EFFICIENT WEB-VULNERABILITY DETECTION TECHNIQUE USING HYBRID SAST-DAST ANALYSIS AND MACHINE LEARNING

Kondo K.N., master's degree student, 467311

Belarusian State University of Informatics and Radioelectronics
Minsk, Republic of Belarus

Nasonova N. – PhD in Technical Sciences, Associate Professor

Abstract: This study presents a lightweight, interpretable machine learning framework that correlates the outputs of static (SAST) and dynamic (DAST) security testing tools to reduce false positives and prioritize true vulnerabilities. Trained on the OWASP Benchmark (2,740 test cases), the Random Forest model achieves F1=0.837, recall=0.943, and reduces false positives by 66.8% compared to standalone DAST. Validation on realistic vulnerable applications (crAPI, WebGoat) showed precision@10=0.70 and 80% alert reduction on crAPI, and precision@20=0.80 with 84% alert reduction on WebGoat. SHAP analysis provides transparency, enabling analysts to understand each prediction.

Keywords. SAST, WebGoat, DAST, OWASP.

Modern web applications face increasing security threats; however, existing SAST and DAST tools suffer from high false-positive rates and fragmented results, leading to alert fatigue. Although hybrid approaches exist, they often introduce performance overhead or lack interpretability [1, 2]. This study investigates the following research question: Can an interpretable machine learning framework effectively combine SAST and DAST outputs to significantly reduce false positives and prioritize vulnerabilities compared to standalone tools while maintaining high recall? To address this question, the study introduces an innovative framework that integrates SAST (Semgrep, SpotBugs, SonarQube) and DAST (OWASP ZAP) outputs via an interpretable machine learning model to intelligently correlate evidence and prioritize vulnerabilities.

This framework is a lightweight post-processing engine that operates outside of the production network. It ingests normalized outputs from SAST tools (that are run during development) and DAST tools run against a staging environment and provides a prioritized vulnerability report. As this is a decoupled architecture, it has zero runtime overhead on production services. Integration into a DevSecOps pipeline is simple: the framework can be called as a CI/CD piece after security scans, enabling an actionable risk-ranked list to surface without requiring any modifications to existing tooling or network architecture.

The OWASP Benchmark v1.2 (2,740 test cases, 1,415 vulnerable) was used as the training dataset. For each endpoint, a feature vector is constructed. The target label is 1 if the test case/endpoint is truly vulnerable, and 0 otherwise. Raw tool outputs are normalized into a feature matrix containing binary flags for each SAST tool (semgrep, spotbugs, sonarqube) and ZAP, plus ZAP severity counts. Engineered features include: num_tools (total tools flagging), agreement_semgrep_zap (a binary indicator of corroboration between Semgrep and ZAP), weighted_severity = 3·zap_high + 2·zap_medium + 1·zap_low, and zap_relevant (1 if any ZAP alert's CWE belongs to the 11 CWE classes present in the Benchmark, e.g., SQL injection, XSS, path traversal). A URL-to-source mapper aligns each ZAP alert with its corresponding source file using OpenAPI paths or routing definitions; SAST findings are linked similarly, enabling per-endpoint correlation. Multiple alerts or findings on the same endpoint are aggregated: ZAP severity counts are summed, and SAST flags are set to 1 if at least one finding exists in the mapped source file.

A Random Forest classifier was trained on an 80% stratified split, with hyperparameters tuned using 5-fold cross-validation. Probabilities were calibrated using Platt scaling (CalibratedClassifierCV with sigmoid) to ensure well-calibrated scores:

$$P(\mathrm{y}=1|f)=\frac{1}{1+\exp(A\cdot f+B)}, \quad (1)$$

where y is the binary vulnerability label (1 = vulnerable, 0 = safe), f is the raw score from the Random Forest, $\exp$ denotes the exponential function, and $A,\ B$ are parameters learned via 5-fold cross-validation.

A threshold of 0.30 was selected to maintain a recall of ≥90%. Model interpretability was provided using SHAP. Missing tool outputs (e.g., SpotBugs on non-Java services) default to 0; the model learns to handle such cases. Inference time was measured to quantify the lightweight nature of the framework. The overall architecture is shown in Figure 1.

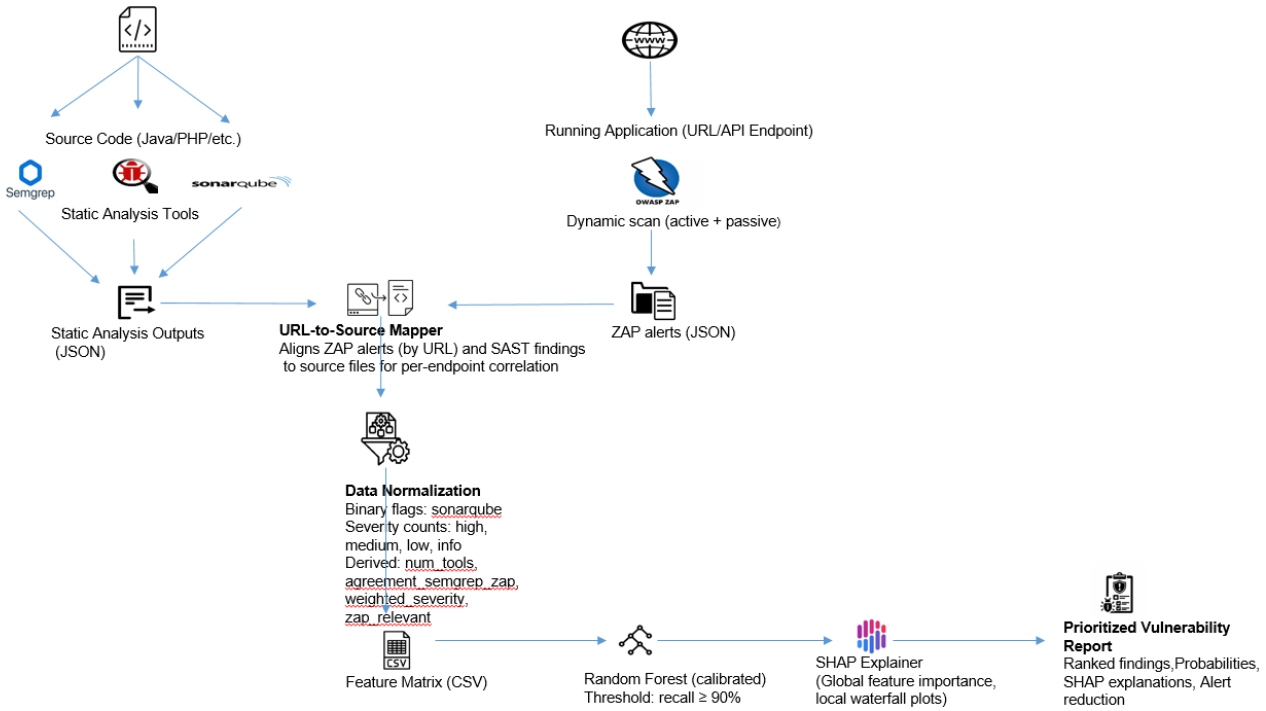

Figure 1 – Architecture of the hybrid vulnerability detection framework

On the held-out test set, which accounted for 20% of the OWASP Benchmark, the calibrated Random Forest did the following: Precision: 0.752, Recall: 0.943, F1-score: 0.837, False Positive Rate: 0.332.

Compared to the best standalone tool (Semgrep, F1=0.785), the hybrid model (F1=0.837) improves F1 by 5.2 pp and reduces false positives by 66.8% relative to ZAP's raw output. Ablation studies confirmed that ZAP features were the most important, while engineered features added measurable value. SHAP analysis (Figure 2) revealed that zap_medium and agreement_semgrep_zap as the most influential features.

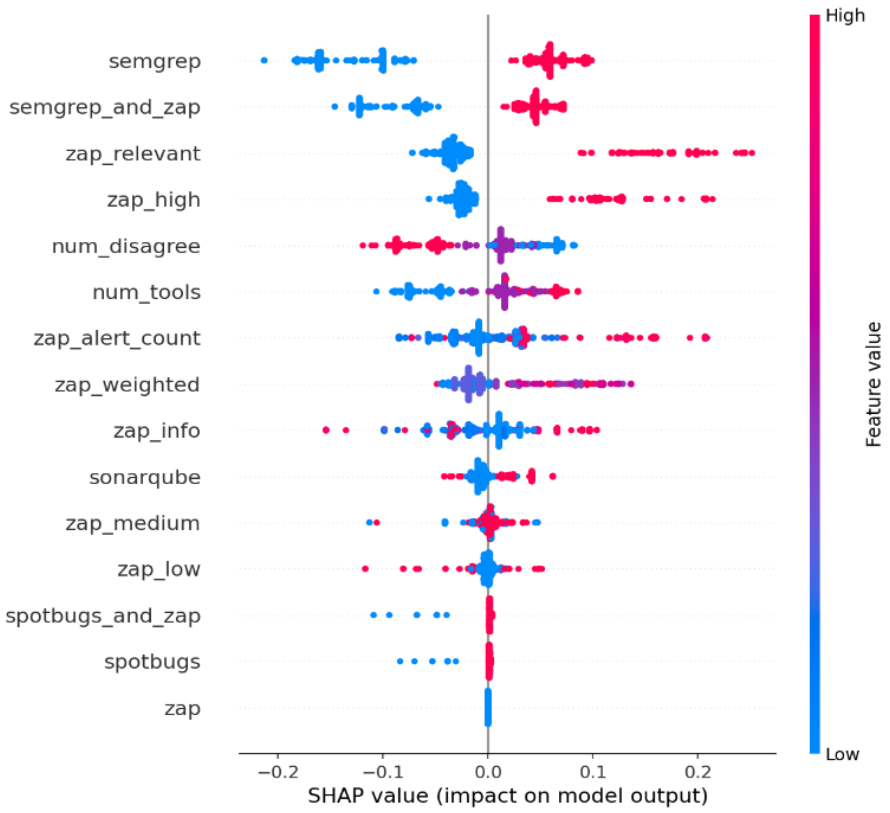

Figure 2 – SHAP summary plot showing feature importance

To assess generalizability, the framework was validated on realistic, vulnerable applications:

OWASP crAPI (modern API-first application): On a representative sample of 18 endpoints manually mapped to source code and labelled via challenge documentation (manual mapping was used only for ground-truth accuracy; in practice, mapping can be automated using route parsers), an authenticated ZAP scan produced 80 raw alerts. At a threshold of 0.30, the model flagged 16 endpoints as high-confidence, reducing the alert volume by 80%. Among the top-10 predictions, seven were true vulnerabilities (precision@10 = 0.70), and the model detected 13 of 14 known vulnerabilities (recall = 0.93). Inference time was <0.1 s, confirming lightweight operation.

OWASP WebGoat: For 126 endpoints, the model achieved precision@20 = 0.80 and reduced raw ZAP alerts by 84% (from 125 to 20), confirming consistent performance.

DVWA (PHP/MySQL): For the 10 core vulnerable endpoints, Semgrep produced 38 findings, and ZAP generated 109 alerts. The model assigned equally low probability (0.055) to all endpoints because the tool outputs were highly noisy; nevertheless, all 10 endpoints are truly vulnerable, and the top-10 predictions (all endpoints) gave precision@10 = 1.0. This result illustrates that the model remains conservative when evidence is weak, yet still ranks true vulnerabilities appropriately.

The hybrid framework successfully combines SAST and DAST evidence using an interpretable ML model, resulting in high recall while significantly reducing false positives. The explicit research question is answered positively: the model significantly reduces false positives (by 66.8–84%) while keeping the recall above 90% on both real-world and benchmark applications. Validation of realistic applications confirmed its practical value.

References

1. F. M. Tolosa et al., "On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications," Appl. Sci., vol. 10, no. 24, Art. no. 9119, Dec. 2020.

2. M. Keltek, R. Hu, M. Fani Sani, and Z. Li, "LSAST: Enhancing static vulnerability detection by integrating large language models with SAST scanners," arXiv, 2024. [Online]. Available: https://arxiv.org/pdf/2409.15735v2

3. S. Qadir, E. Waheed, A. Khanum, and S. Jehan, "Comparative evaluation of approaches & tools for effective security testing of Web applications," PeerJ Comput. Sci., vol. 11, p. e2821, 2025, doi: 10.7717/peerj-cs.2821.

4. G. Ibba, G. Orrù, G. Barabino, and D. R. Recupero, "A machine learning approach to vulnerability detection combining software metrics and topic modelling," Machine Learning with Applications, vol. 19, p. 100759, 2025.

5. M. J. S. Wicaksana and M. N. Fauzan, "Impact of Security Testing on Software Quality: A Systematic Literature Review," Sishifo, vol. 7, no. 2, pp. 186–200, 2025.