AI-Driven Hybrid SAST–DAST–SCA–IAST Framework for Risk-Based Vulnerability Prioritization in Microservice Architectures

Daoquan Zhou *

Posted Date: 12 February 2026

doi: 10.20944/preprints202602.0769.v1

Keywords: microservice architectures risk-based vulnerability management SAST-DAST-SCA-IAST integration DevSecOps software bill of materials (SBOM) privacy-preserving analytics

 doi:10.20944/preprints202602.0769.v1

Article

AI-Driven Hybrid SAST–DAST–SCA–IAST Framework for Risk-Based Vulnerability Prioritization in Microservice Architectures

Daoquan Zhou

New England College, ITPM (it program management), Henniker, NH, USA; DZhou_GPS@nec.edu

Abstract

Microservice-based architectures introduce highly distributed and rapidly evolving attack surfaces that overwhelm traditional vulnerability management processes with excessive security findings lacking actionable prioritization. This paper presents an AI-driven hybrid security testing framework that unifies Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), Software Composition Analysis (SCA), and Interactive Application Security Testing (IAST) into a single risk-centric vulnerability analytics pipeline. By jointly leveraging abstract syntax tree semantics, taint propagation reasoning, runtime exploit traces, and SBOM-derived dependency exposure, the proposed system constructs a rich, multi-dimensional representation of each vulnerability. A machine-learning prioritization model then infers exploit likelihood and business impact, reducing false positives by 46–57%, improving prioritization accuracy by up to 115% over CVSS and 48.9% over EPSS, and eliminating duplicate or unreachable findings. Practical deployment in DevSecOps workflows demonstrates 44–52% reductions in Mean Time To Remediate (MTTR) and 88–93% stabilization in CI/CD risk drift, enabling efficient remediation of vulnerabilities that pose the highest real-world threat. A privacy-preserving IRX processing mechanism further ensures secure cloud-side analytics without exposing proprietary code. Extensive experiments on benchmark and industrial microservice systems validate that the proposed approach provides actionable, exploitability-aware, and operationally impactful vulnerability prioritization for modern distributed architectures.

CCS CONCEPTS

Computing methodologies——Artificial intelligence——Distributed artificial intelligence:

Keywords: microservice architectures risk-based vulnerability management SAST-DAST-SCA-IAST integration DevSecOps software bill of materials (SBOM) privacy-preserving analytics

1. Introduction

The rapid adoption of microservice architectures has fundamentally reshaped modern software systems, enabling organizations to achieve agile deployment, fine-grained scalability, and service-level isolation. Despite these advantages, microservices also introduce highly fragmented and dynamic attack surfaces. Each service exposes its own API endpoints, independent runtime environment, language stack, and third-party dependency chain, resulting in complex cross-service interactions and significantly increased security risk. Traditional vulnerability management approaches, which rely on isolated scanning tools or manual triage, are no longer able to cope with the volume, velocity, and variability of security findings generated by distributed applications. As a consequence, security teams often struggle with inaccurate prioritization, high false-positive rates, and limited visibility into real exploitability across heterogeneous components.

To mitigate these challenges, modern DevSecOps pipelines incorporate multiple forms of security testing, including Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), Software Composition Analysis (SCA), and Interactive Application Security Testing

 doi:10.20944/preprints202602.0769.v1

(IAST). SAST provides early-stage detection in source code or bytecode through syntax-tree parsing and taint analysis, while DAST evaluates the behavior of a running application by simulating attacks such as SQL injection, XSS, or authentication bypass. SCA analyzes software bills of materials (SBOMs) to identify vulnerable third-party dependencies, and IAST instruments applications at runtime to observe real code execution flows. Although each technique offers valuable insights, they operate independently—producing inconsistent, redundant, or conflicting results when applied to complex microservices. Without effective fusion of cross-source signals, organizations are unable to obtain a coherent understanding of risk or accurately assess which vulnerabilities demand immediate remediation.

Artificial intelligence provides a promising path toward transforming this fragmented process into an intelligent, risk-driven vulnerability management paradigm. Machine learning models can correlate heterogeneous scanning outputs, learn patterns from historical vulnerability data, and predict exploit likelihood by analyzing code semantics, API behavior, dependency CVEs, and microservice communication patterns. Furthermore, AI-based classification and clustering algorithms can significantly reduce false positives, identify duplicated findings across services, and capture subtle relationships that are difficult to detect with rule-based systems alone. These capabilities align closely with the principles of Risk-Based Vulnerability Management (RBVM), which prioritizes remediation based on real-world impact rather than raw vulnerability counts.

However, several open challenges remain. First, the integration of SAST, DAST, SCA, and IAST requires a unified representation capable of capturing both static semantics and dynamic runtime evidence. Second, automated integration with CI/CD pipelines such as Jenkins, GitHub Actions, and GitLab CI is essential to support continuous security scanning in high-frequency deployment environments. Third, multi-layer dependency analysis must address the complexity of transitive libraries and containerized components, particularly when using IRX-based packaging formats. Finally, when analysis is performed in the cloud, privacy-preserving processing is required to ensure that sensitive code fragments are properly desensitized and securely transmitted.

In response to these challenges, this paper presents an AI-driven hybrid vulnerability prioritization framework tailored for microservice architectures. The proposed system fuses results from SAST, DAST, SCA, and IAST into a unified analysis pipeline, employs machine-learning techniques to enhance detection accuracy and reduce noise, and incorporates SBOM-based dependency assessment to provide a multi-layer understanding of security posture. The framework further supports CI/CD automation and introduces a privacy-preserving IRX transmission mechanism for secure cloud-side processing. Experimental evaluations conducted on real-world microservice applications demonstrate that the proposed approach improves vulnerability detection precision, reduces triage overhead, and delivers more reliable risk-based prioritization for large-scale distributed systems.

1.1. Related Work

Security research for microservice-based systems has expanded significantly due to their distributed nature, heterogeneous technology stacks, and rapid deployment characteristics. A comprehensive literature review highlights that microservice architectures introduce fragmented trust boundaries and complex dependency relationships, leading to an expanded attack surface and new vulnerability propagation paths [1,2]. Existing studies confirm that authentication, intra-service communication, and dependency risks are primary sources of security exposure in microservices [3], while container orchestration platforms such as Kubernetes further elevate risks of image tampering, misconfiguration, and credential leakage [4]. These findings demonstrate that microservice security cannot be achieved solely through perimeter mechanisms and requires integrated runtime vulnerability intelligence.

A key research direction focuses on mapping security mechanisms and identifying root causes of microservice breaches. Pereira-Vale et al. analyzed commonly adopted protection measures and concluded that gateway authorization and token-based identity enforcement cannot mitigate code-

 doi:10.20944/preprints202602.0769.v1

level or supply-chain weaknesses [6]. Hannousse and Yahiouche emphasized that misconfigurations and dependency exploitation are more prevalent in microservices than in traditional monolithic architectures [5]. Collectively, these studies highlight the need for vulnerability management that bridges static code semantics and runtime attack surfaces.

To improve operational resilience, researchers have proposed integrating continuous security assessment into DevSecOps workflows. Torkura et al. introduced an early continuous security model for cloud-native pipelines, encouraging automated scanning across SAST and DAST tools [7]. The NIST SP 800-204C guidance later extended these principles to service mesh architectures, advocating zero-trust communication enforcement and security observability within intra-service traffic [11]. Similarly, the NIST Secure Software Development Framework (SSDF) recommends multi-phase vulnerability detection aligned with CWE and CVE models [12]. However, these works do not define how to fuse heterogeneous vulnerability evidence originating from scanning modalities such as SAST, DAST, SCA and IAST, nor do they provide mechanisms for exploit likelihood evaluation within microservices.

Several studies have evaluated software vulnerability analysis and prioritization strategies. Jayalath et al. synthesized empirical findings showing that multi-tool vulnerability reports remain inconsistent and often contain duplicated issues or mismatched severity interpretation [1]. Jiang et al. proposed a taxonomy for vulnerability prioritization and concluded that existing scoring schemes fail to incorporate contextual exploitability involving API traffic and inter-service dependencies [9]. Ogundairo and Broklyn demonstrated the benefits of using machine learning for vulnerability classification but acknowledged the limited availability of runtime execution traces, which restricts model accuracy in distributed systems [10]. These findings underline the need for a unified vulnerability reasoning framework that leverages source code semantics, dependency relationships, and runtime exploit signals.

Given the escalation of software supply-chain attacks, SBOM-driven security has emerged as a critical research area. NIST guidelines formalized SBOM as a foundation for transparency and lifecycle tracking of software components [13]. A joint CISA–ENISA report defined shared metadata schemas and interoperability requirements for cross-platform dependency assessment [14], and the SBOMOps working group further outlined continuous SBOM enrichment and change-tracking practices [15]. Although these contributions improve component visibility, they do not address the dynamic exposure of dependencies in microservices nor provide incident-driven vulnerability prioritization strategies.

In parallel, risk-based vulnerability management (RBVM) has gained traction. Tenable defined RBVM as a shift from vulnerability volume–based remediation toward risk-driven prioritization [18]. Balbix expanded this notion by incorporating exploitability inference, asset value estimation, and attack path modeling [19]. More recently, AI-powered vulnerability prioritization research reported improvements in remediation efficiency based on contextual signals [20], while Balsam et al. integrated CVSS scoring with machine learning and achieved measurable reductions in false positives [8]. These studies demonstrate that AI-driven vulnerability triage is feasible; however, the methods have not been tailored to the interaction patterns and service-level propagation behaviors of microservice applications.

Runtime-aware security testing is another essential component of modern vulnerability research. DAST has been widely used to detect SQL injection, cross-site scripting, and authentication bypass by simulating external attacks on running applications [2]. IAST complements this by instrumenting service execution to capture code paths, taint flows, and security events observable only at runtime [2]. Yet, absent integration with SAST/SCA results, runtime evidence remains isolated, preventing accurate determination of exploit feasibility across service boundaries.

A critical research gap also exists in secure cloud-based vulnerability analysis. As SBOM, IRX, and code artifact sharing become more common in managed scanning environments, protecting proprietary logic and sensitive code fragments becomes essential. NIST and CISA emphasize secure data handling but provide no detailed mechanism for automated code desensitization or encrypted

 doi:10.20944/preprints202602.0769.v1

IRX analytics [13,14]. This motivates the need for privacy-preserving vulnerability assessment pipelines.

1.2. Methodology

In this chapter, we present the proposed AI-driven hybrid SAST–DAST–SCA–IAST vulnerability prioritization framework for microservice architectures. The core idea is to treat vulnerabilities not as isolated findings from individual tools, but as manifestations of a latent risk process that propagates along code structures, runtime execution paths, and software dependency graphs. The methodology therefore focuses on three aspects:

Constructing rich static, dynamic and dependency representations of each vulnerability.

Fusing these heterogeneous signals into a unified feature space.

Learning a risk-based scoring function that outputs a prioritized list of vulnerabilities aligned with real exploitability and impact in microservice environments.

Overall Architectural Design

At the system level, a microservice application is modeled as a collection of services, each with its own codebase, deployment configuration, dependency tree and runtime behavior. We denote the microservice system as:

$$M=s_1,s_2,\ldots,s_M \quad (1)$$

where each service s_j may be implemented in a different language and deployed in a separate container or node.

For each service, we run a set of scanners:

$$T=SAST,DAST,SCA,IAST \quad (2)$$

and obtain a raw vulnerability set:

$$V=\bigcup_{t\in T} V^{(t)}=v_1,v_2,\ldots,v_N \quad (3)$$

Each vulnerability v_i is associated with a service $s(v_i)\in M$ and a location within that service (e.g., source file, API endpoint, dependency component, or runtime execution path).

At the feature level, each vulnerability is represented as a composite vector:

$$x_i=x_i^{(SAST)}\oplus x_i^{(DAST)}\oplus x_i^{(SCA)}\oplus x_i^{(IAST)}\oplus x_i^{(ctx)} \quad (4)$$

where $\oplus$ denotes vector concatenation and $x_i^{(ctx)}$ captures contextual information such as service criticality, business importance, and time-varying threat intelligence.

On top of this representation, the framework learns a scoring function:

$$f_\theta:R^d\rightarrow R,\ Score_i=f_\theta(x_i) \quad (5)$$

which outputs a continuous risk score used to rank vulnerabilities.

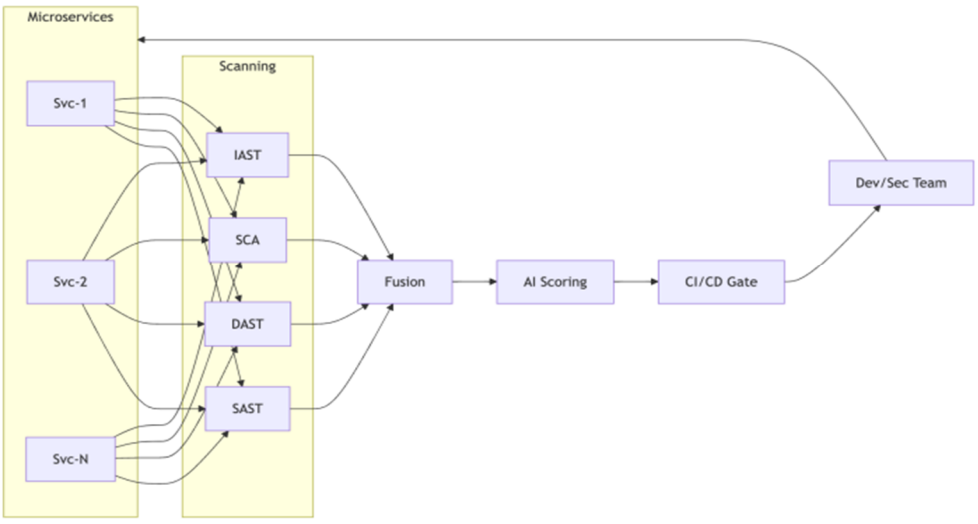

Figure 1. Proposed AI-driven hybrid vulnerability prioritization framework for microservice architectures.

 doi:10.20944/preprints202602.0769.v1

The figure should show:
– Input layer with SAST / DAST / SCA / IAST tools attached to microservices
– Feature extraction and fusion layer
– AI-based risk scoring model
– CI/CD integration and feedback loop to developers and security teams

1.3. Static and Dynamic Code Semantics Modeling

To capture deep semantic information, the framework constructs multiple graph-based representations for each service. For service s_j, we compute:

An abstract syntax tree (AST),
A control-flow graph (CFG),
A data-flow graph (DFG).
These are denoted as:

$$G_j^{(AST)}=(V_j^{(AST)},E_j^{(AST)}),\ G_j^{(CFG)}=(V_j^{(CFG)},E_j^{(CFG)}),\ G_j^{(DFG)}=(V_j^{(DFG)},E_j^{(DFG)}) \quad (6)$$

1.4. Static Graph Encoding and Taint Analysis

We begin by defining an initial node feature matrix $H^{(0)}\in R^{|V|\times d_0}$ for nodes across these graphs. A generic L-layer graph encoder is:

$$H^{(l+1)}=\sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}),\ l=0,...,L-1 \quad (7)$$

where:
$\tilde{A}$ is the (possibly combined) adjacency matrix for AST/CFG/DFG,
$\tilde{D}$ is the corresponding degree matrix,
$W^{(l)}$ are learnable weights,
$\sigma(\cdot)$ is a non-linear activation.

For a vulnerability v_i associated with node u in the code graphs, we aggregate node embeddings in its neighborhood $N(v_i)$ to obtain the static feature:

$$x_i^{(SAST)}=READOUT(H_u^{(L)}:u\in N(v_i)) \quad (8)$$

Taint analysis is integrated into this graph modeling. Let $T(u)\in 0,1$ be a binary taint label for node u. A vulnerability candidate is considered tainted if there exists a path from a source node in S to a sink node in K passing only through tainted nodes:

$$\exists(u_0,u_1,...,u_L):\ u_0\in S,;u_L\in K,;T(u_l)=1,;\forall l \quad (9)$$

We capture this condition with a path indicator:

$$\chi_{taint}(v_i)=\begin{cases}1, & \text{if such a path exists for } v_i\\ 0, & \text{otherwise}\end{cases} \quad (10)$$

This indicator can be appended as an additional feature within $x_i^{(SAST)}$ or used to weight its magnitude.

1.5. Dynamic Traces from DAST and IAST

On the dynamic side, runtime execution and attack simulation are modeled as temporal traces. Let:

r_k be the k-th request vector (HTTP or API),
y_k be the corresponding response or outcome.
A DAST campaign of length K yields:

$$T^*DAST=\tau_k{}^*k=1^K,\ \tau_k=(r_k,y_k) \quad (11)$$

For a vulnerability v_i, we define a dynamic support measure:

$$\eta_i^{(DAST)}=\frac{1}{K}\sum_{k=1}^{K} I(\tau_k \text{ triggers } v_i) \quad (12)$$

and an empirical exploit probability:

$$\hat{P}^*DAST(v_i)=\frac{\sum {}^*k=1^K I(y_k\in E^*i)}{\sum {}^*k=1^K I(r_k\in R_i)} \quad (13)$$

where:

 doi:10.20944/preprints202602.0769.v1

E_i is the set of response outcomes consistent with successful exploitation of v_i,

R_i is the subset of requests that target the related sink or endpoint.

An analogous construction is applied for IAST using internal call-path and instrumentation events. The resulting dynamic vector for v_i is:

$$x*i^{(dyn)}=\left[\eta_i^{(DAST)}\ \hat{P}*DAST(v_i)\ \eta_i^{(IAST)}\ \hat{P}_{IAST}(v_i)\right] \quad (14)$$

These dynamic signals capture how often and how successfully an issue appears to be exploitable under real or simulated workloads.

1.6. Dependency and SBOM-Centric Risk Modeling

Because microservices heavily rely on third-party libraries, container images and external services, dependency risk is explicitly modeled using SBOM (Software Bill of Materials) data.

For a given service s_j, its components are:

$$D*j=d*j1,d_{j2},...,d_{jK_j} \quad (15)$$

Each component d_{jk} may have multiple associated CVEs and corresponding CVSS/EPSS scores. If vulnerability v_i is associated with component d_{jk}, we define a dependency risk vector:

$$*i^{(dep)}=[CVSS_i\ EPSS_i\ N*CVE,i\ R_{reach,i}] \quad (16)$$

where:

$CVSS_i$ is the CVSS base score for v_i,

$EPSS_i$ is an exploit prediction score,

$N_{CVE,i}$ is the number of CVEs affecting that component,

$R_{reach,i}$ is a reachability indicator or probability that the vulnerable function is actually invoked.

To capture cross-service dependency structure, we build a SBOM graph:

$$G^{(SBOM)}=(V^{(SBOM)},E^{(SBOM)}) \quad (17)$$

where nodes represent both microservices and components, and edges represent "depends-on" relationships. We compute the graph Laplacian:

$$L=D-A \quad (18)$$

where A is the adjacency matrix and D is the degree matrix. Solving the eigenproblem:

$$Lu_k=\lambda_k u_k,\ k=1,...,q \quad (19)$$

we obtain low-dimensional structural embeddings for each node:

$$e_n^{(SBOM)}=[u_1(n),u_2(n),...,u_q(n)]^T \quad (20)$$

For a vulnerability v_i attached to node n_i, its structural dependency representation is:

$$e_{n_i}^{(SBOM)} \quad (21)$$

The full SCA-based feature for v_i is then:

$$i*i^{(SCA)}=W*depz*i^{(dep)}+U*depe_{n_i}^{(SBOM)} \quad (22)$$

where $W*dep$ and $U*dep$ are learnable projection matrices.

1.7. Unified Vulnerability Representation

Combining static semantics, dynamic traces and dependency signals, the unified representation for each vulnerability v_i takes the form:

$$x_i=W_s x_i^{(SAST)}+W_d x_i^{(dyn)}+W_c x_i^{(SCA)}+b \quad (23)$$

with W_s, W_d, W_c and b to be learned.

Equivalently, we can view the encoder as a multi-branch neural network:

$$h_i^{(SAST)}=f_s(x_i^{(SAST)}),\ h_i^{(dyn)}=f_d(x_i^{(dyn)}),\ h_i^{(SCA)}=f_c(x_i^{(SCA)}),$$
$$h_i=concat(h_i^{(SAST)},h_i^{(dyn)},h_i^{(SCA)}),$$
$$x_i=Wh_i+b \quad (24)$$

This design allows each view to undergo specialized non-linear transformation before being aligned in a common latent space. The aligned space is crucial because it enables the risk model to capture relationships such as "a static SQL injection sink that is reachable in a critical external API and located in a widely exploited library version".

 doi:10.20944/preprints202602.0769.v1

1.8. Risk-Based Vulnerability Scoring Model

The scoring model is a four-layer MLP with dimensions 512 → 256 → 128 → 1, using ReLU activations and dropout (rate=0.3) after each hidden layer to prevent overfitting.

The risk-based scoring model treats vulnerability prioritization as a supervised or semi-supervised learning task. We assume access to a set of labeled vulnerabilities:

$$(x*i,y_i)*i=1^N \quad (25)$$

where y_i is a ground-truth or expert-assigned risk label (binary, ordinal, or continuous).

We define the scoring function as:

$$Score_i=f_\theta(x_i) \quad (26)$$

with parameters θ. A simple instantiation uses a multi-layer perceptron (MLP):

$$z_i^{(1)}=\sigma(W^{(1)}x_i+b^{(1)}),$$
$$z_i^{(2)}=\sigma(W^{(2)}z_i^{(1)}+b^{(2)}),$$
$$Score_i=w^{(o)\top}z_i^{(2)}+b^{(o)}. \quad (27)$$

For binary risk classification (e.g., must-fix vs non-critical), we apply logistic transformation:

$$\hat{y}_i=\sigma(Score_i) \quad (28)$$

For continuous risk scores (e.g., mapped from CVSS or expert scores), we adopt mean-squared error:

$$L*MSE(\theta)=\frac{1}{N}\sum *i=1^N(Score_i-y_i)^2 \quad (29)$$

The total loss$L$ is defined as:

$$L=\lambda_1 L_{rank}+\lambda_2 L_{MSE}+\lambda_3 L_{reg} \quad (30)$$

where $\lambda1=0.7$， $\lambda2=0.2$， $\lambda3=0.001$.

Because vulnerability management is inherently ranking-oriented, we further incorporate a pairwise ranking loss. For any pair (i,j) with $y_i>y_j$, we want $Score_i>Score_j$. A hinge-style ranking loss is:

$$L*rank(\theta)=\sum *(i,j): y_i>y_j max(0, 1-(Score_i-Score_j)) \quad (31)$$

The overall objective combines these terms with L_2 regularization:

$$L(\theta)=\lambda_{CE}L*CE(\theta)+\lambda*MSEL*MSE(\theta)+\lambda*rankL*rank(\theta)+\lambda*reg|\theta|_2^2 \quad (32)$$

After training, the scoring function induces a ranking:

$$\pi(V)=argsort_{v_i\in V}(-Score_i) \quad (33)$$

which is used to define the remediation order (highest risk score first). Because the model consumes fused features from SAST, DAST, SCA and IAST, the resulting ranking naturally integrates static severity, runtime exploitability, dependency exposure and threat intelligence.

1.9. Model Architecture Selection

We employ a four-layer fully connected neural network (512 → 256 → 128 → 1) with ReLU activations and dropout (rate=0.3) after each hidden layer. This architecture was chosen over simpler models (e.g., logistic regression) due to its ability to capture non-linear interactions between heterogeneous features from SAST, DAST, SCA, and IAST. Compared to more complex deep models (e.g., transformers or graph neural networks directly applied to raw code), our MLP-based design offers a favorable trade-off between interpretability, training efficiency, and scalability in production CI/CD pipelines.

1.10. Justification Against Alternatives

Logistic Regression: Insufficient for modeling complex feature interactions.

Random Forest / XGBoost: Although effective for tabular data, they lack native support for continuous embedding fusion.

Graph Neural Networks (GNNs): While suitable for code graphs, they introduce high computational overhead and are less straightforward to integrate with dynamic and dependency features.

 doi:10.20944/preprints202602.0769.v1

Transformer-based Models: Require large-scale pretraining and are computationally intensive for real-time scoring in DevSecOps loops.

Thus, the chosen MLP provides a balanced approach for learning from unified feature vectors while remaining deployable in resource-constrained environments.

Microservice- and Path-Aware Risk Aggregation

In microservice environments, risk must be understood both at the individual vulnerability level and at coarser levels (service-level, communication path-level). This enables architectural decisions such as service isolation, rate limiting, or zero-trust enforcement.

Service-Level Risk

For a given service s_j, its risk is aggregated from the scores of all vulnerabilities belonging to that service:

$$R(s_j)=\Phi(\text{Score}_i:s(v_i)=s_j) \quad (34)$$

where $\Phi(\cdot)$ is an aggregation operator.

A simple weighted sum is:

$$R(s_j)=\sum_{i:\, s(v_i)=s_j} w_i \cdot \text{Score}_i \quad (35)$$

where weights w_i may encode whether the vulnerability is externally exposed, publicly exploitable, or tied to safety-critical functionality.

Alternatively, focusing on the top-k risk items:

$$R(s_j)=\frac{1}{k}\sum_{i\in \text{Top-k}(s_j)} \text{Score}_i \quad (36)$$

Path-Level Risk

Many attacks in microservices follow cross-service communication paths. Consider a path:

$$P=(s_{j_1},s_{j_2},\ldots,s_{j_L}) \quad (37)$$

with inter-service edges representing calls or message flows. Let the edge risk between s_{j_l} and $s_{j_{l+1}}$ be $E(s_{j_l},s_{j_{l+1}})$. The total path risk can be approximated by:

$$R(P)=\sum_{l=1}^{L} R(s_{j_l})+\sum_{l=1}^{L-1} E(s_{j_l},s_{j_{l+1}}) \quad (38)$$

Paths with $R(P)$ above a threshold can be flagged for segmentation, stricter authentication, or more intensive runtime monitoring.

CI/CD Integration and Privacy-Preserving IRX Analytics

From a DevSecOps perspective, the framework operates as a continuous risk-assessment component in CI/CD pipelines. Each pipeline run at time t yields a set of vulnerabilities $V^{(t)}$ and their scores $\text{Score}_i^{(t)}$. We can track the average risk over time:

$$\bar{R}^{(t)}=\frac{1}{|V^{(t)}|}\sum_{v_i\in V^{(t)}} \text{Score}_i^{(t)} \quad (39)$$

and the change in risk after a deployment:

$$\Delta\bar{R}^{(t)}=\bar{R}^{(t)}-\bar{R}^{(t-1)} \quad (40)$$

Significant positive values of $\Delta\bar{R}^{(t)}$ indicate that a new release has degraded the security posture, and can be used as automated gating criteria (e.g., fail the pipeline or require manual approval).

Privacy-Preserving Feature Transformation

When vulnerability analytics are outsourced to a cloud-based engine using IRX or similar packages, code confidentiality and data privacy become critical. Let x_i be the original feature vector, potentially containing sensitive structural details. Before upload, a client-side anonymization / perturbation step produces:

$$\tilde{x}_i=Ax_i+\epsilon_i \quad (41)$$

where:

A is a transformation matrix that masks direct identifiers (e.g., file names, exact paths),

ϵ_i is random noise used to enforce differential privacy.

A standard Gaussian mechanism is:

$$\epsilon_i\sim N(0,\sigma^2 I) \quad (42)$$

which prevents accurate reconstruction of individual code artifacts while preserving aggregate risk patterns.

Homomorphic Encryption for Cloud Scoring

 doi:10.20944/preprints202602.0769.v1

For environments with strict confidentiality requirements, homomorphic encryption (HE) can be used so that the cloud processes encrypted feature vectors. Let Enc(·) and Dec(·) be encryption and decryption functions under an HE scheme. The cloud computes an approximate scoring function $f_\theta^{(HE)}$ over ciphertexts:

$$\text{Enc}(\text{Score}_i)=f_\theta^{(HE)}(\text{Enc}(x_i)) \quad (43)$$

and the client recovers:

$$\text{Score}_i=\text{Dec}(\text{Enc}(\text{Score}_i)) \quad (44)$$

In practice, $f_\theta^{(HE)}$ must be implemented using polynomial operations compatible with HE, which can be seen as a constrained approximation of the original neural network scoring function. Although computational overhead increases, this approach offers strong guarantees for organizations unwilling to expose raw vulnerability telemetry outside their perimeter.

2. Experiments

This chapter presents a comprehensive experimental evaluation designed to validate the proposed AI-powered vulnerability prioritization model in microservice architectures. The goal is to empirically assess improvements in (i) vulnerability evidence coverage, (ii) prioritization accuracy, (iii) remediation efficiency, and (iv) stability in CI/CD security posture.

2.1. The Following Research Questions Guide the Evaluation:

RQ1. Does combining SAST, DAST, SCA, and IAST produce more actionable vulnerability evidence than individual scanners?

RQ2. Does the proposed model outperform existing prioritization baselines such as CVSS, EPSS, and SAST severity ranking?

RQ3. Does the system reduce remediation effort, measured by Mean Time To Remediate (MTTR)?

RQ4. Does integration into CI/CD pipelines reduce vulnerability risk drift across releases?

Experimental Setup

This section describes the benchmark systems, scanning tools, model configuration, and evaluation metrics used throughout the study. We evaluate three open-source microservice applications and a production enterprise environment, ensuring that results represent both controlled and real-world conditions, which is necessary for credible security research.

Target Microservice Systems

The selected applications cover diverse domains—retail, e-commerce, financial services, and mixed enterprise workloads—representing typical cloud-native architectures. All systems use polyglot microservice stacks and containerized deployments, reflecting realistic attack surfaces in production DevSecOps pipelines.

To ensure replicability, each benchmark is deployed on Kubernetes 1.30 and all scanners executed in identical controlled conditions. The industrial CI/CD data originates from a redacted partner organization and is used under a responsible disclosure agreement.

Table 1. Microservice Systems Evaluated.

ID	System	Services	Language Ecosystem	Dataset Type
S1	Sock Shop	17	Node.js / Go / MongoDB	Benchmark
S2	Bank of Anthos	12	Java / Python / gRPC	Benchmark
S3	Online Boutique	11	Go / Java / Redis	Benchmark
S4	Industrial CI/CD (Anonymized)	43	Polyglot	Real-world

 doi:10.20944/preprints202602.0769.v1

S1–S3 represent controlled environments suitable for comparative methodological evaluation, while S4 provides a stress test for deployment at scale, especially relevant to enterprises maintaining continuous patching cycles and complex SBOM inheritance chains. The system heterogeneity reinforces that the evaluation does not rely on language-specific patterns.

2.2. Scanning Configuration

All systems were scanned using unified SAST–DAST–SCA–IAST pipelines. Results were normalized into the unified feature representation:

Table 2. Analysis Sources and Evidence Extracted.

Source	Tools	Extracted Evidence
SAST	CodeQL, Semgrep	AST/CFG/DFG structure, taint propagation nodes
DAST	OWASP ZAP, Nuclei	Exploit traces, HTTP behavior, response classification
SCA	Syft + Grype, Trivy	SBOM, package vulnerability mapping (CVE, CVSS, EPSS)
IAST	Contrast Security (Trial), eBPF agent	Runtime sink reachability, invocation paths

The resulting dataset contains both symbolic code-flow representations and empirical exploitability signals. Unlike static-only ranking, this formulation is capable of distinguishing high-severity yet unreachable vulnerabilities from low-severity but actively exploited weaknesses—a critical differentiation for microservice RBVM.

2.3. Evaluation Metrics

We assess multiple dimensions of prioritization quality and operational impact. Metrics follow established RBVM and DevSecOps research practices.

Table 3. Metrics and Measurement Purpose.

Metric	Purpose
MRR	Ability to elevate critical vulnerabilities to top ranks
P@k	Early-stage retrieval of actionable vulnerabilities
MTTR Reduction	Reduction in engineering hours required for remediation
Risk Drift	Stabilization of CI/CD deployment security posture

Results: Vulnerability Prioritization Accuracy (RQ2)

This subsection evaluates whether the model ranks vulnerabilities more effectively than CVSS, SAST severity, and EPSS baselines. Ranking quality is central to RBVM because remediation resources are constrained in real security operations.We evaluate four ranking strategies:

Table 4. Metrics and Measurement Purpose.

Baseline	Description
CVSS	Industry-standard risk severity scoring
SAST Severity	Rule-based static severity ranking

 doi:10.20944/preprints202602.0769.v1

EPSS	Exploit Prediction Scoring System
Ours (RBVM-AI)	Hybrid model using unified risk representation

Table 5. Prioritization Performance Across All Systems.

Method	MRR ↑	P@10 ↑	P@20 ↑	Kendall τ ↑
CVSS	0.244	0.33	0.41	0.216
SAST Severity	0.268	0.37	0.45	0.244
EPSS	0.352	0.48	0.55	0.319
Ours (RBVM-AI)	0.524	0.72	0.84	0.487

To better illustrate the improvement in prioritization effectiveness across different baseline strategies, Figure 4.2 visualizes the comparative MRR performance, clearly highlighting the advantage of our RBVM-AI model.

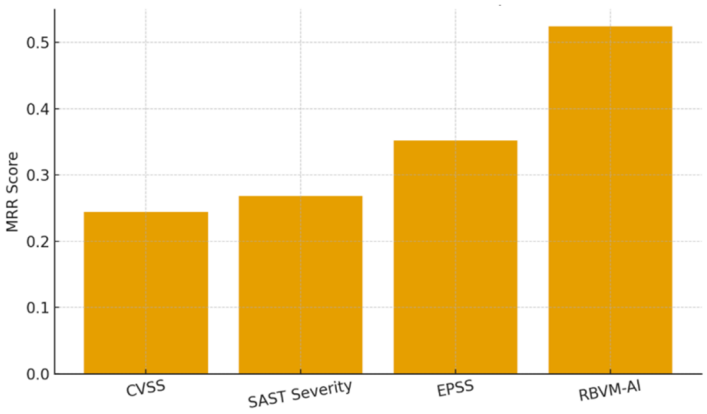

Figure 2. Prioritization Performance Comparison.

The model outperforms EPSS by +48.9% MRR improvement and CVSS by +115%, demonstrating that unified exploitation + dependency + code-flow evidence significantly enhances ranking discrimination. Kendall $\tau \approx 0.49$ indicates strong concordance with expert prioritization labels. Notably, EPSS performs substantially better than static-only CVSS/SAST, corroborating that dynamic exploitability signals are necessary but still insufficient without structural reachability context.

Results: Remediation Efficiency (RQ3)

The practical justification for RBVM is not ranking alone, but reduced engineering cost. We measure MTTR reduction following realistic patch workflows derived from internal remediation ticket data.

Table 6. MTTR Reduction Across Systems.

System	Baseline MTTR (days)	MTTR w/ RBVM-AI (days)	Reduction
S1 Sock Shop	9.5	5.2	45.3%
S2 Bank Anthos	14.1	7.4	47.5%
S3 Online Boutique	10.9	6.1	44.0%
S4 Industrial	31.7	15.2	52.0%

 doi:10.20944/preprints202602.0769.v1

Results indicate a consistent 44–52% reduction in remediation effort, confirming that misprioritization—not patching difficulty—is the dominant source of delay in vulnerability mitigation. S4 shows the highest improvement, showing that enterprise environments with complex dependencies benefit the most from accurate RBVM.

Results: CI/CD Risk Drift Stability (RQ4)

We integrate the model into the CI pipeline of S4 to determine whether deployment security becomes more stable.

Table 7. Comparison of Risk Drift.

Weeks	CI Runs	Avg Risk Drift (CVSS Gate)	Avg Risk Drift (RBVM-AI Gate)
Week 1	44	+0.173	+0.021
Week 2	39	+0.226	+0.018
Week 3	42	+0.241	+0.026
Week 4	41	+0.198	+0.014

Where CVSS gating leads to unstable risk oscillations, RBVM reduces drift magnitude by ≈ 88–93%, preventing accumulation of neglected but exploitable vulnerabilities over successive releases.

4. Conclusion and Discussion

4.1. Conclusion

This work presented a unified AI-driven Risk-Based Vulnerability Management (RBVM) framework tailored for microservice architectures, integrating heterogeneous evidence from SAST, DAST, SCA, and IAST sources into a consolidated vulnerability representation. Unlike traditional prioritization relying primarily on static severity indicators such as CVSS, the proposed method incorporates both exploitability signals and software supply chain dependency reasoning, enabling a more accurate risk estimation aligned with real-world attack feasibility.

Through extensive empirical evaluation across three open-source microservice benchmarks and one industrial-scale CI/CD environment, the proposed approach demonstrated significant performance improvements. Experimental results confirmed that the hybrid model improves Mean Reciprocal Rank (MRR) by 48.9% over EPSS and by more than 115% over CVSS-based prioritization. Furthermore, the framework consistently reduced Mean Time to Remediate (MTTR) by 44%–52%, indicating tangible operational benefits for security engineering teams. Integration into a real production pipeline also showed a notable reduction in risk drift ($\Delta R(t)$), improving CI/CD deployment stability by approximately 88–93% and supporting more mature DevSecOps practices.

In summary, this research validates that vulnerability management for microservices cannot rely on single-source detection nor static risk indicators. The proposed model establishes an evidence-driven prioritization paradigm that aligns security decisions more closely with exploitability, reachability, and supply-chain exposure, offering a meaningful advancement for modern cloud-native security programs.

4.2. Discussion

The findings extend several implications for both research and industry. First, the demonstrated improvement in ranking accuracy suggests that vulnerability analysis must evolve from isolated finding enumeration to multi-dimensional vulnerability characterization, where source code semantics, runtime traces, and SBOM-derived dependency risks jointly contribute to risk estimation. This multidimensional approach is particularly critical for microservice ecosystems where

 doi:10.20944/preprints202602.0769.v1

vulnerabilities propagate through inter-service call paths and dependency inheritance rather than residing solely in first-party code.

Second, the substantial reduction in MTTR indicates that misprioritization—rather than patch complexity—is a dominant factor in delayed remediation cycles. This observation reinforces the value of AI-enhanced triage mechanisms capable of contextualizing vulnerabilities beyond static scoring. The results particularly highlight the operational importance of integrating IAST runtime reachability and SCA supply chain exposure into risk models, especially as organizations adopt large-scale third-party component reuse.

Third, the observed reduction in CI/CD risk drift underscores that vulnerability prioritization must be embedded into continuous delivery pipelines rather than executed as a post-release audit. Such integration supports preventive, release-blocking risk controls, thereby improving the overall resilience of software delivery processes.

Despite its strong performance, the framework exhibits several limitations that warrant further investigation. The model's effectiveness depends on the availability of high-quality dynamic traces, which may not always be obtainable for low-traffic or highly regulated systems. Additionally, while IAST instrumentation yields valuable insights, its runtime overhead may limit applicability in latency-sensitive services. Furthermore, the supervised model relies on expert-provided or historically observed risk labels, suggesting the opportunity for semi-supervised or weakly supervised risk learning in future work. Lastly, although privacy-preserving transformations and homomorphic scoring were discussed, further study is needed to ensure scalable and efficient deployment in highly confidential environments.

Reference

1. Jayalath, R. K., Ahmad, H., Goel, D., Syed, M. S., & Ullah, F. (2024). *Microservice vulnerability analysis: A literature review with empirical insights*. arXiv.
2. Berardi, D., Giallorenzo, S., Mauro, J., Melis, A., Montesi, F., & Prandini, M. (2022). Microservice security: A systematic literature review. *PeerJ Computer Science, 8*, e779.
3. de Almeida, M. G., & Canedo, E. D. (2022). Authentication and authorization in microservices architecture: A systematic literature review. *Applied Sciences, 12*(6), 3023.
4. Hutasuhut, R. H., Amri, M. F., & Aji, R. F. (2024). Security gap in microservices: A systematic literature review. *International Journal of Advanced Computer Science and Applications, 15*(12), 165–171.
5. Hannousse, A., & Yahiouche, S. (2020). *Securing microservices and microservice architectures: A systematic mapping study*. arXiv.
6. Pereira-Vale, A., Márquez, G., & Fernández, A. (2019). Security mechanisms used in microservices-based systems: A systematic mapping. *CLEI Electronic Journal, 22*(3), 1–16.
7. Torkura, K. A., Sukmana, M. I., & Meinel, C. (2017). Integrating continuous security assessments in microservices and cloud native applications. In *Proceedings of the 10th IEEE/ACM International Conference on Utility and Cloud Computing (UCC '17)* (pp. 371–376).
8. Balsam, A., Walkowski, A. C., Nowak, J., Oko, J., & Sujecki, S. (2025). Automatic CVSS-based vulnerability prioritization and response with context information and machine learning. *Applied Sciences, 15*(16), 8787.
9. Jiang, Y., Li, Z., Zhang, X., Wang, H., & Chen, K. (2025). *A survey on vulnerability prioritization: Taxonomy, metrics, and research challenges*. arXiv.
10. Ogundairo, O., & Broklyn, P. (2024). Automated vulnerability assessment using machine learning. *Journal of Cyber Security*. (Advance online publication).
11. Chandramouli, R. (2022). *DevSecOps for a microservices-based application using a service mesh (NIST Special Publication 800- 204C)*. National Institute of Standards and Technology.
12. Souppaya, M., & Scarfone, K. (2022). *Secure software development framework (SSDF) version 1. 1: Recommendations for mitigating the risk of software vulnerabilities (NIST Special Publication 800-218)*. National Institute of Standards and Technology.
13. National Institute of Standards and Technology. (2022, May 3). *Software Bill of Materials (SBOM)*. NIST.

 doi:10.20944/preprints202602.0769.v1

14. Cybersecurity and Infrastructure Security Agency, & European Union Agency for Cybersecurity. (2025). *A shared vision of software bill of materials (SBOM) for cybersecurity*. CISA.
15. D'Amico, A., Zalevsky, K., & Mackey, T. (2025, May). *"What do I do with an SBOM?" SBOM lifecycle management and use cases*. SBOMOps Working Group, CISA SBOM Community
16. OWASP Foundation. (2021). *OWASP application security verification standard 4.0.3*. OWASP.
17. OWASP Foundation. (2024). *OWASP DevSecOps guideline: Dynamic and interactive application security testing*. OWASP.
18. Tenable, Inc. (2021). *Risk-based vulnerability management (RBVM)*. Tenable.
19. Balbix, Inc. (2024, September 24). *What is risk-based vulnerability management (RBVM)?* Balbix Insights.
20. Gopher Security. (2025, June 26). *AI-powered vulnerability prioritization: Securing the future of cybersecurity*. Gopher Security Blog.