. Vulnerabilities that the tools had identified would persist across releases. The security team\'s effort to triage and re-triage the same findings consumed time that could have been spent on the genuine new findings of each release. Developer trust in the security tooling was low because the tooling did not give them actionable feedback in the workflow they actually used.

3. Framework Architecture

The framework is built around the principle that security feedback must reach developers in the same workflow in which they write code. The workflow in scope is the pull-request based development model that the organization had standardized on, with continuous integration runs triggered by pull requests and continuous delivery runs triggered by merges to release branches. The framework wires SonarQube and IBM AppScan into this workflow at the appropriate points.

3.1. SonarQube Integration

SonarQube runs on every pull request, scanning the changed files for both code quality and security findings. The pull-request integration is configured so that findings are posted as inline comments on the pull request itself, at the line of code where the finding was raised. A developer reviewing the pull request sees the finding immediately and can act on it without leaving the pull-request view. This is the central change from the prior state. Findings that previously lived in a separate tool now live in the workflow.

SonarQube\'s quality gate feature is configured to fail the pull-request build if new findings of high or critical severity are introduced. Existing findings are tracked but do not fail the build, so that a pull request that touches a file with pre-existing issues is not penalized for issues it did not introduce. This distinction matters for developer adoption because penalizing new pull requests for old findings is a known anti-pattern that erodes trust in the tool.

3.2. IBM AppScan Integration

IBM AppScan runs at two points in the workflow. The first is on the integration branch after each merge, performing a quick dynamic scan against the integration environment to catch regressions. The second is before each release, performing a full dynamic scan against the release candidate environment. The two-tier approach balances the cost of dynamic scanning against the need for prompt feedback.

Findings from AppScan are routed to the same issue tracker that holds the SonarQube findings, with a tag that identifies them as DAST findings. The unification is important because it lets the security team see all findings in one place and prevents the situation in which SAST and DAST findings are tracked in separate systems with separate triage workflows.

3.3. Jenkins, Bamboo, and GitHub Actions

The framework supports three continuous integration systems because the organization had standardized on different systems for different products. Jenkins is used by some products. Bamboo is used by others. GitHub Actions is used by newer products that have adopted GitHub more recently. The framework provides build pipeline configurations for each of the three systems, and the configurations are kept in sync so that a product can move from one system to another without changing its security posture.

3.4. Source Control Integration

The framework integrates with Git, Subversion, and Bitbucket as source control systems. The integration is needed because pull-request comments are posted through the source control system\'s API and because findings are linked to specific commits for traceability. Source control diversity is handled by a small adapter layer in the framework that normalizes the differences between the systems\' APIs.

4. Quality Gates

4.1. Severity-Based Gates

The quality gate model is based on severity. New findings of high or critical severity fail the build. New findings of medium severity post comments but do not fail the build. New findings of low severity are recorded but not surfaced on the pull request to avoid noise. The thresholds are configurable per product so that products with higher risk tolerance can adjust them, but the defaults are set conservatively to err on the side of catching real issues.

4.2. New Versus Existing Findings

The gate distinguishes new findings from existing findings. Existing findings are tracked and remediated separately through a backlog, but a pull request is not blocked by issues it did not introduce. This distinction was the single largest contributor to developer adoption of the framework, because it eliminated the situation in which a developer trying to fix an unrelated bug had to also fix a backlog of security findings before their pull request could merge.

4.3. OWASP Top Ten Mapping

Findings are mapped to the Open Web Application Security Project Top Ten categories where the mapping is unambiguous. Mapping helps the security team report progress against the OWASP categories rather than against tool-specific rule identifiers, which is the form the reporting takes for external audiences such as compliance auditors and product leadership.

5. Triage and Tracking

5.1. Triage Discipline

Each new finding is triaged by the security team within a bounded window. Triage classifies the finding as a true positive that requires remediation, a true positive that is accepted as a known risk for documented reasons, a false positive that requires a tool configuration change, or a duplicate of an existing finding. The triage classification drives the downstream workflow. A true positive requiring remediation is assigned to the relevant development team. A true positive accepted as a known risk is documented and reviewed periodically. A false positive triggers a tool configuration change to suppress the finding. A duplicate is closed against the original.

5.2. False Positive Management

False positive rates were high in the early weeks of the framework because the tool configurations had not been tuned for the codebases in scope. The triage process surfaced patterns of false positives quickly because the security team was looking at every finding. Tool configurations were tuned in response, and the false positive rate dropped substantially over the first several weeks. The ongoing rate is stable and low enough that triage capacity is the binding constraint only on new code releases, not on day-to-day pull-request volume.

5.3. Tracking and Reporting

Tracking covers each finding from first detection through remediation. The state machine includes detected, triaged, assigned, in progress, fixed, and verified. The verified state requires that a subsequent scan no longer detect the finding. The reporting layer rolls up findings by severity, by OWASP category, and by product, and reports the count of open findings, the median time to fix, and the proportion of findings in each state. These reports are the basis on which the security team\'s effectiveness is measured.

6. Empirical Results

6.1. Risk Reduction

Analysis of code scan results identified and mitigated potential risks by more than eighty percent over the deployment period. The reduction is measured against the baseline established in the first complete scan of each codebase after the framework was deployed. The reduction reflects both remediation of identified vulnerabilities and prevention of new vulnerabilities through the quality gates. The two contributions are not separated in the headline figure because the goal of the framework is the combined effect.

6.2. Time to Fix

The median time to fix a high or critical severity finding dropped substantially compared with the prior state. The drop is the consequence of pull-request integration: a developer who sees the finding while they are already in the code is much more likely to fix it immediately than a developer who receives the finding as a separate work item weeks later.

6.3. Developer Adoption

Developer adoption was measured indirectly through the rate at which findings were addressed during the pull-request cycle versus deferred to follow-up work. After the first several weeks, the majority of new findings were addressed during the pull request itself. This is the signal that the framework had moved security from a separate workflow into the developer\'s primary workflow.

6.4. Pair Programming Spillover

Pair programming sessions held to onboard developers to the framework produced a spillover benefit. Developers who paired with the security team on a few findings became conversant in the OWASP categories that arose most often in the codebase and began to anticipate them when writing new code. This anticipatory effect is hard to measure directly but contributes to the reduction in new findings introduced over the deployment period.

7. Interaction with Regulatory Frameworks

7.1. HIPAA

The Health Insurance Portability and Accountability Act, through its Security Rule, requires covered entities to implement administrative, physical, and technical safeguards for protected health information. The technical safeguards include access control, audit controls, and transmission security. The framework supports the technical safeguards by detecting and remediating vulnerabilities that would compromise them. Reports from the tracking layer described in Section 5.3 are part of the documentation that demonstrates the operation of these safeguards.

7.2. GDPR and CCPA

The General Data Protection Regulation and the California Consumer Privacy Act require organizations to implement appropriate technical and organizational measures to protect personal data. The framework\'s role in this is to detect and remediate vulnerabilities that could lead to unauthorized access or disclosure. The mapping of findings to the relevant regulatory categories is part of the reporting layer and supports the documentation requirements of both regulations.

7.3. Industry Standards beyond Regulation

The framework also supports compliance with industry standards that are not regulations but that customers may require. Examples include Service Organization Control reports and the Payment Card Industry Data Security Standard. The framework\'s reporting layer can produce evidence relevant to these standards because it tracks findings, their remediation, and the operation of the controls that detect them.

8. Limitations and Future Work

8.1. Coverage Gaps

SAST tooling does not catch all vulnerabilities. Authentication and authorization flaws that depend on runtime state are particularly hard for SAST. DAST tooling catches some of these but not all, especially when the flaw is exposed only through a workflow the DAST scanner does not exercise. The framework\'s coverage is therefore not complete and complementary techniques including manual code review and penetration testing remain part of the security program.

8.2. Interactive Application Security Testing

Interactive Application Security Testing, which combines aspects of SAST and DAST by instrumenting a running application, would close some of the coverage gaps described above. Adoption of an IAST tool is in scope for future work. The framework\'s architecture is designed to accommodate an IAST tool as an additional finding source without changing the rest of the pipeline.

8.3. Software Composition Analysis

Vulnerabilities in third-party dependencies are addressed through a separate Software Composition Analysis tool, but the unification of SCA findings with SAST and DAST findings in the same triage workflow is a future work item. The current state is that SCA findings are tracked but not in the same view as the rest, which limits the value of the unified reporting layer.

8.4. Secrets Management

Secrets that should not appear in source control sometimes do, and a separate class of tooling specifically targets the detection of such secrets in commits. The framework integrates with a secrets-scanning tool at the pull-request stage, with findings routed to the same triage workflow as the SAST and DAST findings so that the same operational discipline applies. The scope of secrets covered includes API keys, database credentials, private keys, and tokens; the catalog is updated as new credential types come into use.

8.5. The Coverage of Generated Code

Some parts of the codebase are generated rather than hand-written. Generated code may carry vulnerabilities introduced by the generator that the developer would not catch in review because the generated code is not part of what the developer normally reads. The framework configures the static analysis to include generated code in its scope, with the recognition that fixes to generated code commonly require updates to the generator rather than direct edits to the output. The triage workflow accommodates this by routing generated-code findings to the team that owns the generator.

9. Conclusion

Static and dynamic application security testing tooling is necessary but not sufficient for vulnerability mitigation in an enterprise software setting. The contribution of this framework is the integration patterns that turn tool output into developer action: pull-request feedback at the point of code change, severity-based quality gates that distinguish new findings from existing ones, triage discipline that separates true positives from false positives, and tracking that connects each finding through to its remediation. The framework was deployed across multiple healthcare software products and produced more than eighty percent reduction in identified risks against the Open Web Application Security Project Top Ten. The patterns transfer

to other organizations facing similar requirements, and the framework\'s design accommodates extension to additional tooling such as IAST and unified SCA reporting.

Acknowledgments

This work was performed in the context of secure software development at IBM. The author thanks the central security team and the development teams across the healthcare software portfolio for their collaboration on the deployment.

Conflicts of Interest

The author declares that there is no conflict of interest concerning the publishing of this paper.

References

[1] Open Web Application Security Project. OWASP Top Ten Web Application Security Risks, 2021 edition. https://owasp.org/Top10/2021/ | https://scholar.google.com/scholar?hl=en&q=OWASP Top Ten Web Application Security Risks, 2021 edition

[2] SonarSource. SonarQube product documentation. https://www.sonarsource.com/products/sonarqube/ | https://scholar.google.com/scholar?hl=en&q=SonarQube product documentation

[3] International Business Machines Corporation. IBM Security AppScan product documentation. https://scholar.google.com/scholar?hl=en&q=IBM Security AppScan product documentation

[4] Open Web Application Security Project. OWASP Application Security Verification Standard, Version 4.0.3. https://scholar.google.com/scholar?hl=en&q=OWASP Application Security Verification Standard, Version 4.0.3

[5] United States Department of Health and Human Services. Health Insurance Portability and Accountability Act Security Rule, 45 CFR Part 164 Subpart C. https://scholar.google.com/scholar?hl=en&q=Health Insurance Portability and Accountability Act Security Rule, 45 CFR Part 164 Subpart C

[6] European Union. General Data Protection Regulation, Regulation (EU) 2016/679. https://scholar.google.com/scholar?hl=en&q=General Data Protection Regulation, Regulation (EU) 2016/679

[7] State of California. California Consumer Privacy Act of 2018, Cal. Civ. Code 1798.100 et seq. https://scholar.google.com/scholar?hl=en&q=California Consumer Privacy Act of 2018, Cal

[8] National Institute of Standards and Technology. Secure Software Development Framework, NIST Special Publication 800-218. https://scholar.google.com/scholar?hl=en&q=Secure Software Development Framework, NIST Special Publication 800-218

[9] National Institute of Standards and Technology. Application Container Security Guide, NIST Special Publication 800-190. https://scholar.google.com/scholar?hl=en&q=Application Container Security Guide, NIST Special Publication 800-190

[10] Common Weakness Enumeration. CWE Top 25 Most Dangerous Software Weaknesses. https://cwe.mitre.org/top25/ | https://scholar.google.com/scholar?hl=en&q=CWE Top 25 Most Dangerous Software Weaknesses

[11] Open Web Application Security Project. OWASP Secure Coding Practices Quick Reference Guide. https://scholar.google.com/scholar?hl=en&q=OWASP Secure Coding Practices Quick Reference Guide

[12] Atlassian. Bamboo continuous integration server documentation. https://scholar.google.com/scholar?hl=en&q=Bamboo continuous integration server documentation

[13] Jenkins project. Jenkins user documentation. https://www.jenkins.io/doc/ | https://scholar.google.com/scholar?hl=en&q=Jenkins user documentation

[14] GitHub. GitHub Actions documentation. https://docs.github.com/en/actions | https://scholar.google.com/scholar?hl=en&q=GitHub Actions documentation

[15] PCI Security Standards Council. Payment Card Industry Data Security Standard, Version 4.0. https://scholar.google.com/scholar?hl=en&q=Payment Card Industry Data Security Standard, Version 4.0

[16] American Institute of Certified Public Accountants. SOC 2 Trust Services Criteria. https://scholar.google.com/scholar?hl=en&q=SOC 2 Trust Services Criteria

[17] International Organization for Standardization. ISO/IEC 27001 Information Security Management Systems. https://scholar.google.com/scholar?hl=en&q=ISO/IEC 27001 Information Security Management Systems

[18] Howard, M. and Lipner, S. The Security Development Lifecycle. Microsoft Press, 2006. https://scholar.google.com/scholar?hl=en&q=and Lipner, S