Kuwait Journal of Advanced Computer Technology
ISSN: 2984-7907
Vol 4 Issue 1 (2026)

Autonomous Security Testing and Vulnerability Prediction in Cloud-Native DevSecOps Using AI

Akshay Bansal[1], Pratap Patwal[2]

[1]Author, Computer Science and Engineering
(M Tech – Final Year)
Bikaner Technical University / Laxmi Devi Institute of Engineering and Technology
Email: aksban1@gmail.com

[2]Supervisor / Guide, Head of Department, Computer Science and Engineering
Laxmi Devi Institute of Engineering and Technology
Email: pratappatwal@gmail.com

Submitted: 03/12/2025 **Revised**: 17/12/2025 **Published**: 28/12/2025

Abstract

Cloud-native development and CI/CD have accelerated software delivery but expanded the attack surface. Integrating autonomous security testing and predictive vulnerability analytics into DevSecOps workflows promises to shift security left while reducing alert noise and manual triage. This review synthesizes recent work on AI-driven security testing (SAST/DAST/IAST/IaC), machine learning models for vulnerability prediction and prioritization, evaluation of benchmarks and datasets, and practical challenges for production adoption. We identify limitations in datasets and evaluation practices, discuss responsible deployment (explainability, privacy, false positive risk), and propose a research roadmap for robust, context-aware, and auditable autonomous DevSecOps systems.

Keywords: DevSecOps, cloud-native, CI/CD, autonomous security testing, vulnerability prediction, machine learning, vulnerability prioritization, SAST, DAST, IaC.

1. Introduction

Modern software development increasingly targets cloud-native platforms (microservices, containers, Kubernetes) and relies on automated CI/CD pipelines to deliver frequent updates. While this improves velocity, it also multiplies the rate at which new code — and thus potential vulnerabilities — are introduced. DevSecOps advocates embedding security into the pipeline, but manual security reviews cannot scale to the velocity and complexity of distributed cloud systems. AI and ML methods promise two complementary capabilities: (1) autonomous security testing that integrates into CI/CD to detect and sometimes remediate vulnerabilities automatically, and (2) predictive models that estimate which newly-disclosed or discovered vulnerabilities are likely to be exploited or pose high risk, enabling prioritized remediation. Industry surveys show growing adoption of security automation in CI/CD, but challenges remain around accuracy, evaluation, and operational trust.

2. Background

2.1 Cloud-Native DevSecOps

Cloud-native systems emphasize containerization, microservices, and infrastructure as code (IaC). DevSecOps integrates security gates and tooling (SAST, DAST, SCA, IaC scanning, runtime monitoring) directly into pipelines so that security feedback is fast and actionable. Popular CI/CD platforms (GitHub Actions, GitLab CI, Jenkins) are extended with scanners and policy-as-code. Automation reduces human error but requires reliable detection and meaningful prioritization to avoid developer fatigue.

2.2 Security Testing Modalities

- **SAST (Static Application Security Testing):** analyzes source code or binaries for pre-execution to find code patterns that map vulnerabilities.

- **DAST (Dynamic Application Security Testing):** test running applications (black box) to find runtime issues.
- **IAST (Interactive Application Security Testing):** combines static and dynamic information during tests or runtime agents.
- **IaC Scanning** analyzes Terraform/Ansible/Kubernetes manifests for misconfigurations.
- **Runtime/RASP/Observability** monitors live systems for anomalous behavior.

Each modality produces alerts with different characteristics (precision, context needs), making unified triage and prioritization a challenge.

3. AI Techniques Applied to Autonomous Testing & Prediction

3.1 Supervised Learning & Classic ML

Feature-based classifiers (random forests, gradient boosting) have been used to predict severity or exploitability from CVE metadata, package attributes, and historical exploit occurrences. These approaches rely on curated features (CVSS fields, references, textual descriptions) and labelled outcomes (whether exploited).

3.2 Natural Language & Representation Learning

Transformer-based models and embeddings applied to vulnerability descriptions; patch diffs, and code snippets enable richer representations for predicting exploitability, severity, or root cause categories. They also power code-analysis assistants that suggest fixes or detect patterns missed by rule-based scanners.

3.3 Unsupervised & Anomaly Detection

Unsupervised models detect deviations in telemetry (API call distributions, resource usage) to flag suspicious runtime behavior not covered by signatures. These are important for detecting zero-day behavior and configuration of drifts in cloud environments.

3.4 Reinforcement & AutoML Approaches

RL agents have been proposed for automated incident response and active testing (e.g., adaptive fuzzing strategies), though their real-world deployment remains experimental. AutoML pipelines assist in model selection and hyperparameter tuning speed experimentation in security contexts.

4. Literature Review

4.1 Autonomous Security Testing in CI/CD

Recent studies and industry reports document increased integration of automated security tools into pipelines, with AI components enhancing triage and detection. Practical toolchains combine SAST/DAST/IaC scanning with security orchestration (SOAR) and automated policy enforcement. However, tooling remains fragmented: many pipelines chain multiple specialized scanners which produce overlapping and noisy alerts. Research emphasizes automation plus human-in-the-loop oversight to avoid blocking delivery.

4.2 Vulnerability Prediction & Prioritization

A growing body of work seeks to predict which disclosed vulnerabilities will be exploited (exploit prediction) or which poses the greatest operational risk for a given organization. Data-driven scoring systems like EPSS provide probabilistic exploit estimates; ML models attempt to improve static CVSS scores by incorporating contextual signals and historical exploitation patterns. Systematic reviews indicate that traditional scoring (CVSS) lacks contextual and temporal sensitivity; ML approaches can add predictive power but depend heavily on dataset quality and temporal validation.

4.3 Benchmarks, Datasets, and Evaluation Issues

Benchmarks for ML-based vulnerability detection/prioritization are proliferating, but recent meta-analyses highlight methodological weaknesses: improper train/test splits, label leakage, unrealistic sampling of positive examples, and inconsistent metrics. The ML4VD community has raised concerns about reproducibility and

overfitting to narrow datasets. Robust temporal evaluation (training on older vulnerabilities, testing on future ones) is essential but not always done.

4.4 Industrial Adoption Case Studies

Case studies from enterprises show that combining automated scanners with ML-driven triage reduces mean time to remediation high-risk issues, but teams often disable noisy tools or tune them heavily. Use of context (asset criticality, exposure, exploit trends) in prioritization of workflows proves crucial.

5. Datasets and Tooling

5.1 Public Data Sources

- **NVD/CVE feeds** for vulnerability metadata and CVSS scores.
- **EPSS** for historical exploitation probability estimates.
- **Exploit DB / vendor advisories / malware feeds** for evidence of exploitation.
- **Open-source code corpora and patch diffs** for model training.

5.2 Tooling Ecosystem

Open-source and commercial SAST/DAST/IaC tools are commonly integrated into pipelines; newer offerings add ML-based triage, fuzzy matching of findings, or automated fix suggestions. Evaluating models often requires combining scanner outputs with downstream incident records, which is challenging due to data silos.

6. Challenges & Gaps

6.1 Data Quality and Labeling

- Exploit labels are noisy: absence of observed exploitation does not equal no risk.
- Temporal leakage: using features that become known only after disclosure inflates performance.
- Asset/context signals are often private and unavailable for public benchmarking.

6.2 Evaluation and Reproducibility

- Many studies report high in-sample accuracy but fail temporal generalization tests.
- Metrics like precision@k and time-to-detection may be more operationally meaningful than ROC AUC.

6.3 Explainability & Developer Trust

- Black-box models that flag high-risk vulnerabilities without explainable reasons hinder developer adoption. Explainable outputs and recommended remediation steps increase trust.

6.4 Operational Integration

- Integrating prediction outputs into ticketing, automated patching, or gating policies requires robust confidence calibration and rollback safety. False positives can disrupt delivery; false negatives risk breaches.

6.5 Adversarial & Privacy Concerns

- Attackers may probe and poison telemetry or intentionally craft vulnerabilities to evade ML detection.
- Models trained on sensitive telemetries can leak organization-specific signals unless privacy-preserving learning is used.

7. Best Practices & Design Patterns for Autonomous DevSecOps

1. **Shift-Left with Guardrails:** Combine fast SAST checks in pre-commit with heavier DAST/IAST in staging; fail builds only on high-confidence findings.
2. **Contextual Prioritization:** Fuse exploit prediction with asset criticality, public exploit trends, network exposure, and business impact.

3. **Temporal Validation:** Always validate models using temporal holdout (train on older data, test on newer) to simulate real deployment.
4. **Human-in-the-Loop:** Present model outputs as recommendations with explanations and confidence scores; enable easy escalation or override.
5. **Audit Trails & Explainability:** Log model inputs/outputs and provide human-readable rationales for flagged items for compliance and forensics.
6. **Continuous Retraining & Monitoring:** Monitor model drift and retrain as threat landscape and code patterns evolve.

8. Open Research Directions

8.1 Context-Aware, Asset-Specific Prioritization

Develop models that accept organization-level context (CICD topology, exposure, asset criticality) while preserving privacy (federated learning or secure aggregation). This reduces false positives and focuses remediation on what matters.

8.2 Robust Benchmarking & Datasets

Create community benchmarks that enforce temporal splits, realistic class imbalance, and include multi-modal inputs (textual descriptions, patches, telemetry). Shared, well-curated corpora will improve comparability and reproducibility.

8.3 Explainable Vulnerability Prediction

Design models that emit human-interpretable features (e.g., which tokens in a patch or description drove the prediction) and provide counterfactuals to guide fixes.

8.4 Autonomous Test Orchestration

Use AI to orchestrate which tests to run (prioritize fast checks for dev commits, more expensive fuzzing/DAST in staging) and—where safe—initiate automated mitigations (feature flags, temporary blocks) with human approval.

8.5 Adversarial Robustness & Privacy

Research methods to harden models against poisoning and evasion, and methods (differential privacy, secure FL) to allow cross-org learning without sharing raw telemetry.

9. Ethical and Regulatory Considerations

Automated security systems interact with code owners, CI/CD tooling, and potentially production systems—so errors can have outsized effects. Responsible deployment requires human oversight, recording decisions, and compliance with data protection rules when telemetry contains personal data. Additionally, models that infer exploitability may affect disclosure practices and vendor liabilities; transparency and documented decision processes are advisable.

10. Conclusion

AI-enabled autonomous security testing and vulnerability prediction hold strong promise to make cloud-native DevSecOps both faster and safer. Recent research demonstrates advances in representation learning, exploit prediction, and automated triage, and industry adoption of automation is increasing. However, methodological weaknesses, data quality issues, evaluation biases, and operational challenges remain major impediments. Future work should prioritize robust, temporally-sound benchmarks, context-aware models with explainability, and safe integration patterns that preserve developer velocity while improving security outcomes.

References

The following references point to representative recent work, surveys, and industry resources cited in this review.

1. Risse, N. et al., *Top Score on the Wrong Exam: On Benchmarking in Machine Learning for Vulnerability Detection (ML4VD)* — evaluation and reproducibility concerns in ML for vulnerability detection.
2. ArXiv Survey, *A Survey on Vulnerability Prioritization: Taxonomy, Metrics, ...* — discusses CVSS limitations and ML-based prioritization approaches.
3. SANS/Sonatype DevSecOps Survey (2023) — industry adoption trends for security automation in CI/CD.
4. Mahbub, M. et al., *A Novel Vulnerability Exploit Prediction System Using the ...* (ACM/2025) — exploit prediction methodologies and EPSS comparisons.
5. ACM: *AI for DevSecOps: A Landscape and Future Opportunities* — an overview of AI techniques and integration patterns for DevSecOps.

Additional tools and dataset references such as public SAST/DAST tool lists and EPSS/NVD feeds were consulted during this review.