iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.79296

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

Autonomous LLM Based Framework for End-To-End Software Vulnerability Detection and Verification

S. Priyanka[1], Mirthun Jai D.[2], Pranav Sundar S.[#3],Logesh R.[4]
Computer Science & Engineering (Cyber Security), K.L.N. College of Engineering, Madurai, Tamil Nadu, India

Abstract: ***Software security vulnerabilities remain one of the leading causes of data breaches and cyberattacks in modern enterprise systems. Traditional static analysis tools rely primarily on rule-based detection techniques and often produce high false-positive rates while failing to detect contextual vulnerabilities. This paper presents an Autonomous Threat-Hunting AI Agent designed to automatically detect security vulnerabilities in source code repositories using a hybrid approach combining pattern-based detection, transformer-based machine learning models, and Retrieval-Augmented Generation (RAG) supported by Common Vulnerabilities and Exposures (CVE) knowledge bases. The system integrates FastAPI for backend processing, Streamlit for visualization, and fine-tuned transformer models for semantic vulnerability classification. Experimental evaluation demonstrates improved detection accuracy, reduced false positives, and real-time scanning capability across multiple programming languages. The proposed system provides automated remediation suggestions and severity scoring, making it suitable for enterprise-scale deployment and integration with CI/CD pipelines.***

Keywords: Vulnerability Detection, Static Code Analysis, Transformer Models, Cybersecurity, Threat Hunting, Retrieval-Augmented Generation, Secure Coding.

I. INTRODUCTION

Software vulnerabilities remain one of the primary causes of cyberattacks and data breaches in modern enterprise systems. With the rapid growth of web applications, cloud computing platforms, and distributed software architectures, attackers frequently exploit weaknesses such as SQL injection, command injection, cross-site scripting, insecure credential storage, and hardcoded secrets. Detecting these vulnerabilities during early stages of development is essential to ensure secure deployment and to reduce the risk of financial loss and data compromise in production environments. Traditional software security testing methods rely heavily on manual inspection and static rule-based analysis tools, which often lack contextual understanding of source code and generate a large number of false positives.

These systems are limited in adaptability and fail to detect complex vulnerabilities that depend on program semantics and developer intent. As software complexity increases, there is a growing need for intelligent automated frameworks capable of improving detection accuracy while supporting scalable and continuous security assessment. To address these challenges, this paper proposes an Autonomous LLM-Based Framework for End-to-End Software Vulnerability Detection and Verification that integrates rule-based detection techniques with transformer-based semantic analysis and Retrieval-Augmented Generation (RAG) supported by CVE knowledge bases.

The proposed system performs automated repository-level scanning, assigns severity scores to detected vulnerabilities, and generates actionable remediation suggestions through a scalable FastAPI-based architecture, making it suitable for DevSecOps workflows and enterprise cybersecurity applications.

II. LITERATURE REVIEW

1) E. M. Pasca, D. Delinschi, R. Erdei, and O. Matei, in *"LLM-Driven, Self-Improving Framework for Security Test Automation: Leveraging Karate DSL for Augmented API Resilience"* (2025), propose an LLM-driven framework for automated security testing that integrates Retrieval Augmented Generation (RAG) for contextual vulnerability analysis. Their work demonstrates that combining AI reasoning with self-improving mechanisms can enhance detection accuracy while reducing reliance on manually written security rules. Additionally, the framework improves explainability by providing context-aware insights into detected vulnerabilities.

2) A. Barabanov, D. Dergunov, D. Makrushin, and A. Teplov, in *"Automatic Detection of Access Control Vulnerabilities via API Specification Processing"* (2022), present a deep learning-based approach for detecting access control vulnerabilities using structured API specifications and program analysis. Their study shows that machine learning models can effectively identify complex vulnerability patterns with high accuracy. However, the approach lacks contextual reasoning and explainability, limiting its effectiveness in understanding the root cause of vulnerabilities.
3) M. Bhatt et al., in *"Purple Llama CyberSecEval: A Secure Coding Benchmark for Language Models"* (2023), introduce a comprehensive benchmark designed to evaluate security risks in code generated by large language models. Their work demonstrates that advanced LLMs frequently produce insecure or vulnerable code despite achieving functional correctness. The study uses automated testing and static analysis across multiple programming languages, highlighting the need for improved security-focused training in LLMs.
4) M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, in *"An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation"* (2024), conduct a large-scale empirical study on the effectiveness of LLMs in generating unit tests. Their findings show that LLM-generated tests achieve higher code coverage and produce well-structured, readable outputs. However, the study primarily focuses on functional testing and does not emphasize security aspects or vulnerability detection.
5) M. L. Siddiq, J. C. D. S. Santos, S. Devareddy, and A. Müller, in *"SALLM: Security Assessment of Generated Code"* (2024), propose a framework to systematically evaluate the security of code generated by large language models. Their work demonstrates that functional correctness alone is insufficient for secure software development, as LLMs often generate insecure coding patterns. The study uses security-focused datasets and evaluation metrics to identify common vulnerabilities, emphasizing the importance of integrating security awareness into AI-based code generation.

.

III. PROPOSED METHODOLOGY

A. *Input Code Ingestion*

The system accepts compressed software repositories through a web interface and API endpoints for secure submission. It recursively scans project directories to identify relevant source code files across multiple programming languages.

B. *Code Parsing and Preprocessing*

Uploaded source files are filtered and divided into structured code snippets while preserving contextual metadata. Irrelevant or non-code content is removed to improve processing efficiency and prepare inputs for AI analysis.

C. *RAG-Based CVE Reference Integration*

A Retrieval-Augmented Generation module retrieves relevant vulnerability information from a CVE-style knowledge base. This improves explainability by linking detected vulnerabilities with standardized security advisories and attack patterns.

D. *AI-Based Vulnerability Analysis*

Transformer-based Large Language Models analyze code semantics to detect security vulnerabilities with contextual understanding. This approach reduces false positives and improves detection accuracy compared to traditional rule-based methods.

E. *Severity Scoring*

Detected vulnerabilities are assigned numerical severity scores based on type and confidence level of analysis. The system classifies risks into high, medium, and low categories to support prioritized remediation.

F. *Automated Remediation Suggestions*

The framework generates secure coding recommendations for identified vulnerabilities using AI-assisted reasoning.

IV. SYSTEM ARCHITECTURE

A. *Repository Upload Interface*

Users upload compressed project repositories through a Streamlit-based web interface or REST API endpoints. The interface securely transfers files to the backend processing module for automated vulnerability scanning.

B. Code Extraction And Parsing Module

The system extracts source files from uploaded repositories and filters relevant programming language files. Parsed code snippets are structured with contextual metadata for semantic vulnerability analysis.

C. Rag-Based CVE Knowledge Integration

A Retrieval-Augmented Generation module retrieves related vulnerability references from a CVE knowledge base. This enhances explainability by associating detected issues with known security threats and attack patterns.

D. Ai Vulnerability Detection Engine

Transformer-based LLM models analyze code semantics to detect security weaknesses beyond rule-based matching. The hybrid detection approach improves accuracy while reducing false positives in multi-language repositories.

E. Severity Scoring And Risk Classification Module

Detected vulnerabilities are assigned severity scores based on type, confidence level, and contextual relevance. Issues are categorized into high, medium, and low severity levels for prioritized remediation.

F. Automated Fix Recommendation Module

The system generates remediation suggestions using secure coding practices and AI-assisted reasoning techniques. These recommendations help developers quickly resolve vulnerabilities and strengthen application security.

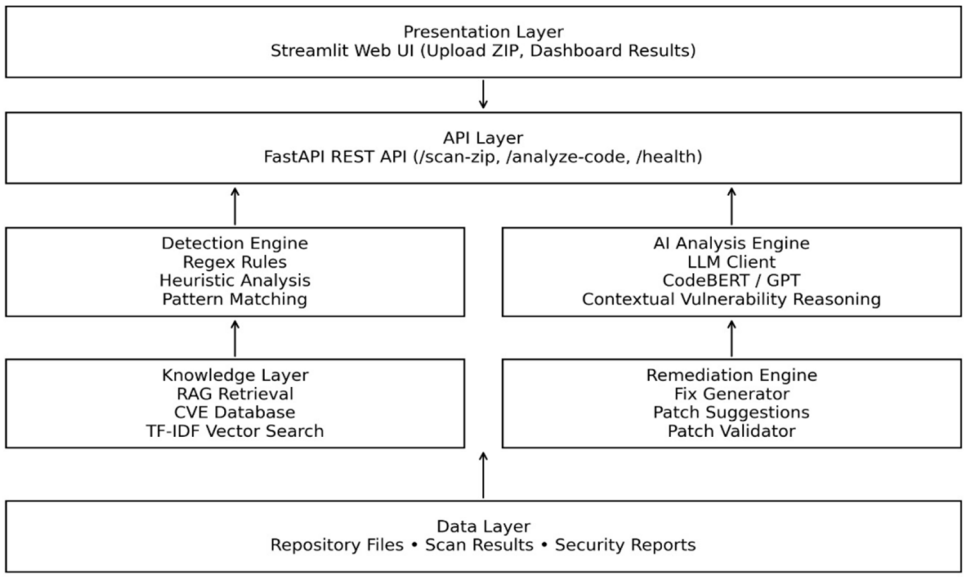


Fig. 1. Architecture Diagram

V. PERFORMANCE EVALUATION

This module evaluates the effectiveness of the proposed Autonomous Threat-Hunting AI Agent by measuring vulnerability detection accuracy and False Positive Rate (FPR) across multiple security patterns such as SQL injection, plaintext credential storage, command injection, hardcoded secrets, and cross-site scripting vulnerabilities. The evaluation analyzes how accurately the system identifies insecure coding patterns from uploaded repositories and classifies them into relevant vulnerability categories using hybrid rule-based and transformer-based semantic analysis techniques.

Accuracy is calculated using the following formula:

$$\textbf{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)} \quad (1)$$

Where:
TP = True Positives,
TN = True Negatives,
FP = False Positives,
FN = False Negatives.
True Positives represent correctly detected vulnerabilities and False Negatives represent security weaknesses missed by the system during repository scanning.

The system performance is further analyzed using detection metrics such as:

$$\textbf{False Positive Rate (FPR)} = \frac{(FP)}{(FP + TN)} \quad (2)$$

Where:
The False Positive Rate represents incorrectly detected vulnerabilities in secure code segments. A lower FPR indicates improved reliability of the detection framework and reduces unnecessary remediation effort for developers.

TABLE I
PERFORMANCE COMPARISON

Metric	Baseline Model	Proposed Model
Accuracy	0.853	0.921
FPR	0.182	0.078

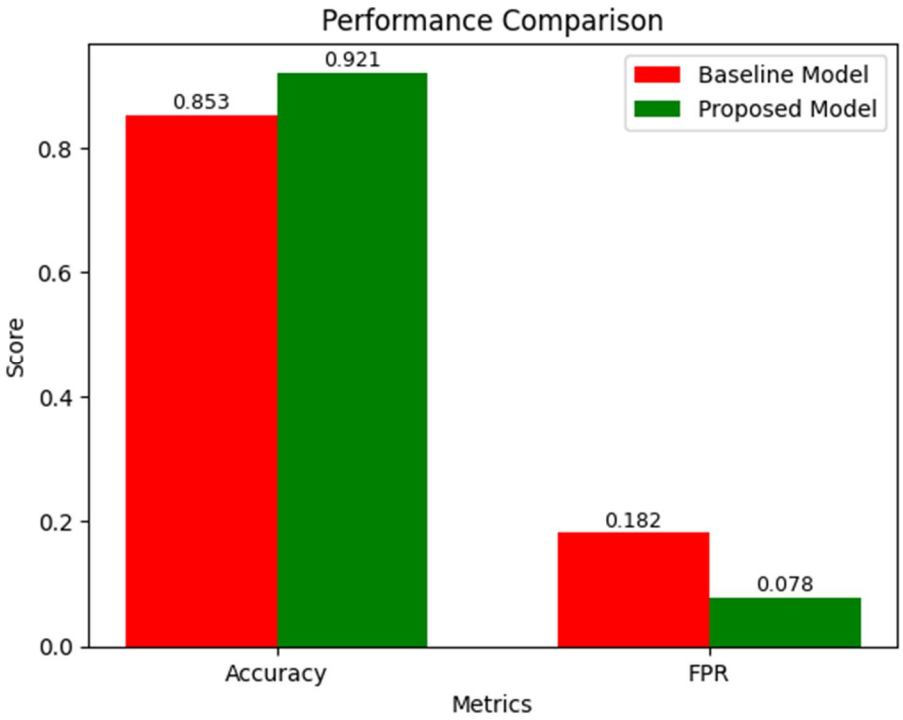


Fig. 2. Performance Comparison

VI. SYSTEM ANALYSIS

The proposed Autonomous Threat-Hunting AI Agent is evaluated based on its capability to automatically analyze source code repositories and detect security vulnerabilities using hybrid AI-assisted techniques. The system integrates a Streamlit-based user interface, a FastAPI backend processing engine, transformer-based semantic analysis, and a Retrieval-Augmented Generation (RAG) module supported by CVE knowledge references. Source code files are extracted and preprocessed to identify insecure patterns such as SQL injection risks, plaintext credential storage, command execution vulnerabilities, and hardcoded secrets. The extracted features are analyzed using both rule-based detection mechanisms and transformer-based models to improve classification accuracy and reduce false positives. The system enhances operational efficiency by automating repository scanning, severity scoring, and remediation suggestion generation within a unified framework. Additionally, the modular architecture enables integration of updated vulnerability rules and advanced AI models to adapt to evolving cybersecurity threats. Performance evaluation is conducted using detection accuracy and False Positive Rate (FPR) to measure system reliability. Overall, the proposed architecture ensures scalability, flexibility, and efficient resource utilization, making it suitable for secure software development lifecycle integration and enterprise DevSecOps environments.

VII. EXPECTED RESULTS

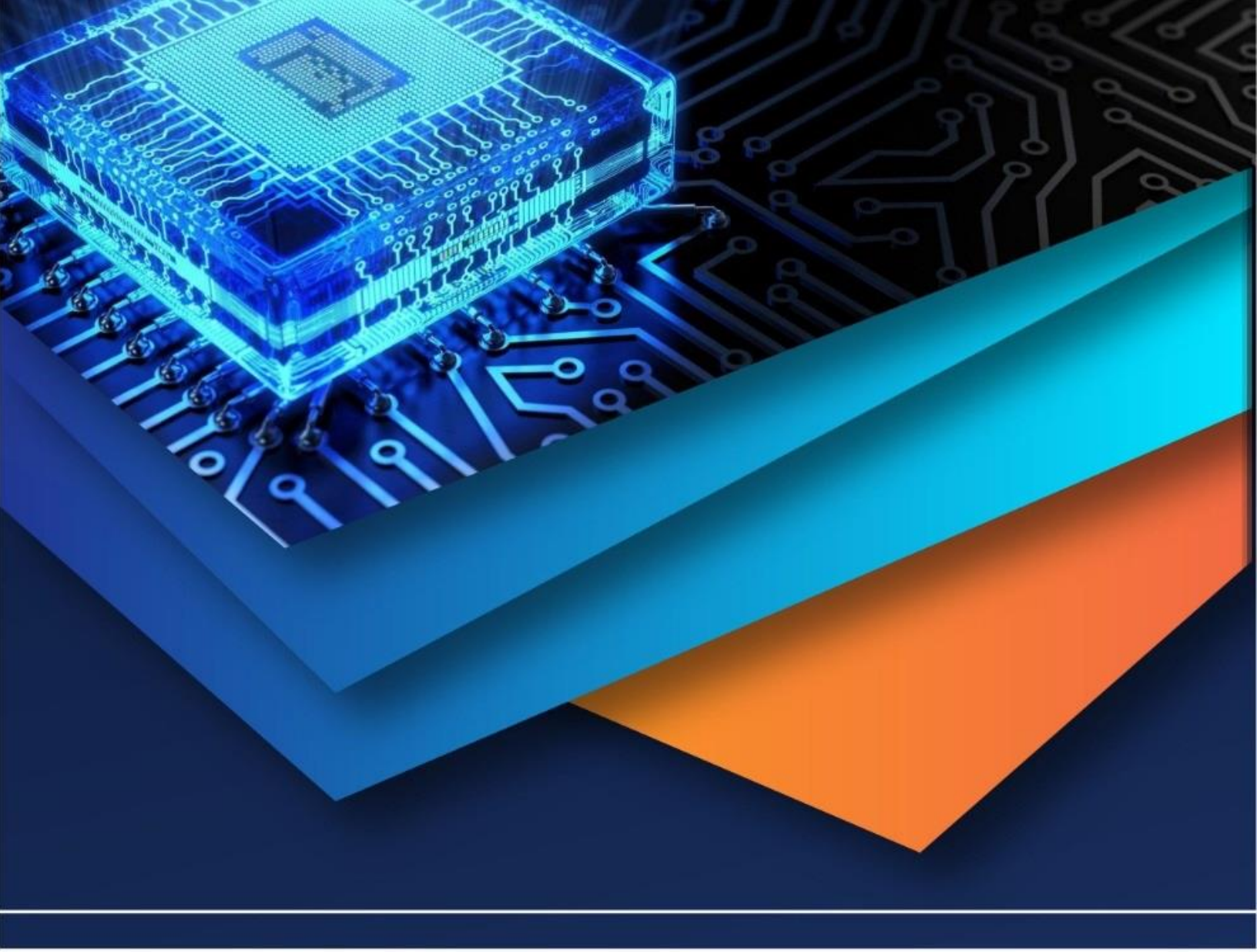
Future improvements to the proposed system include integration with graph neural networks for advanced dependency analysis, support for zero-day vulnerability detection using adaptive learning techniques, and real-time monitoring within CI/CD pipelines for continuous security assessment. Additionally, expanding multilingual vulnerability datasets will improve detection capability across diverse programming environments, while deployment on distributed cloud security platforms will enhance scalability and performance. These enhancements will further strengthen detection accuracy and enable the system to support enterprise-level secure software development applications.

VIII. CONCLUSION

This paper presented an Autonomous Threat-Hunting AI Agent for intelligent vulnerability detection using transformer-based analysis and RAG-supported CVE knowledge retrieval. The proposed system improves detection accuracy while reducing false positives and providing automated remediation suggestions. It supports multi-language scanning and real-time analysis, making it suitable for modern DevSecOps environments. Overall, the framework contributes to scalable and automated AI-assisted software security solutions.

REFERENCES

[1] E. M. Pasca, D. Delinschi, R. Erdei, and O. Matei, “LLM-Driven, Self-Improving Framework for Security Test Automation: Leveraging Karate DSL for Augmented API Resilience,” 2025.
[2] A. Barabanov, D. Dergunov, D. Makrushin, and A. Teplov, “Automatic Detection of Access Control Vulnerabilities via API Specification Processing,” Voprosy Kiberbezopasnosti, 2022.
[3] M. Bhatt et al., “Purple Llama CyberSecEval: A Secure Coding Benchmark for Language Models,” arXiv:2312.04724, 2023.
[4] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, “An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation,” 2024.
[5] M. L. Siddiq, J. C. D. S. Santos, S. Devareddy, and A. Müller, “SALLM: Security Assessment of Generated Code,” in Proc. IEEE/ACM Int. Conf. Automated Softw. Eng. Workshops, 2024
[6] G. Baye, F. Hussain, A. Oracevic, R. Hussain, and S. M. A. Kazmi, “API security in large enterprises: Leveraging machine learning for anomaly detection,” in Proc. Int. Symp. Netw., Comput. Commun. (ISNCC), Oct. 2021, pp. 1–6
[7] M. Sowmya, A. J. Rai, V. Spoorthi, M. Irfan, P. B. Honnavalli, and S. Nagasundari, “API traffic anomaly detection in microservice architecture,” in Proc. IEEE/ACM Int. Symp. Cluster, Cloud Internet Comput. Workshops (CCGridW), May 2023, pp. 206–213.
[8] A. Aumpansub and Z. Huang, “Detecting software vulnerabilities using neural networks,” in Proc. Int. Conf. Mach. Learn. Comput., ACM, Feb. 2021, pp. 166–171, doi: 10.1145/3457682.3457707.
[9] Y. Huang, C. Shi, J. Lu, H. Li, H. Meng, and L. Li, “Detecting broken object-level authorization vulnerabilities in database-backed applications,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Dec. 2024, pp. 2934–2948, doi: 10.1145/3658644.3690227.
[10] A. G. Mirabella, A. Martin-Lopez, S. Segura, L. Valencia-Cabrera, and A. Ruiz-Cortés, “Deep learning-based prediction of test input validity for RESTful APIs,” in Proc. IEEE/ACM Int. Workshop Deep Learn. Test. (DeepTest), Jun. 2021, pp. 9–16.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY