International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Secure Coding Analysis Tool

Priyanka K[1], Narendran A V[2], Shanjaiy S B[3],Sivanesh M[4], Vasanthbalan A[5],Yokesh G[6]

[1] Computer Science and Engineering(Cyber Security) , Sri Shakthi Institute of Engineering and Technology, Coimbatore, India
[2] Computer Science and Engineering(Cyber Security) , Sri Shakthi Institute of Engineering and Technology, Coimbatore, India

ABSTRACT:

The Secure Code Analysis Tool is an advanced, end-to-end automated platform designed to identify, explain, and report security vulnerabilities across diverse software codebases, including web applications, mobile apps, and APIs. Built with a focus on security integration within the Software Development Life Cycle (SDLC), SCAT leverages static code analysis to uncover a broad spectrum of vulnerabilities—ranging from injection flaws (like SQL or command injection) and insecure data storage to broken authentication, misconfigurations, and insufficient input validation. What makes SCAT uniquely effective is its ability to generate clear, human-readable explanations for each issue it identifies, enabling developers at all skill levels to understand not only the existence of the vulnerability but also its cause, impact, and possible exploitation methods.

Keywords: Vulnerability Detection,Static Code Analysis,Web Application Security,API Security

INTRODUCTION

This project focuses on enhancing software security by developing an AI-powered secure code analysis system. Modern software development, especially for web and mobile applications, often involves writing thousands of lines of code under tight deadlines. This increases the risk of introducing vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), hardcoded secrets, insecure APIs, and poor input validation. Traditional static analysis tools provide a base layer of defense, but they often lack context-awareness and adaptability. This project introduces an intelligent, machine learning-based approach to analyze source code, detect known vulnerabilities, and provide contextual explanations and mappings to known standards like OWASP and CVEs.

2. System Design and Architecture

Secure Code Analysis Tool’s architecture has four key components designed for comprehensive security analysis and reporting.

2.1 Static Code Analyzer (Python)

This module scans source code using pattern matching and AST parsing to identify vulnerabilities such as injection flaws, insecure data handling, and misconfigurations, mapping results to OWASP and CVE standards for accuracy.

2.2 Explanation Generator (NLP Engine)

Using prompt engineering, this component generates easy-to-understand descriptions of vulnerabilities, translating complex technical findings into simplified explanations that help developers quickly grasp and remediate issues.

2.3 Visualization Module (Heatmap Generator)

The heatmap generator visually represents the severity and distribution of vulnerabilities in the codebase using color-coded maps, enabling teams to easily identify risk-prone areas and prioritize fixes.

2.4 Report Engine (Export System)

This module produces structured reports in PDF or HTML formats containing detailed vulnerability data, severity ratings, remediation advice, and supports integration with CI/CD pipelines via APIs or webhooks.

3. Development Methodology

The tool is developed using Python 3.11, employing libraries such as Tree-sitter and Bandit for code analysis, Matplotlib for heatmaps, and Flask for the web interface, with prompt engineering ensuring clear vulnerability explanations and datasets drawn from OWASP, CVE, and industry standards.

3.1 Tools and Libraries

Python, Tree-sitter, Bandit, Matplotlib, Seaborn, and Flask are utilized to implement scanning, visualization, explanation generation, and API functionalities in a modular, scalable manner.

3.2 Prompt Engineering

Prompts are carefully designed to produce contextual, user-friendly vulnerability explanations, allowing the NLP engine to translate complex findings into easily understandable language tailored for developers.

3.3 Dataset and References

The system relies on curated security datasets including OWASP Top 10, CVE entries, MITRE CWE, and NIST guidelines to ensure comprehensive and up-to-date vulnerability detection and reporting.

4. Evaluation and Results

Evaluation against 25 open-source projects demonstrated 89% detection accuracy, with developer surveys rating explanation clarity at 4.6/5 and heatmap usage reducing code review times by approximately 32%, confirming the tool's practical effectiveness.

4.1 Vulnerability Detection Accuracy

The static analyzer achieved high accuracy in identifying known vulnerabilities compared to manual tools, supporting confident automation of code security checks.

4.2 Explanation Clarity

Developers found the generated vulnerability explanations clear and helpful, improving their understanding and speeding remediation.

4.3 Heatmap Visualization

Heatmaps provided intuitive visual risk assessments, enabling developers and managers to focus efforts on the most vulnerable code areas.

5. Use Cases

The tool aids developers by detecting security issues during coding, supports DevOps teams by integrating with CI/CD pipelines for continuous security checks, and serves as an educational resource for secure coding training.

5.1 Developer Code Review

Enables faster, more accurate detection of vulnerabilities during code development, reducing manual review burden.

5.2 Secure SDLC Integration

Integrates with development pipelines to enforce security gates and prevent vulnerable code from progressing.

5.3 Cybersecurity Education

Provides practical hands-on learning through vulnerability analysis, improving developer security awareness and skills.

6. Future Work

Planned enhancements include supporting additional programming languages, developing real-time IDE plugins, and incorporating AI-driven automated code fixes to further streamline secure software development.

6.1 Multi-language Support

Adding scanning capabilities for Java, JavaScript, Go, and other popular languages to broaden applicability.

6.2 Real-time IDE Plugin

Developing extensions for IDEs like VS Code and IntelliJ to provide inline vulnerability alerts during coding.

6.3 AI-Powered Auto-Fix Suggestions

Integrating large language models to suggest and apply secure code modifications automatically.

REFERENCES:

1.GimpelSoftware. Products Overview. http://www.gimpel.com/html/products.htm
2.Reasoning, Inc. Software Analysis Tools. http://www.reasoning.com
3.Klocwork. Static Code Analysis Tools. http://klocwork.com
4.V. R. Basili, S. Green, et al., "The Empirical Investigation of Perspective-Based Reading," Empirical Software Engineering, vol. 1, no. 2, 1996.
5.M. Young and R. N. Taylor, "Rethinking the Taxonomy of Fault Detection Techniques," in Proc. Conf. Software Engineering, pp. 53–62, 1989.
6.L. J. Osterweil, "Integrating the Testing, Analysis, and Debugging of Programs," in Proc. Symp. Software Validation, 1984.
7.N. Rutar, C. B. Almazan, and J. S. Foster, "A Comparison of Bug Finding Tools for Java," in Proc. IEEE Symp. Software Reliability Engineering (ISSRE), pp. 245–256, 2004.
8.R. Chillarege, I. S. Bhandari, et al., "Orthogonal Defect Classification – A Concept for In-Process Measurements," IEEE Trans. Software Engineering, vol. 18, no. 11, pp. 943–956, Nov. 1992.
9.IEEE, IEEE Standard Classification for Software Anomalies, IEEE Std. 1044-1993, 1993.
10.W. S. Humphrey, A Discipline for Software Engineering, Addison Wesley, 1995.
11.C. Jones, "Software Defect Removal Efficiency," Computer, vol. 29, no. 4, pp. 94–95, Apr. 1996.
12.C. Jones, Software Assessments, Benchmarks, and Best Practices, Addison-Wesley, May 2000.
13.B. Chess, "Improving Computer Security Using Extended Static Checking," in Proc. IEEE Symp. Security and Privacy, pp. 160–173, 2002.