© Volume 2, Issue 10, October 2026 | JATIR
e-ISSN: 2839-5786, p-ISSN: 3139-8464

AutoPatch AI: An Autonomous Web Application Vulnerability Scanner and AI-Powered Code Remediation Engine

[1]Prof. Sayali Gaikwad, [2]Siddhi Ghogare, [3]Tejaswini More
[4]Nikita Chitalkar,[5]Trupti Ganeshkar
[1,2,3,4,5]*Information Technology, Parvatibai Genba Moze College, Pune, India*

***Abstract*—Web applications are increasingly exposed to security vulnerabilities such as SQL injection, cross-site scripting, command injection, and insecure access-control weaknesses. Traditional Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) techniques provide important capabilities for identifying such vulnerabilities, but the subsequent analysis and remediation process can still require substantial developer effort. Recent research has introduced artificial intelligence, machine learning, deep learning, and large language models (LLMs) for vulnerability detection, prioritization, autonomous security assessment, and automated code remediation. However, existing studies often address these capabilities as separate tasks, while practical integration of detection, code-context extraction, patch generation, patch validation, and repository-level remediation remains a significant challenge. This review examines the evolution of AI-assisted web application vulnerability detection and automated remediation and analyses representative approaches in the existing literature. Based on this analysis, the paper discusses AutoPatch AI as an integrated workflow that connects SAST/AST and DAST-based vulnerability detection with code-context extraction, LLM-based patch generation, isolated Docker-based validation, post-patch re-scanning, code-diff generation, and developer-approved GitHub or GitLab pull requests. The review further identifies key challenges related to patch reliability, validation, generalization, benchmarking, and human oversight. The proposed workflow provides a practical direction for connecting vulnerability discovery with secure and verifiable code remediation within a DevSecOps-oriented development process.**

I. INTRODUCTION

WEB applications have become an essential part of modern digital services, but their increasing complexity has also expanded the attack surface for vulnerabilities such as SQL injection, cross-site scripting, command injection, and insecure access-control weaknesses. Traditional Static

Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) techniques are widely used to identify such vulnerabilities, but analysing and remediating detected issues can still require considerable developer effort.

Recent research has applied artificial intelligence (AI), machine learning (ML), deep learning, and large language models (LLMs) to vulnerability detection, prioritization, autonomous security assessment, and automated code remediation [1]. Autonomous vulnerability scanners have also explored LLM-based agents for planning scans, interpreting results, and prioritizing security findings [2]. These developments show the potential of AI to support different stages of the vulnerability management process.

However, existing studies often focus on individual tasks such as vulnerability detection, autonomous scanning, or patch generation. The integration of vulnerability detection, code-context extraction, AI-assisted patch generation, isolated patch validation, post-patch re-scanning, and repository-level integration remains an important practical challenge. This review examines these research directions and discusses AutoPatch AI as an integrated workflow for connecting vulnerability discovery with secure and verifiable code remediation.

The remainder of the paper is organized as follows. Section II reviews related work. Section III presents a comparative analysis and identifies the research gap. Section IV describes the proposed AutoPatch AI framework. Section V discusses research gaps and challenges. Section VI presents future directions, and Section VII concludes the paper.

II. RELATED WORK

A. Traditional SAST and DAST

Static Application Security Testing (SAST) analyzes source code or intermediate representations without executing the application, while Dynamic Application Security Testing (DAST) evaluates running applications through controlled testing. These approaches are widely used for identifying common web vulnerabilities, but their findings may still require manual analysis and remediation.

B. AI-Based Vulnerability Detection

Recent studies have applied machine learning and deep learning techniques to source-code analysis, vulnerability classification, vulnerability prioritization, and security assessment [1]. These approaches can assist in identifying patterns that are difficult to capture using only traditional rule-based techniques. However, challenges such as data quality, generalization, and reproducible evaluation remain.

C. LLM-Based Vulnerability Analysis and Remediation

Large language models have been explored for vulnerability analysis, code explanation, and automated patch generation [1]. LLMs can use vulnerability descriptions and relevant code context to suggest security fixes. However, generated patches require validation because syntactically

correct changes may not completely remove the vulnerability or may introduce unintended behaviour.

D. Autonomous Web Vulnerability Scanning

Recent work has investigated autonomous vulnerability scanners that use LLM-based agents to plan security scans, invoke scanning tools, interpret findings, and adjust the scanning process [2]. Such systems demonstrate the potential of agent-based approaches for improving automated security assessment, but their primary focus is vulnerability discovery and scan orchestration rather than complete code-level remediation.

E. Existing Systematic Reviews

Malkawi and Alhajj [1] systematically reviewed AI-powered vulnerability detection and patch management research and identified challenges including generalization, robustness, explainability, benchmarking, and human oversight. Their findings indicate that AI-assisted vulnerability management is promising but still requires practical and reproducible approaches for integrating detection, remediation, and validation.

III. SYSTEM ARCHITECTURE

A. Overview

The proposed AutoPatch AI framework follows a layered architecture, shown in Fig. 1. A user connects a GitHub repository through the web interface. The backend accesses the selected repository and coordinates vulnerability scanning, code-context extraction, AI-based patch generation, isolated validation, and repository integration.

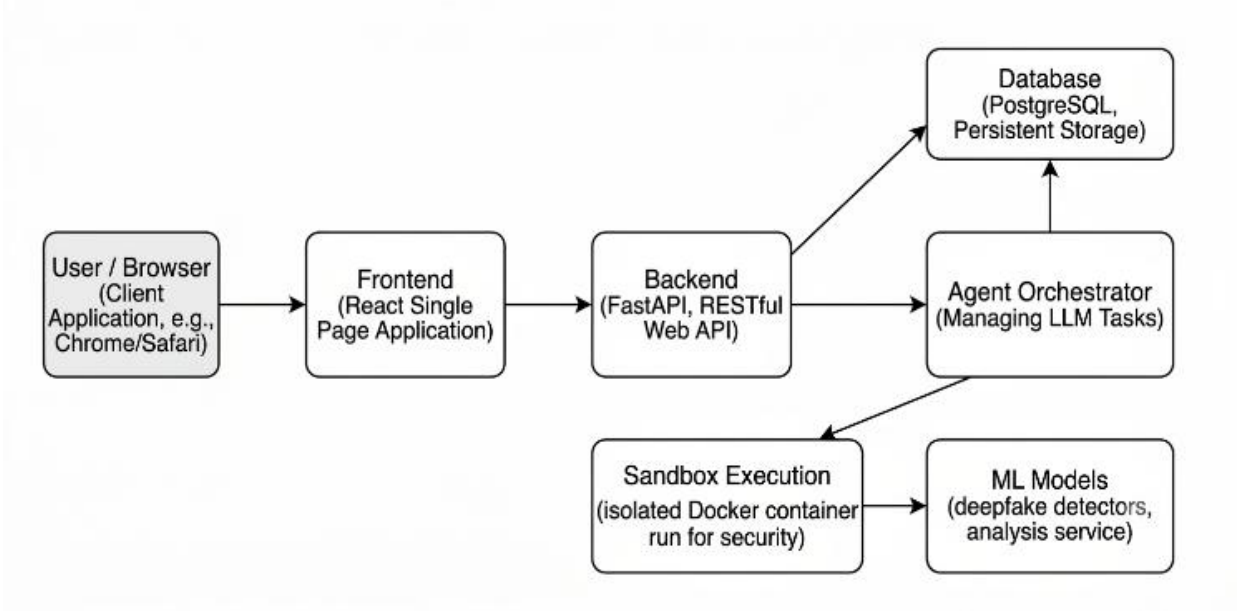

Fig. 1. System architecture of the AutoPatch AI framework

B. Frontend

The web interface allows users to connect their GitHub account, select a repository, start security scans, view detected vulnerabilities, inspect generated code changes, and review patch validation results.

C. Backend

The backend manages repository access, scan requests, vulnerability information, code context, patch generation requests, validation status, and communication between the system components. It acts as the central coordinator of the remediation workflow.

D. Vulnerability Scanner

The scanner combines Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST). SAST/AST analysis examines source-code structure, while DAST evaluates the running application to identify security vulnerabilities such as SQL injection, cross-site scripting, and command injection.

E. Code Context Extraction

When a vulnerability is detected, the system identifies the affected file, function, vulnerable code section, and relevant surrounding context. This information is prepared as input for the remediation component.

F. LLM Remediation Engine

The LLM remediation engine receives the vulnerability details and extracted code context and generates a candidate security patch. The generated patch is treated as a candidate change and is not directly applied to the main repository.

G. Sandbox Validation

The candidate patch is executed in an isolated Docker environment. Unit and regression tests are performed, followed by a security re-scan to determine whether the identified vulnerability has been resolved. If validation fails, the patch can be regenerated and tested again. Repeated failure results in manual developer review.

H. Repository Integration

After successful validation, the system generates a code diff for developer inspection. The approved change can then be submitted through a GitHub or GitLab pull request, allowing the developer to review and merge the remediation into the repository.

IV. DETECTION METHODOLOGY

The detection pipeline consists of the following stages:

• Repository Access: the selected GitHub repository is authorized and accessed using the required permissions.

• Static Analysis: the source code is analyzed using SAST and AST-based techniques to identify potential vulnerabilities.

• Dynamic Testing: controlled security requests are performed on the running application to detect vulnerabilities during execution.

• Vulnerability Identification: findings from SAST and DAST are analyzed and classified according to vulnerability type, affected file, and code location.

• Context Extraction: the vulnerable code, affected function, and relevant surrounding context are extracted for further remediation.

The detected vulnerability and extracted code context are then provided to the LLM-based remediation component to generate a candidate security patch. The generated patch is subsequently validated in an isolated environment and re-scanned to verify whether the vulnerability has been resolved.

V. IMPLEMENTATION

The frontend is developed using React and Vite, while the backend is implemented in Python using FastAPI. PostgreSQL provides persistence for repository information, vulnerability findings, scan history, and patch validation results. The SAST/AST scanner, DAST testing, LLM-based remediation, and validation components are integrated through the backend. Docker containers provide an isolated environment for executing candidate patches and regression tests. The validated changes are presented as code diffs for developer review and can be submitted to GitHub or GitLab through a pull request.

VI. EXPERIMENTAL EVALUATION

A. Datasets and Metrics

We plan to evaluate the system using vulnerable web application benchmarks and controlled test applications containing SQL Injection, Cross-Site Scripting, and Command Injection vulnerabilities. Performance can be measured using detection accuracy, precision, recall, F1-score, false-positive rate, patch success rate, and validation time. Re-scanning is used to verify whether

the generated patch removes the identified vulnerability, while regression testing is used to assess whether the patch affects existing functionality.

B. Results

Table I summarizes the evaluation results. The values are to be filled in from the experiments.

TABLE I AUTOPATCH AI EVALUATION PERFORMANCE

Evaluation Metric	Value
Detection Accuracy	To be filled
Precision	To be filled
Recall	To be filled
F1-score	To be filled
Patch Success Rate	To be filled
Re-scan Verification Rate	To be filled
Regression Test Success Rate	To be filled
Average Validation Time	To be filled

VII. DISCUSSION AND LIMITATIONS

AI-generated security patches may not always completely remove a vulnerability or may introduce unintended changes in application behavior. The effectiveness of the remediation process can also vary across programming languages, frameworks, and codebases. Docker-based sandbox validation provides an isolated environment for testing candidate patches but adds execution overhead. In addition, vulnerability detection and patch generation may produce false positives or incorrect fixes. Therefore, generated patches should be validated and reviewed by developers before integration into the main repository.

VIII. CONCLUSION AND FUTURE WORK

This paper presented AutoPatch AI, an integrated workflow for web application vulnerability detection and AI-assisted code remediation. The framework combines SAST/AST and DAST-based detection with code-context extraction, LLM-based patch generation, Docker-based validation, post-patch re-scanning, and developer-controlled repository integration. The approach aims to reduce the gap between vulnerability discovery and secure code remediation. Future work includes supporting additional vulnerability classes, improving patch validation, evaluating the

framework across different codebases and programming languages, and developing standardized benchmarks for AI-generated security patches.

ACKNOWLEDGMENT

The authors would like to thank [supervisor/institution] for their guidance, support, and valuable suggestions throughout the development of this review paper.

REFERENCES

[1]. M. Malkawi and R. Alhajj, "AI-Powered Vulnerability Detection and Patch Management in Cybersecurity: A Systematic Review of Techniques, Challenges, and Emerging Trends," Machine Learning and Knowledge Extraction, vol. 8, no. 1, p. 19, 2026, doi: 10.3390/make8010019.

[2]. A. A. Shaikh and A. J. Chinchawade, "Smart Web Vulnerability Scanner," International Journal of Engineering Research & Technology (IJERT), vol. 14, no. 12, Dec. 2025, paper ID IJERTV14IS120471.

[3]. Q. Zhang, C. Fang, Y. Xie, Y. Ma, W. Sun, Y. Yang, and Z. Chen, "A Systematic Literature Review on Large Language Models for Automated Program Repair," arXiv preprint arXiv:2405.01466, 2024.

[4]. L. Zhang, A. Singhal, Q. Zou, X. Sun, and P. Liu, "Can AI Fix Buggy Code? Exploring the Use of Large Language Models in Automated Program Repair," IEEE Computer, vol. 58, no. 7, pp. 122–128, 2025, doi: 10.1109/MC.2025.3527407.

[5]. A. Sajadi, K. Damevski, and P. Chatterjee, "How Safe Are AI-Generated Patches? A Large-scale Study on Security Risks in LLM and Agentic Automated Program Repair on SWE-bench," arXiv preprint arXiv:2507.02976, 2025.

[6]. B. Yang, Z. Cai, F. Liu, B. Le, L. Zhang, T. F. Bissyandé, Y. Liu, and H. Tian, "A Survey of LLM-based Automated Program Repair: Taxonomies, Design Paradigms, and Applications," arXiv preprint arXiv:2506.23749, 2025.