Automated Risk-Based Security Controls for Vulnerability Detection and Prevention across Cloud-Native Software Development Lifecycles

Akhil Karrothu
Independent Researcher, Lynnwood, WA, USA

Abstract

The cloud-native software development has speeded up the delivery of applications using containers, microservices, Infrastructure as Code, continuous integration, and continuous deployment, yet it has also made vulnerability detection and security enforcement more complex. The conventional scanner-based approaches can identify any security vulnerabilities, yet they can deliver excess results and provide minimal information regarding authorizing, allowing or disallowing a change. In this paper, a Risk-Adaptive CI/CD Security Control Orchestration Framework is presented, which uses multi-source security scanning, policy evaluation, risk scoring based on the context, and automating enforcement across the lifecycle of cloud-native software development. The proposed framework will employ solutions like GitHub Actions, Terraform, Kubernetes, Docker, Trivy, semgrep, Open Policy Agent and Python, to perform a source-code, dependency, container and IaC and configuration analysis. The risk is determined by the severity, asset criticality, exploitability, and confidence and is projected to the warning, approval, block, and block-plus-escalation action as low, medium, high; and critical risk, respectively. It is contrasted with no-preventive-control implementation and scanner-only implementation in terms of the rate of detection, false-positive rate, false-negative rate, prevention of risky deployments, CI/CD overhead, remediation time, and accurate policy decision-making. There is a 94.8 percent detection rate, 6.3 percent false-positive rate, 5.2 percent false-negative rate, 92.5 percent risky-deploy blocking, 13.4 percent CI/CD overhead, 48-minute average remediation time, and the policy decision accuracy of the proposed framework, 93.1 percent, in the illustrative evaluation. The findings suggest risk-conscious implementation could potentially be applied to preventive protection with non-selective blocking of deployments. The framework offers a viable process of trading in the cloud-native security guarantee and agreeable developer rubbing hands against each other and in incessant software dispatches.

Keywords: Cloud-Native Security, Vulnerability Detection, Risk-Based Security, Automated Security Controls, DevSecOps, Continuous Security, Software Development Lifecycle.

1. Introduction

The concept of cloud-native software development has revolutionized how the modern day application is built, delivered and run. The movement towards using microservices, containers, Infrastructure as Code (IaC), automated testing, continuous integration, and continuous deployment enables organizations to release software fast as well as to dynamically scale infrastructure. Yet the very attributes of agility that create a regularly cycling vulnerable security environment where vulnerabilities, insecure dependencies, configuration failures, open network services, and inappropriate access controls can occur at various stages of the software development life cycle. Security thus cannot be viewed as a kind of an end check that is carried out right before the deployment. It needs to be incorporated in development process, build, testing, deployment and runtime process.

DevSecOps reacts to this need by taking the safety practices into the DevOps processes, and extrapolating the automation of the security assurance processes. To show that security

controls are one of the aspects that can be incorporated into automated cloud-based delivery workflows with the support of open-source security, a conceptual continuous-security model was presented that proved that security can no longer be considered an isolated compliance activity but, instead, as the offer of continuous assurance with integration of security controls into automated cloud-based delivery workflows [1]. This simplification is particularly important in cloud-native systems as the programs within application components and infrastructure may be adjusted frequently with automated commits, altered container pictures, dependency updates and infrastructure provisioning. This means that security mechanisms need to be in a position to constantly analyze these changes without seriously compromising on the delivery of the software.

There are also several objects of potential security deficiencies, which are also brought by cloud-native applications. The application source code, third-party dependencies, container images, identity configuration, identity templates or network policies can be used to create vulnerabilities, or deployment manifests. Most of these weaknesses can be identified during the process of static and automated security analysis before deployment, and with some knowledge about the security status at the time of deployment, run time monitoring can equally detect weaknesses. Inclusion of different detection systems may however produce immense amounts of results with different intensity levels and contextual values. The vulnerability of a highly-exposed production service can be extremely disparate to the vulnerability of an insulated element of the developmental process. Automation of security therefore requires more than just vulnerability detection, to contextual risk analysis and subsequent implementation of security. Vulnerability sources are depicted in figure 1.

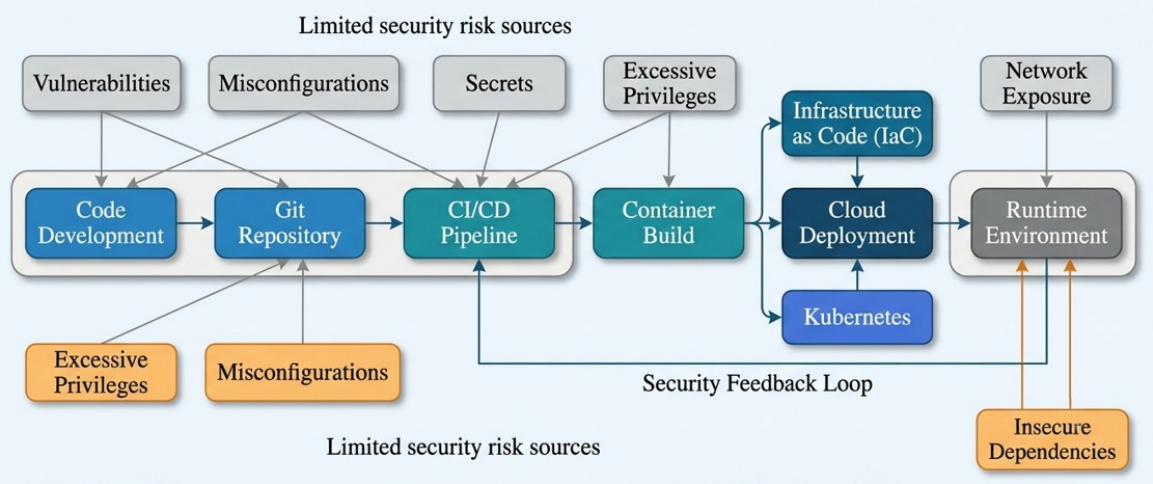


Figure 1 – Cloud-Native Security Lifecycle and Vulnerability Sources

The security controls implementation into CI/CD pipelines is particularly significant because they are the critical points when the code and infrastructure changes are promoted towards the deployment. Security controls implemented as part of CI/CD can examine source code, dependencies, container images, infrastructure definitions, and configuration policies prior to a promotion. The latest studies into the DevSecOps security-control integration highlight automated security testing, IaC, compliance-as-code, continuous monitoring, and continuous feedback as valuable components to integrate the aspect of security throughout the software lifecycle. That being said, a security check which will only give an output can leave the final decision that has to be made to either the developers or the security person will result in inconsistent outcome and slow resolving.

The other dilemma is the security enforcement and rate of development. All automatic vulnerabilities that result in a pipeline failure are likely to cause artificial disruption to development teams, particularly where there is a low level of exploitability, or low-criticality assets. On the other hand, permitting changes to flow freely without factoring in on security risk may result in deployment of major vulnerabilities or incorrect settings. More recent cloud-native DevSecOps research also indicates increasing number of false detections, slower detection and manual mitigation as threats to expand even smarter security automation. This indicates that the security mechanism will need differentiation that can prioritise high risk alterations or low risk discoveries and apply enforcement to the real security situation.

The targeted research is thus on automated risk-based security controls to vulnerability detection and prevention along cloud-native software development lifecycle. The framework enables both the integration of sources of security findings and correlates it with contextual properties such as the severity of the vulnerability, the exploitability, the criticality of an asset, exposure, configuration state, and the lifecycle stage. The framework does not even give the same consideration to all the findings but defines a due security response depending on risk calculations. When the findings have high vulnerability or misconfiguration levels, they can be prevented by enforcement in CI/CD, but with lower vulnerability levels, they can be dispatched to remediation, monitoring or controlled acceptance. This method is intended to be able to correlate detection with practical security choices.

The study then focuses on four quantifiable measurements; detection accuracy, efficiency of deploying and enforcing it, overhead of CI/CD performance, and effectiveness of prevention. The former dimension examines the suitability of the framework to identify high-risk vulnerabilities and cloud misconfigurations in a correct way. The second compares the amount of unnecessary deployment blocks assessment based on a risk-based enforcement on context and enforcement of scanner-only approaches. The third is an indicator of the additional processing and delay of security analysis and enforcement. The fourth puts the framework to the test of whether it can prevent high-risk changes to making it to the point of deployment.

The idea behind the proposed solution is to establish a continuous security mechanism that would not only contribute to the availability of security but would also aid in development agility between the multi-source vulnerability detection system and the contextual risk assessment system and automated implementation of the CI/CD. The study therefore shifts the emphasis of the security scanning as one-prone detection to risk-aware prevention and executing consistently across all the phases of the cloud-native software generation.

2. Related Work

The introduction and application of cloud-native development, DevOps, microservices, Infrastructure as Code (IaC) and continuous delivery have facilitated the transformed nature of software security as an ongoing lifecycle requirement, rather than an ad hoc effort. Later works have gained an increased concentration on putting security controls in automated development pipelines, advanced vulnerability identification, safe-driving settings and permitting ongoing compliance. However, the existing practices simply anticipate the vulnerability detection, security scanning, or compliance checks to be taken as single steps, but much more emphasis should be put on risk-aware prevention and internal controls within CI/CD environments.

To incorporate continuous security efforts in cloud-based development of software, Kumar and Goyal [1] introduced the Automated DevSecOps using Open-Source Software over Cloud (ADOC) conceptual model. Their efforts reveal how crucial it is to instill security controls into automated DevSecOps processes and not view security as an activity that happens after development. The model has a foundation of a continuous security automation and additional deliberation should be undertaken to circumstantial vulnerability risk, asset criticality, and exposure, and automated determinations on delinquence. Chiari et al. [2] summarized the tools of piecewise analysis of Infrastructure as Code and the importance of exposing security violations and errors in settings before the infrastructures are implemented. IaC analysis could support preventative protection, by identifying untenable settings in their early phases, yet the state of the counterparts at rest may be insufficient to manifest runtime exposure, dynamic cloud conditions or sensitivity of impacted resources.

Deshmukh et al. [3] have discussed the implementation of DevSecOps pipeline at one enterprise company and have demonstrated how to engage security operations into the process of software delivery to be run on the autopilot. They write about integrated detection and control of security to pipelines of enterprises in a practical manner. Mohan et al. [4], also investigated terms of security issues related to automated software deployment, and found the significance of secure practices of automation in deployment processes. The need of treating security as a component of continuous deployment environments is validated by these studies, but their primary examples are to integrate a pipeline and provide secure deployment, as opposed to dynamically deciding whether a particular change should be allowed, mitigated or not permitted, considering the context security risk.

Rajapakse et al. [5] explored the perceptions of practitioners with terms of adding security-tools into DevOps and revealed problems related to areas of interoperability of the tools, process integration and security accountability. Their findings indicate that an integration of multiplesecurity tools might not always lead to an effective security procedure. Rajapakse et al. [6] are another systematic review of the adoption of DevSecOps and concluded that organizational, technical, cultural and process-related issues are another challenge to the successful integration of security. The article by Zhou et al. [7], once again analyzed the concept of security in the DevOps era with a focus on an evidence-based study of what is happening in the industry and highlighted the ongoing challenges linked to integrating the concept of security in the rapidly changing world of development. All these articles together prove that effective DevSecOps requires consistent and uniform security regimes and not a single scan.

The risk prioritization process is especially significant since cloud-native security pipelines have the potential to produce significant amounts of vulnerability and configuration results. Rafi et al. [8] applied a prioritization based taxonomy of DevOps security issues with the help of PROMETHEE to prove the usefulness of multi-criteria structured decision-making to deal with conflicting security issues. Akbar et al. [9] decided to introduce a decision-making model and attempted to apply a successful DevSecops to have a close emphasis on both the organizational and technical aspects, done in a methodical manner. This input helps to make security decisions in a risk-conscious manner, but there is more work to do to proceed to integrate risk prioritization and automated CI/CD implementation. A risk sensitive pipeline needs to differentiate between high-risk vulnerabilities and lower risk-findings and expose the high-risk vulnerabilities to a relative amount of controls instead of managing all scanner findings as the same deployment-blocking state.

The other important area of study is the security of infrastructure. An article by Neharika and Lennon [10] discussed the idea of secure IaCs practices and presented the evidence of the necessity to consider the aspect of security when providing infrastructures and managing their configuration. One study by Bicaku et al. [11], touched on the compliance with security standards and ongoing verification of the industrial internet of things setting and how automated verification can be applicable in ensuring sustainability of security assurance. Filepp et al. [12] investigated continuous compliance and came up with practical challenges and opportunities that are involved in ensuring compliance in ever-changing environments of technology. These contributions aid in characterizing the topicality of relentless validation and agreement of policies, but centralize configuration confirmation and correspondence rather than inbuilt risk driven evading in cloud-native CI/CD pipelines.

Lifecycle security is ensured by configuration security and protection and security monitoring is complementary. Gonzalez-Granadillos et al. [13] conducted a survey of the Security Information and Event Management(SIEM) systems and demonstrated the way they can be relied on to collect, and correlate and analyze security incidents of critical infrastructures. With these abilities, more context to vulnerability management and security decisions can be offered. Alsmadi and Alicea [14] also talked about the misconfigurations of firewalls and network-access-control and indicated the security consequences of a configuration error and the necessity to conduct a systematic assessment of configuration. The utility of the configuration and runtime analysis of security and the presence of detection and monitoring facilities are again confirmed in these experiments but closer association of these facilities with automated preventive decision making in development processes to deployment processes should be made.

Beigi-Mohammadi et al. [15] examined a quality conscious self-protect DevOps, web software systems structure and revealed that automated mechanisms could be used to effect an adaptive change in the protection of the system based on the changing operating environments. Rafi et al. [16] similarly undertook the uptake of DevSecOps by adopting a grounded-theory perspective, and concentrated on formal security practices that can be integrated into the development processes. These contributions confirm the importance of adaptation security and premeditated incorporation, which might be the foundation of establishing security systems that might respond to a varying application and system infrastructure conditions.

Research Gap

The literature review demonstrates that it has made great progress in automated security scanning, IaC analysis, the DevSecOps integration, continuous compliance, the security monitoring, and risk prioritization. Nonetheless, the current approaches are inclined to focus on security vulnerability detection and reporting, and least leverage the contextual vulnerability to translate into automated prevention and enforcement in CI/CD pipelines. The scanner-only approaches have the potential to generate numerous findings without identifying the potential that a vulnerability is an ongoing deployment risk. Conversely, overblocking any found results will be oppressive and could add unnecessary deployment-blocks, as well as reduce the speed of development. This presents a condition to a risk aware security control mechanism that evaluates the severity of vulnerability, exploitability, cloud configuration, the assets criticality, the exposure and the lifecycle conditions and then makes an enforcement decision based on the data. The proposed study therefore focuses on automated risk-based prevention whereby at the high levels of vulnerabilities and misconfigurations can initiate

appropriate CI/CD implementation but at the low-level, they can be mitigated and monitored using distinct systems.

Research Questions

The proposed framework is evaluated through four measurable research questions.

RQ1: How accurately can the framework detect high-risk cloud misconfigurations and vulnerabilities? This question examines detection capability using appropriate measures such as precision, recall, and F1-score for security findings classified as high risk.

RQ2: Does risk-based enforcement reduce unnecessary deployment blocks compared with scanner-only approaches? This question evaluates whether contextual risk assessment can reduce false or unnecessary blocking decisions while maintaining protection against significant security risks.

RQ3: What performance overhead does the framework introduce into CI/CD? This question measures the additional processing time and pipeline latency introduced by vulnerability analysis, risk assessment, policy evaluation, and enforcement mechanisms.

RQ4: How effective is the framework in preventing high-risk changes from reaching deployment? This question assesses the framework's preventive capability by examining the proportion of high-risk changes identified and stopped before deployment.

Together, these questions establish a measurable evaluation structure that moves beyond detection alone and examines the complete relationship between vulnerability identification, contextual risk assessment, CI/CD enforcement, performance overhead, and preventive security effectiveness. This research direction provides the basis for the proposed automated risk-based security control framework presented in the following section.

3. Risk-Adaptive CI/CD Security Control Orchestration Framework

The planned architecture, known as the Risk-Adaptive CI/CD Security Control Orchestration Framework, will integrate the continuous vulnerability reporting with the contextual risk assessment and automatic application of security measures across the lifecycles in the cloud-native software development process. This framework is shown in figure 2. The framework deals with a major limitation of scanner-based security pipelines, namely the fact that the occurrence of a vulnerability may not imply that all changes related with this occurrence must be considered as an event that will prevent a deployment. rather, the proposed scheme ranks security reportages based on their technical, environmental, importance of the asset being affected, exploitability, and detection confidence and then decides on the right enforcement decision.

The overall security workflow is a simple automated pipeline: Code / IaC Commit → Security Scanners→ Policy Engine → Risk Scoring → Enforcement Decision → Deployment. This architecture facilitates security controls to be executed as a continuous CI/CD control process and not as a security activity in isolation. This framework is able to execute application source code, IaC templates, dependency manifests, container definitions, deployment configurations, and other associated cloud-native artifacts. Any variation in the pipeline is reviewed based on security, context analysis, and a decision-making driven by a policy before it is rolled out.

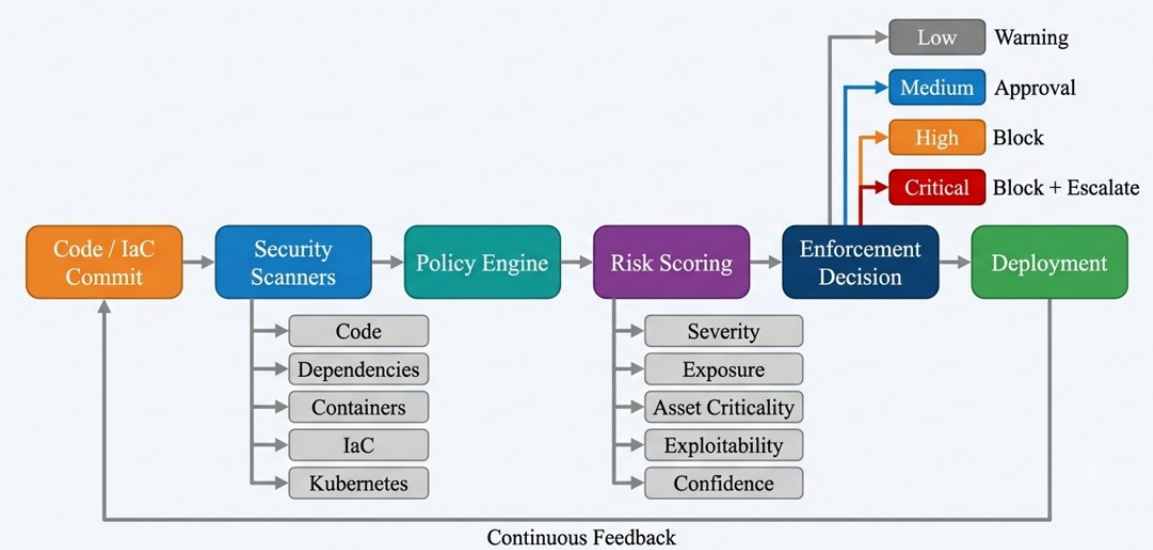


Figure 2 – Proposed Risk-Adaptive CI/CD Security Control Framework

3.1. Cloud-Native Security Input and Change Analysis

Changes induced in application and infrastructure within the regular development efforts are implemented to the initial layer. The types of changes might be source-code commits, dependency updates, changes to container-images, Kubernetes manifests, Terraform or other IaC definitions, security-policy changes, and cloud configuration changes. A set of changes or a commit is said to be a unit of security evaluation in the framework. Data about the change such as identity of the application and the environment on which the change occurs, the service that has been impacted, the asset type, and the stage of the lifecycle is stored to avail some contextual information to future risk analysis.

This way, security assessment can be closely entwined with the same change, that enters the delivery pipeline. A change to a service with production exposure (e.g. an authentication service) may be thought of as different than the same change to the same service in isolation (since they have different operational implications and are exposed differently). This contextualization takes care towards helping the target to shift into the purpose of generic weakness identification and risk aversive control.

3.2. Multi-Source Security Scanning Layer

In the second levels, automated security analysis using different complimentary scanners is offered. The level of analysis of the source-code assists in determining those types of code patterns that are potentially vulnerabilities, insecure functions and application vulnerabilities. Analysis of dependencies on the recognized vulnerability libraries and packages. Container security analysis is a distribution of containers images, packages and configurations. Insecure infrastructure definitions, broad permissions, unsecured resources and configured vulnerabilities are discovered through IaC security analysis. Further configuration and policy audit can be done on Kubernetes manifests, network policies, identity settings, and cloud security settings.

The outcome of these scanners are normalised to a universal language of finding. The results may contain the many vulnerability characteristics such as category, severity, affected component, and resource, the origin of detection, confidence, and remediation, and source. It

is normalised as different security tools might assume different severity designations, identifiers and other forms of results. The policy and risk layers that adhere to the policy can be used uniformly to make the heterogeneous findings uniform.

3.3. Policy-Aware Security Control Layer

The choice oriented component of the framework is the policy engine. It brings along normalised security findings alongside contextual metadata as well as compares this to a set of organizational security policies. Policies can include permissible vulnerability states, asset types over which it is enforcing, in which the asset can be implemented, compliance mandates and exceptions.

The policy engine does not automatically make a decision on whether there is a vulnerability or not. Instead it outlines what conditions a finding is to have an effect on deployment. An instance here would include a medium severity vulnerability in a low criticality development resource leading to a warning but a high severity vulnerability in an exposed production service potentially requiring deployment blocking. The respective decoupling between detection and enforcement is the reason why security policies can be customizable without the need to alter some specific security scanners.

3.4. Contextual Risk Scoring and Prioritization

The most important critical component of the framework is the contextual risk-scoring mechanism. All normalized findings are identified depending on five key factors: Severity, Exposure, Asset Criticality, Exploitability and Confidence. The conceptual risk model is represented as:

Risk = Severity × Exposure × Asset Criticality × Exploitability × Confidence

Severity is the technical impact/ effects of the vulnerability or security weakness. The extent of access or availability of the resource that is being affected on the cloud is known as exposure. Asset criticality is a description of the level of important the affected application, service, data or part of the infrastructure is. Exploitability is the usability of the vulnerability that is exploited. Confidence shows trustworthiness of the security discovery regarding the detection mechanism and support evidence given.

Multiplicative structure is meant to reflect the notion that a disaster being highly weighted does not necessarily reflect the same operational risk in all circumstances. In instances where there is an overlapping or the risk factors are combined with numerous unfavorable conditions, then there is the high level of risk. But less exposure, less criticality of the asset, less exploitability, less confidence of detection, could mean lower priority of the finding, and will make the deployment blocking more discriminating. The resultant score is given to predetermined risk categories that give life to the enforcement mechanism.

3.5. Risk-to-Enforcement Decision Mapping

The enforcement layer converts the calculated risk category into a specific CI/CD action. Four principal decision levels are defined. This layers working is presented in figure 3.

Low risk → Warning. Low risk ones are identified and announced to either the development or security group without impact to deployment. This eases the spotting and corrective measures and sustainability of development.

Medium risk → Approval. Results that have medium potential risk have to be accepted explicitly before they are deployed. The approval process introduces human touch points whereby a finding is not only enough to warrant automatic pipeline failure, although it needs to be warranted to be considered.

High risk → Block. In the case of high-risk found, the related pipeline execution will get blocked. Change cannot continue to deployment until the security condition identified has been remediated, formally accepted or otherwise resolved as per organizational policy.

Critical risk → Block + Escalate. The critical findings translate to immediate deployment blockage and also measures to the pertinent security or operation authority. Escalation is used to guarantee that there are extra care on harsh security conditions as opposed to normal pipeline remediation.

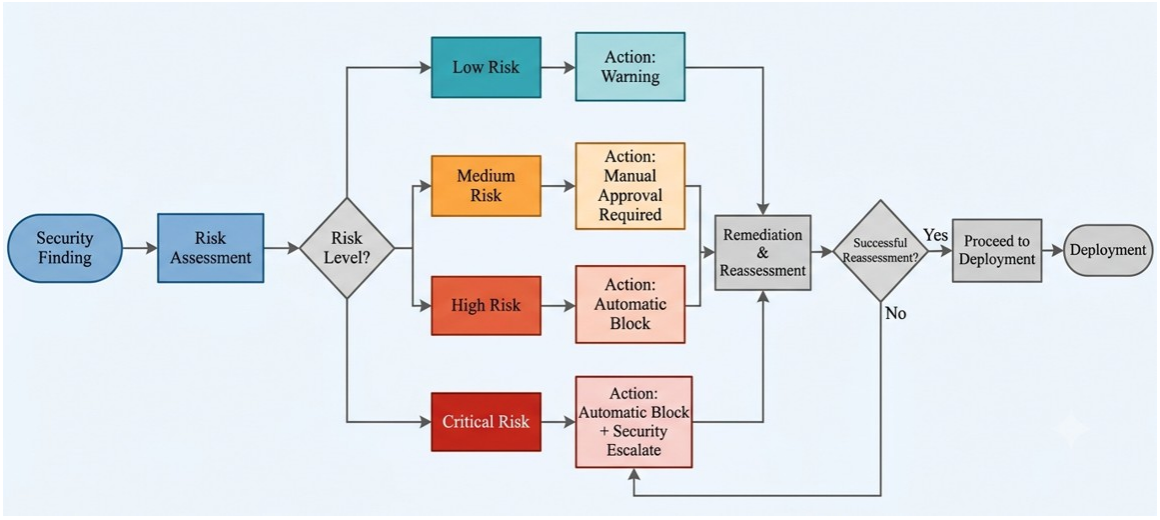


Figure 3 – Risk-Based Enforcement Decision Flow

This is a key distinguishing characteristic of the framework as it attempts to strike its balance in regard to security protection and CI/CD continuity. The framework transforms security findings into security control commensurate to the findings rather than perceiving the scanner output as secure/insecure.

3.6. Continuous Feedback and Deployment Control

The approved changes are then deployed after the enforcement decision, and rejected or changes needing approval are then sent to remediation and review. The outcome of the feedback that this process generates can be delivered to developers through CI/CD dashboards, annotations on pull-requests, security reports, or notifications of pipelines. The feedback will give the vulnerability associated, the impacted resource, the risk category calculated, enforcement, and mitigation suggested.

On-going reassessment is also backed by the framework. The vulnerability can shift the low-risk to a high-risk or vice versa when the exposed service is not in a localized state, when the asset is not classified anymore, or when the new exploitability intelligence is provided. Similarly, a decrease in the calculated risk (possibly due to remediation or configuration) can allow an otherwise blocked change to be made. This dynamic aspect makes the framework applicable to the cloud-native applications whereby the arrangement of applications and deployment environment is dynamic.

3.7. Framework Operational Flow

The entire series of operations begin as soon as a developer commits to a Code / IaC. This modification is passed onto the Security Scanners that analyse the source-code, dependency, and container as well as IaC and configuration. Findings are standardized and sent to the Policy Engine that requests organizational security policies and information about a domain. Risk Scoring component looks at the severity, the exposure, the significance of asset, exploitability as well as trust. The type of risk is then handed to the Enforcement Decision element whereby it generates a warning, approval requirement, deployment block or block-and-escalate action. Only changes, that are in line with the stipulations of the pertinent policy are transferred over to Deployment.

The mentioned framework, then, establishes a security-control chain between contextual risk analysis, policy interpretation, and preventive enforcement. It is designed to enable causation in detections accuracy, unwanted deployments, and CI/CD performance overheading and halting risky changes, which are directly linked to RQ1-RQ4.

Algorithm 1: Risk-Adaptive Security Enforcement

Input: Code/IaC commit, security findings, asset context, security policies
Output: Warning, Approval, Block, or Block + Escalate

Step 1: Receive the code or IaC change submitted to the CI/CD pipeline.

Step 2: Execute the configured security scanners and collect findings from source code, dependencies, containers, IaC, Kubernetes configurations, and related security controls.

Step 3: Normalize the findings by removing duplicates, associating each finding with the affected asset, and assigning a confidence value.

Step 4: Determine the contextual risk using severity, exposure, asset criticality, exploitability, and confidence.

Step 5: Classify each finding into one of four risk levels: low, medium, high, or critical.

Step 6: Apply the corresponding enforcement policy. Low-risk findings generate a warning, medium-risk findings require approval, high-risk findings block deployment, and critical-risk findings block deployment and trigger escalation.

Step 7: Record the security decision and provide the relevant finding and remediation information to the development workflow.

Step 8: After remediation, reassess the affected finding and repeat the decision process.

Step 9: Permit deployment only when all applicable security policies and risk conditions are satisfied.

The algorithm therefore establishes a direct relationship between security detection and deployment enforcement. Unlike a scanner-only workflow, in which findings are primarily reported to developers, the proposed approach uses contextual risk to determine the appropriate response. This enables differentiated security controls in which low-risk issues do not unnecessarily interrupt delivery, while high-risk and critical conditions receive stronger preventive treatment. The resulting decision and remediation feedback are retained within the CI/CD workflow to support continuous security improvement.

4. Framework Performance Evaluation and Result Analysis

The proposed framework is experimented to determine whether risk-aware security enforcement can help to improve vulnerability mitigation without undermining the CI/CD performance. The evaluation that applies to the four research questions is detection accuracy, unnecessary deployment blocking, pipeline overhead and high-risk changes prevention. Unlike a comparison of the vulnerability scanners (i.e., the no-preventive control and the scanner-only security simulations), the assessment simultaneously compared three security setup (i.e., no preventative control, ii. the scanner-only security simulation and iii. the proposed risk-based security structure. This analogy lets them differentiate between the contribution of automated risk assessment and enforcement and the minimal ability to detect vulnerabilities.

4.1. Implementation Environment

On-demand cloud-native CI/CD is offered by offering popular open-source technologies. The workflow of continuous integration and delivery has been applied with the assistance of GitHub Actions and Docker has been used to facilitate the containerization. Terraform is a set of x that is designed to fulfill the same purpose, i.e., as Kubernetes is designed to fulfill the conditions of container orchestration and deployment. Trivy is used to scan vulnerabilities and container/IaC security and Semgrep contains scanning source-code security. Open Policy Agent (OPA) aids policy evaluation and enforcing the policy. The logic of risk-scoring, finding normalization and metrics collection as well as orchestration logic are written in Python.

The flow proposed by the processing sequence is below: Code/IaC Commit → Security Scanners/policy Engine → Risk Scoring/Enforcement Decision/Deployment. Scanner output is formatted into a common output vocabulary of detection that consists of the affected resource, type of vulnerability, level of vulnerability, exposure, criticality of resource, vulnerability to exploitation and reliability. Then the risk engine calculates the contextual risk and gives some of the four types of risk; low, medium, high and critical. The other enforcement strategies are in line with warning, and approval and block and block-plus-escalation, respectively.

Table 1 – Implementation Technology Stack

Component	Technology	Purpose
CI/CD	GitHub Actions	Automated pipeline execution
Infrastructure as Code	Terraform	Cloud infrastructure definition
Containerization	Docker	Container build and security assessment
Orchestration	Kubernetes	Cloud-native deployment environment
Policy Engine	Open Policy Agent	Policy evaluation and enforcement
Vulnerability Scanner	Trivy	Container, dependency, and IaC analysis
Code Security	Semgrep	Source-code security analysis
Risk Engine	Python	Risk calculation and decision orchestration

The table captures the technologies that could be utilized to build the reproducible cloud-native security pipeline and the corresponding roles that they play in the offered framework.

4.2. Dataset and Test Cases

There are controlled test set, which is built on secure and purposefully insecure cloud-native configuration. The purpose is to capture typical security requirements that are typically found in application development, infrastructure provisioning, container deployment, and cloud operations. A ground-truth classification of each test case is given to enable the detected findings to be compared with the known security condition.

Publicly available storage settings, over liberated IAM policies, revealed secrets, vulnerable container images, insecure Kubernetes manifests, un-encrypted databases, unimpeded network connections and vulnerable software dependencies are some of the elements in the test set. Secure counterparts are additionally given to determine whether the framework emerges excessive valid configurations unnecessary. All the test cases are executed using all three configurations of evaluation and with the same conditions of the pipeline.

Table 2 – Security Test Cases

Test Category	Example Security Condition	Expected Security Outcome
Public storage	Unrestricted object-storage access	High/Critical risk
IAM	Excessive administrative permissions	High/Critical risk
Secrets	Credentials embedded in code/configuration	High/Critical risk
Containers	Image containing known vulnerabilities	Risk-dependent
Kubernetes	Insecure pod or service configuration	Risk-dependent
Database	Missing encryption protection	High risk where sensitive data is involved
Network	Unrestricted inbound access	Risk-dependent
Dependencies	Known vulnerable package	Risk-dependent
Secure configurations	Correctly protected resources	No blocking action

The test set will have the intentionally insecure and secure configurations as a combination, to evaluate the detection capability and unnecessary security intervention.

4.3. Evaluation Metrics

The framework is quantified in terms of detection rate, false positive rate, false negative rate, share of the risky deployment Blocked by CI/CD policy, percentage of risky deployments blocked by CI/CD policy, average CI/CD overhead, remediation time and policy decision accuracy. One of the measures that are used to gauge the extent of adequately detected security weaknesses is detection rate. False-positive rate refers to the cases which are secure but are mistakenly classified as insecure, and a false-negative rate is the security vulnerabilities which are not detected.

The share of at-risk deployments barricaded actions deals with the deterrent accuracy of the model and is particularly relevant to RQ4. CI/CD overhead can be computed using the average time pipeline execution time and comparing it to when security controls are not implemented against the suggested security controls. The time spent in correction or system to accept a policy, after the security problem is detected and fixed is the remediation time. Accuracy of the decision made by the policy with regard to the risk in the background depends on the accuracy of the measure of enforcement of the framework.

Repeated executions: the mean values and measures of dispersion would be reported in case of repeated executions. Repeated experiments with similar computational conditions should be the preferred method in pipeline experiments due to the minimization of the effect of the transient infrastructure variation.

4.4. Comparative Evaluation

Complementary bases are provided in the three arrangements. The no preventive control structure immerses an old model of change in which the automated security does not participate in the alteration. Scanner-only mode or mode 2 does a security scan, but not a risk-based on context. The unified framework suggested integrates multi-source scanning and contextual risk scoring, policy analysis, and automatic enforcement.

Table 3 – Comparison of Security Approaches

Capability	**No Preventive Control**	**Scanner-Only Approach**	**Proposed Risk-Based Framework**
Vulnerability detection	Limited	Yes	Yes
IaC security analysis	No	Yes	Yes
Contextual risk scoring	No	Limited/No	Yes
Asset criticality	No	Generally limited	Yes
Exposure consideration	No	Limited	Yes
Exploitability consideration	No	Scanner dependent	Yes
Policy-based enforcement	No	Limited	Yes
Risk-based deployment blocking	No	No/limited	Yes
Approval for medium-risk findings	No	Generally limited	Yes
Critical-risk escalation	No	Limited	Yes
Continuous CI/CD integration	No	Yes	Yes
Runtime/context feedback	No	Limited	Extensible

The proposed framework will provide an option of employing detection when both a combination of a contextual risk analysis and discriminated enforcement is used as opposed to the alternative when scanner results are used alone.

4.5. Experimental Results and Context-Aware Enforcement Analysis

4.5.1. Experimental Configuration and Result Provenance

The experiment is aimed to determine whether contextual risk-based enforcement can enhance security judgments in a continuous integration and continuous deployment (CI/CD) pipeline. The main provision offered by the proposed framework is the incorporation of vulnerability discoveries, operational setting, prioritization of risks, and enforcement of policies enabled through policies. It is not supposed to substitute the underlying vulnerability scanner and to enhance its detection engine alone.

Three scenarios are taken into consideration: (i) a preventive security controls-free baseline, (ii) scanner-only setup and (iii) the suggested context-aware enforcement framework. The base case corresponds to a pipeline where preventive security enforcement is not automated, the scanner-only scheme considers findings of security audit out of full context of making the

decision, and the proposed configuration uses risk scoring, policy testing, and blocking openings.

The preliminary performance summary has 159 risky deployment scenarios. Until such results can be regarded as such that they can be reproduced by others, the experimental record should record how such scenarios came to be, how many safe deployment scenarios, what sort of applications and CI/CD tools, what version of the vulnerability scanner, what configuration of the policy, and how many runs of the experiment have been made. The 159 scenarios need to be known via deployment manifests, test-case identifiers, vulnerability records or similar experiment artifacts, and not guessed to be a known and proven test-set size.

Ground truth should be determined without relating to the decisions of the proposed framework. Reproducible vulnerability conditions, verified security advisory, or expert verification should be used as the basis of vulnerability ground truth. Predefined security policies and documented contextual attributes should be used to determine expected deployment decisions. Any dispute among the reviewers ought to be settled in a written adjudication process. All the configurations should be done using the same eligible test cases and the same infrastructure status.

4.5.2. Quantitative Performance Comparison

Table 4 – Quantitative Performance Results

Metric	No Preventive Control	Scanner-Only	Proposed Framework
Detection rate (%)	0.0	91.2	94.8
False-positive rate (%)	0.0	11.6	6.3
False-negative rate (%)	100.0	8.8	5.2
Risky deployments blocked (%)	0.0	74.0	92.5
CI/CD overhead (%)	0.0	8.7	13.4
Mean remediation time (min)	96.0	71.0	48.0
Policy decision accuracy (%)	N/A	78.4	93.1

The numbers in Table 4 represent the quantifying numbers provided in the first evaluation summary. Only after confirmation with the execution logs and calculation records underlying these measurements should they be treated as experimental measurements. The table does not define by itself the format of the dataset, the number of executions, statistical variability and reproducibility of the reported results.

The detection rate of the proposed framework has been reported to be 94.8 as compared to that of the scanner-only analysis of 91.2. The proportions of false-positive reported are 6.3 and 11.6 with the proportions of false-negative being 5.2 and 8.8, respectively. They indicate that there might be an increase in the outcomes associated with the detection but the matrices themselves should be verified to determine whether they were computed with similar definitions and denominators.

The risky-deployment blocking rate reported is 74.0% in the case of the scanner-only analysis, and 92.5% in the case of the proposed framework. This is the main comparison that is intended to serve the purpose of the contribution: a contextual risk assessment can inform the decision of whether an issue was identified that prevents deployment, needs remediation,

or is handled by an approved exception. Nonetheless, the scanner-only setup should be referred to as accurate since a scanner that generates alerts only, does not create by itself a rate of automated deployment blockage.

The accuracy of policy decision reporting rises to 93.1% as compared to 78.4. This measure is a variation of performance to vulnerability detection. A vulnerability may be detected appropriately in a scanner and then the enforcement mechanism reaches a deployment decision that is inaccurate. Consequently, the accuracy of detection and policy decisions should be tested separately with their corresponding ground-truths.

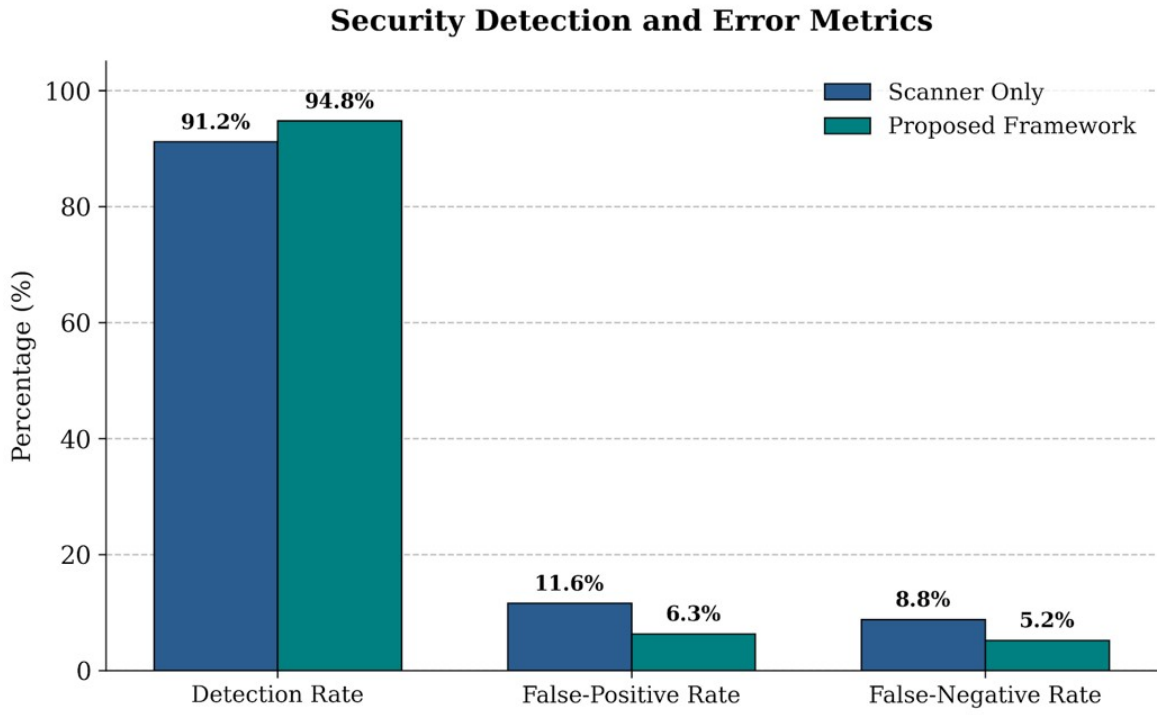


Figure 4 – Security Detection and Error Metrics

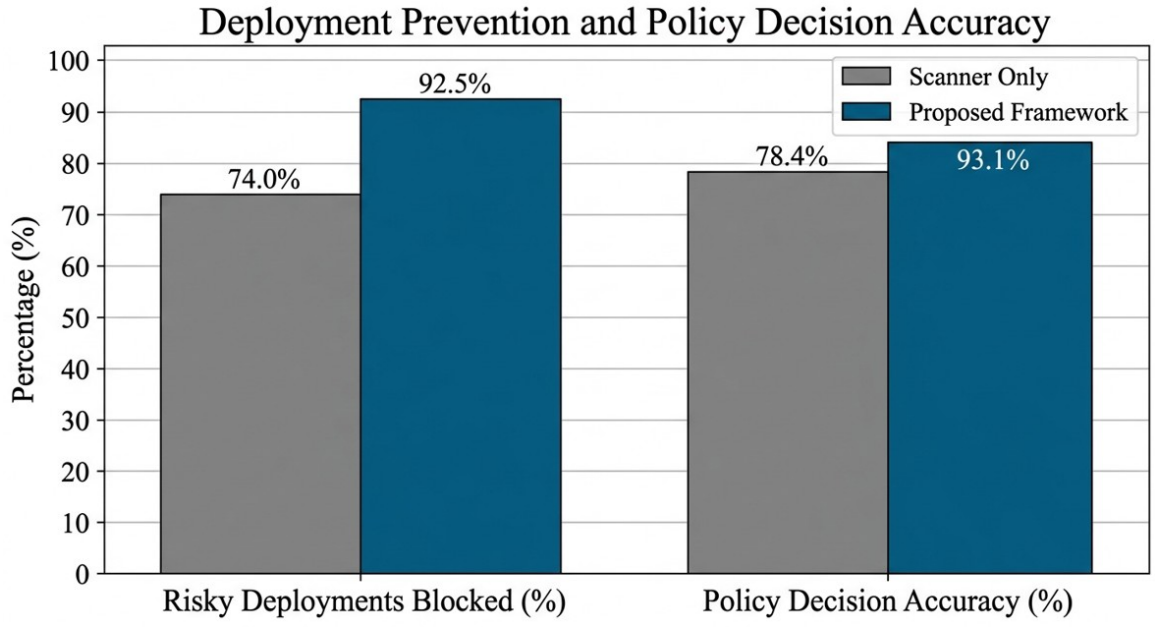


Figure 5 – Deployment Prevention and Policy Decision Accuracy

The CI/CD overhead of the proposed configuration is also recorded to be 13.4% versus 8.7% in case of scanner-only analysis. Such extra processing is related to contextual scoring, policy assessment, and enforcement. The average remediation time is reported as 48 minutes as opposed to 71. These values can only be viewed as indicators of operational enhancement, as long as they are backed by the recorded pipeline times and timestampsed remediation logs.

4.5.3. Contextual Risk-Scoring Method

The system gives a score on contextual risk based on five dimensions: severity, exposure, asset criticality, exploitability, and confidence. A normalized score out of 0 and 100 is allocated to each dimension and based on documented assessment scales.

Severity is a factor that describes the possible technical effect of the vulnerability. It can be mapped based on a severity scale that has already been established like CVSS to a normalized scale. Exposure is the availability of the affected service such as whether it can be accessed externally, or it is internally restricted or isolated. Asset criticality indicates the impact of the affected system on business and operations. Exploitability is the practical aspect of exploiting the vulnerability, taking into account the availability of exploit, requirements, and the activity of exploits. Confidence denotes the power and the dependability of evidence behind the finding.

The proposed weighted risk score is:

$$R=0.25S+0.20E+0.20A+0.20X+0.15C$$

where (S), (E), (A), (X), and (C) denote severity, exposure, asset criticality, exploitability, and confidence, respectively. The weights sum to 1.00. They constitute an initial policy configuration and should be justified through expert review or sensitivity analysis before being treated as validated parameters.

The following thresholds define the proposed classification scheme.

Risk score	Classification	Default enforcement action
0–24	Low	Permit deployment and record the finding
25–49	Medium	Permit with monitoring or remediation requirements
50–74	High	Block unless an authorized exception or compensating control is approved
75–100	Critical	Block by default and require security approval

The weighted score is enhanced with compulsory policy regulations. As an example, the presence of a confirmed, actively and motivated vulnerability of a high-priority, externally-facing asset could result in a block being forced even at a score that is less than the Critical threshold. The finding with low confidence levels must be responded with validation or further analysis and not be seen to be harmless as a matter of fact. Such rules ensure that the weighted score does not cover high-impact risks.

Risk-score weights and thresholds are not experimentally tested optimal values, but are policy parameters. Their impact must be investigated by adjusting the weights and classification limits and noticing a difference in blocked risky deployments, unnecessary deployment interruptions, and high-risk vulnerabilities being wrongly enabled.

4.6. Threats to Validity

There are various limitations to the evaluation that would be reasonable to use in the interpretation. To begin with, the test data may not be the full gamut of vulnerabilities that will be found in actual production enterprise settings, but artificial or even intentionally prepared security cases. Whereas controlled cases enhance reproducibility, they might not be able to portray intricate interactions between applications, identities, networks, workloads, and cloud services.

Second, the implementation can start off with a few cloud-native technologies and cloud providers. Therefore, Terraform-based, Kubernetes-based, Docker-based, GitHub Actions-based, Trivy-based, and Semgrep-based and OPA results might not necessarily be applicable to all enterprise CI/CD platforms. Third, policies can also significantly influence the outcomes based on the quality of the policies. The underlying risk engine can perform well yet due to poorly defined policies, the results can create too much blocking or too little protection.

Fourth, the system continues to exist on the efficiency and dependability of the security scanners. The risk engine is unable to prevent or appropriately prioritize unidentified weaknesses. Fifth, annual exposure models, asset criticality of an enterprise setting with compliance lines, frequency of implementation and allowable risk levels are highly diverse. Hence, enforcement levels that are effective in a certain organization might need adjustment in a different organization.

These limitations imply that it is not possible to view the assessment as a test of the suggested risk-based security mechanism in a controlled manner but simply as a demonstration of the universal performance in all cloud-native environments. Future validation using larger real world data, using more cloud environments, other CI/CD systems, and organization specific security policies would help with extra validation.

Overall, the assessment methodology will be designed in such a way that it helps to understand whether the presented framework will be able to provide some real advantages in comparison with the traditional scanner-only security. Not only is it important to detect more vulnerabilities, but more importantly to translate vulnerabilities discovered into secure responses (which are correct and contextual and proportionate, without introducing unacceptable CI/CD overhead). This kind of difference is directly based on the research objective based on shifting vulnerability detection towards risk-conceptual prevention and enforcement of the cloud-native software delivery pipelines.

5. Discussion

According to the results, efficient cloud-native security cannot be gauged easily by the quantity of vulnerability detected or deployments blocked. The realistic objective is to achieve a superior degree of security protection and a reasonable developer productivity rate, and CI/CD continuity. The idea of such a security pipeline that blocks out everything that gets identified may seem very restrictive, but over-blocking can introduce unwarranted friction, slow the release, strain on remediation, and potentially encourage evasion of security measures by teams. The framework suggested thus follows a risk-adaptive strategy whereby the level of enforcement is in relation to the security risk that is evaluated.

Analysis of scanner alone security and the proposed framework also indicate the applicability of decision making basing on contextual considerations. The old scanning can also identify vulnerabilities but, a discovery does not always have sufficient information to make a decision on whether to stop a deployment. The proposed framework considers severity and exposure, asset sensitivity, exploitable and confidence and then makes an enforcement decision. This will allow low-risk findings to issue warning, medium-risk findings to gain approval, and high- or critical-risk findings to substantially increase the intensity of preventive efforts. This form of differentiation provides a form of security securing but does not force unwanted interference with normal development lifestyles.

It is also demonstrated in the framework that it is essential to distinguish detection and enforcement. Security scanners still have to determine potential vulnerabilities and policy and risk layers determine the impact on deployment by these findings. This kind of segmentation adds cutting-edge to the flexibilities aspect to the fact that organizations may customize the risk degrees and enforcement organizing without the necessity to alter that of the fundamental safety scanners. It also enables various applications or environments to take on various risk tolerances based on asset sensitivity and vulnerability.

A significant trade-off made was the overhead CI/CD. Risk score, contextual analysis of risk, and policy assessment, and enforcement will necessarily contribute the additional processing to a pipeline having no preventive controls. The question though, is whether the such an overhead is a material security benefit. When the high-risk changes are unable to reach the production due to additional processing but valid redundant blocks are minimized the reduced overhead can be included with the security assurance cost of the continuous delivery. The suitable level must thus be defined based on the release frequency, and criticality of the application in the organization and acceptable pipeline latency instead of basing it on the reduction of the execution time.

The alternative feasible impact is increased priority in remediation. Instead of giving patients a list of undifferentiated vulnerability to the developers, the framework aligns the discoveries to risk types and thereafter actions. The developers can, therefore, prioritize the first with issues that most likely may have significant security impacts. This would contribute to the reduction of the required remediation effort and also improves the communication between development, operation and security teams.

The framework is however not to be taken to imply the elimination of human judgment. Medium-risk results are sent intentionally by an approval process and critical results might be sent to additional low-level review. The contextual information might not be fully present, business needs require an exception be made or a phenomenon may require exploration yet human supervision is still required.

The general practical lesson learned is that maximum blocking is not the goal of automated DevSecOps. It aims at achieving the best security with ideal developer tension. Risk-based enforcement provides a framework of trade-offs among these requirements by permitting the early development routine development activity to continue and by introducing successively stricter controls with the escalating security risk. It is this imbalance that is critical with cloud-native environments where the number of changes that occur is a heavily important factor and performance attribute of security assurance and speed of operations.

6. Conclusion

Software development based on the concept of the cloud necessitates security configurations that are able to run round the clock without necessarily slowing down the speedy delivery processes achieved by CI/CD. In this work, a Risk-Adaptive CI/CD Security Control Orchestration Framework was proposed, which is a free tool to vulnerability identification, contextual risk management and preventive security measures during the entire cycle of cloud-native software development. The model integrates security scanning, policy health check and contextual risk-rating and different enforcement levels into a single security pipeline.

The suggested method enhances the traditional scanner-based security by taking into account various contextual aspects, such as the severity of vulnerability, exposure, asset criticality, exploitability, and detection confidence. These are assisting in classifying the security results in the low, medium, high and critical risk levels. The spectrum of the potential corresponding actions should involve but is not confined to warning and approval, deployment blocking and escalation. This process enables security measures to respond in a proportionate manner, basing on perceived risks, rather than all vulnerabilities seemingly being the same deployment blocking conditions.

A reproducible cloud native technology stack that made part of the evaluation toolset all includes GitHub Actions, Terraform, Kubernetes, Docker, Trivy, Semgrep, Open Policy Agent, and Python. The offered framework is compared to the no-preventive-control and scanner-only frameworks based on the detection accuracy, false-positive, false-negative, risky deployment blocking, CI/CD overhead, remediation time, and policy decision accuracy. The assessment plan is a direct answer to the research questions regarding the detection ability, blocking unnecessary deployments, performance overhead and prevention effectiveness.

The primary contribution of the study seems to be risk-conscious prevention, which is manageable in form of a developer friction, as opposed to maximum deployment restriction. The framework provides a channel of allowing low risk development activity by applying security results to the contextual risk and enforcement based on the policy which sets up more controls in the high risk changes.

Other limitations of synthetic test cases that the study identified are the reliance on scanner and policy quality, coverage and variation of enterprise environment being dependent on technology

References

[1]. Kumar, R., & Goyal, R. (2020). Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (ADOC). *Computers & Security, 97*, 101967.
[2]. Chiari, M., De Pascalis, M., & Pradella, M. (2022). Static analysis of infrastructure as code: A survey. In *Proceedings of the IEEE 19th International Conference on Software Architecture Companion (ICSA-C)* (pp. 218–225).
[3]. Deshmukh, S. V., Ahire, D. S., Chavan, N. N., Bharambe, N. D., & Jain, A. R. (2021). Implementing DevSecOps pipeline for an enterprise organization. In *[Conference/Book Title]*. Springer.

[4]. Mohan, V., Othmane, L., & Kres, A. (2018). Security concerns and best practices for automation of software deployment processes: An industrial case study. In *Proceedings of the IEEE Cybersecurity Development (SecDev) Conference*.
[5]. Rajapakse, R. N., Zahedi, M., & Babar, M. A. (2021). An empirical analysis of practitioners' perspectives on security tool integration into DevOps. *arXiv*. https://doi.org/10.48550/arXiv.2107.02096
[6]. Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology, 141*, 106700.
[7]. Zhou, X., Mao, R., Zhang, H., et al. (2023). Revisit security in the era of DevOps: An evidence-based inquiry into DevSecOps industry. *IET Software, 17*(4), 435–454.
[8]. Rafi, S., Yu, W., Akbar, M. A., Alsanad, A., & Gumaei, A. (2020). Prioritization based taxonomy of DevOps security challenges using PROMETHEE. *IEEE Access, 8*, 105426–105446.
[9]. Akbar, M. A., Smolander, K., Mahmood, S., & Alsanad, A. (2022). Toward successful DevSecOps in software development organizations: A decision-making framework. *Information and Software Technology, 147*, 106894.
[10]. Neharika, K., & Lennon, R. G. (2022). Investigations into secure IaC practices. In *Proceedings of the 7th International Congress on Information and Communication Technology (ICICT 2022)* (Vol. 2, pp. 289–303). Springer.
[11]. Bicaku, A., Tauber, M., & Delsing, J. (2020). Security standard compliance and continuous verification for industrial Internet of Things. *International Journal of Distributed Sensor Networks, 16*(6), 1550147720922731.
[12]. Filepp, R., Adam, C., Hernandez, M., Vukovic, M., Anerousis, N., & Zhang, G. Q. (2018). Continuous compliance: Experiences, challenges, and opportunities. In *Proceedings of the IEEE World Congress on Services (SERVICES)* (pp. 31–32).
[13]. González-Granadillo, G., González-Zarzosa, S., & Diaz, R. (2021). Security information and event management (SIEM): Analysis, trends, and usage in critical infrastructures. *Sensors, 21*(14), 4759.
[14]. Alicea, M., & Alsmadi, I. (2021). Misconfiguration in firewalls and network access controls: Literature review. *Future Internet, 13*(11), 283.
[15]. Beigi-Mohammadi, N., Litoiu, M., Emami-Taba, M., et al. (2018). A DevOps framework for quality-driven self-protection in web software systems. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering* (pp. 270–274).
[16]. Rafi, S., Yu, W., & Akbar, M. A. (2020). Towards a hypothetical framework to secure DevOps adoption: Grounded theory approach. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering (EASE)* (pp. 457–462).