International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Automated Code Vulnerability Reviewing

Mr. Dishant Dhanawade
Mr. Samrat Ubhe Mr. Aditya Kate Mr. Vivek Mane Mr. Aaryan Tambe Mr. Vedant Nagane
Lecturer; Diploma Students
Department of Artificial Intelligence & Machine Learning (AIML)
Zeal Polytechnic, Pune, Maharashtra, India

Abstract — Software vulnerabilities can introduce serious security risks in applications, making code review an essential part of the software development process. However, manual code inspection can be time-consuming, while conventional automated tools may focus on specific categories of issues and provide fragmented results. This paper proposes an automated code vulnerability reviewing framework based on a role-based multi-agent architecture integrated with GitHub. The proposed system retrieves repository information, identifies supported programming languages, examines project structure and dependencies, and prepares relevant source-code context for analysis. It coordinates three specialized agents: a Security Auditor Agent to identify potential vulnerabilities, a Software Architect Agent to examine code structure and design-related concerns, and a Review Moderator Agent to consolidate findings, remove duplicates, resolve conflicting observations, and prioritize reported issues. Reviews can be initiated automatically through supported GitHub events or manually by developers. The framework also proposes a locally executable environment with a web-based dashboard and command-line interface (CLI), allowing developers to monitor review activities and access structured reports containing issue descriptions, code locations, supporting evidence, and suggested corrections. The system is designed to provide recommendations without automatically modifying source code. This paper presents the proposed architecture, workflow, and evaluation approach; its effectiveness remains subject to future implementation and experimental validation.

Keywords: Automated code vulnerability reviewing, Multi-agent systems, GitHub integration, Artificial intelligence, Large language models, Software security, Automated code review.

I. INTRODUCTION

Software security has become an important concern in modern software development. As applications continue to grow in complexity, developers must ensure that their source code is not only functional but also secure and reliable. Vulnerabilities in source code can expose applications to unauthorized access, data leakage, and other security threats. Code review plays an important role in identifying such weaknesses before code changes are integrated into a project.

Traditional code review generally involves developers manually examining source code, commits, and pull requests to identify programming errors and security vulnerabilities. Although manual review provides valuable human judgment, it can become time-consuming when projects contain large codebases or receive frequent updates. Automated tools such as static analysis tools, linters, and dependency scanners help reduce this effort. However, these tools may focus on specific categories of issues and often produce results that require additional interpretation and verification.

Recent developments in Artificial Intelligence (AI), particularly Large Language Models (LLMs), have introduced new possibilities for automated code analysis. LLMs can examine source code, explain potential vulnerabilities, and suggest possible improvements. However, relying on a single general-purpose model may result in missed issues, inaccurate explanations, or findings that lack sufficient supporting evidence. A structured approach that separates code analysis responsibilities can help organize the reviewing process.

To address these challenges, this paper proposes an Automated Code Vulnerability Reviewing framework based on a role-based multi-agent architecture integrated with GitHub. The framework consists of three specialized agents: The Security Auditor Agent, which focuses on identifying potential security vulnerabilities; the Software Architect Agent, which examines code structure, organization, and maintainability; and the Review Moderator Agent, which consolidates the observations, removes duplicate findings, handles conflicting results, and prioritizes reported issues.

The proposed framework is designed to retrieve repository information, identify supported programming languages, examine project structure and dependencies, and prepare relevant source-code context for analysis. Reviews can be initiated automatically through supported GitHub events or manually by developers. The system also provides a locally executable environment with a web-based dashboard and Command-Line Interface (CLI), allowing developers to monitor agent activities and access structured review reports.

The generated reports are intended to contain issue descriptions, relevant code locations, supporting evidence, priority levels, and suggested corrections. The system provides recommendations rather than automatically modifying project source code, keeping developers responsible for reviewing and applying any suggested changes.

This paper presents the proposed framework, its architecture, operational methodology, and an evaluation approach for future implementation. Its effectiveness will need to be established through implementation and experimental validation.

II. RELATED WORK

Recent research has explored the use of Artificial Intelligence (AI), particularly Large Language Models (LLMs), for automated code review and software vulnerability detection. These studies provide a foundation for developing systems that can analyze source code, identify potential security weaknesses, and assist developers during software review.

Yu et al. studied the application of LLMs in security code review and discussed their capabilities, challenges, and factors

influencing their performance [1]. Their work highlights that the effectiveness of LLM-based security review can depend on factors such as the quality of prompts, source-code characteristics, and the information provided to the model. This indicates the importance of supplying relevant code context and validating the findings generated by AI-based reviewing systems.

Jensen et al. investigated the use of LLMs for software vulnerability and functionality assessment [2]. Their research explores the potential of language models to analyse software code and identify possible issues. However, AI-generated assessments require careful verification because a model may produce incomplete or inaccurate vulnerability descriptions. This supports the need for an evidence-based review process in which reported issues are treated as potential findings rather than confirmed vulnerabilities.

Multi-agent collaboration has also been explored for software development and code analysis. The study titled "Multi-Agent LLM Collaboration for Adaptive Code Review, Debugging, and Security Analysis" discusses the use of multiple language-model agents for different software review activities [3]. Such approaches provide a basis for separating reviewing responsibilities among specialized agents. However, coordinating multiple agents also introduces challenges such as overlapping findings, inconsistent conclusions, and additional processing overhead. These challenges highlight the importance of a moderation mechanism that can consolidate and organize the results.

Belozerov et al. examined the use of AI in secure coding, covering activities ranging from vulnerability detection to potential repair [4]. This research is relevant to AI-assisted software security because identifying a possible vulnerability is only one part of the process; the suggested explanation or correction must also be assessed for reliability and safety. The proposed framework therefore focuses on presenting evidence and suggested corrections while keeping developers responsible for applying changes.

Source-code parsing technologies can further support automated code analysis. Tree-sitter provides facilities for generating and incrementally updating concrete syntax trees for supported programming languages [5]. Such structured representations can help identify code elements and provide relevant context to analysis agents. In the proposed framework, syntax-based information can complement source-code context during vulnerability reviewing. However, syntax parsing alone cannot establish whether a program is secure.

Security considerations are also important when integrating LLMs with software repositories. The OWASP Top 10 for Large Language Model Applications identifies risks including prompt injection, sensitive information disclosure, and excessive agency [6]. These risks are particularly relevant when AI agents process repository files, source-code comments, and external information. Appropriate permission restrictions, secure credential handling, input validation, and human oversight are therefore necessary considerations when designing an AI-assisted code-review framework.

Based on these studies and technologies, the proposed Automated Code Vulnerability Reviewing framework combines role-based multi-agent analysis, repository-level context preparation, syntax-aware processing, and moderator-based consolidation. Rather than treating an individual model response as a final security verdict, the framework aims to organize findings from specialized agents into a structured report containing relevant code locations, supporting evidence, and suggested corrections. Its effectiveness will require further implementation and experimental evaluation.

III. PROBLEM STATEMENT AND OBJECTIVES

A. Problem Statement

Modern software development involves frequent code changes, updates, and contributions from multiple developers. Reviewing these changes for security vulnerabilities is an important but time-consuming task. Manual code inspection requires developers to examine source files, understand project structure, and identify potentially unsafe coding practices. Conventional automated tools can assist with this process, but their results may be limited to particular categories of issues and may require developers to interpret findings from multiple sources.

AI-based code analysis provides an additional approach to identifying potential vulnerabilities. However, a single general-purpose AI reviewer may overlook certain issues, produce inaccurate findings, or provide suggestions without sufficient supporting evidence. In addition, developers need a convenient way to initiate reviews, monitor the analysis process, and understand the results in the context of their projects.

Therefore, there is a need for a structured automated code vulnerability reviewing framework that can understand repository context, perform specialized code analysis through multiple AI agents, consolidate their findings, and present prioritized results to developers. The proposed system aims to address these challenges through GitHub integration and a role-based multi-agent architecture while maintaining human control over code modifications and final review decisions.

B. Objectives

The main objectives of the proposed Automated Code Vulnerability Reviewing system are:

1. **GitHub Integration:** To provide automatic and manual review initiation through GitHub-related workflows.
2. **Repository Understanding:** To retrieve repository information and identify supported programming languages, project structure, source files, and dependencies.
3. **Automated Vulnerability Analysis:** To identify potential security vulnerabilities in source code using an AI-based Security Auditor Agent.
4. **Architecture and Code Quality Analysis:** To examine code organization, maintainability, readability, and design-related concerns through a Software Architect Agent.
5. **Multi-Agent Coordination:** To coordinate specialized agents in parallel and consolidate their findings through a Review Moderator Agent.
6. **Finding Prioritization:** To organize reported issues according to their estimated severity, confidence, and supporting evidence.

7. **Developer Interaction:** To provide both a locally hosted web dashboard and a Command-Line Interface (CLI) for initiating and monitoring reviews.
8. **Structured Report Generation:** To generate understandable review reports containing issue descriptions, relevant code locations, supporting evidence, priorities, and suggested corrections.
9. **Human-Controlled Review:** To ensure that the system provides recommendations without automatically modifying project source code, leaving final decisions to developers.

IV. PROPOSED SYSTEM ARCHITECTURE

The proposed Automated Code Vulnerability Reviewing system is designed as a GitHub-integrated, locally executable multi-agent framework. It combines repository analysis, source-code context preparation, specialized AI agents, and report generation into a structured reviewing workflow. The system consists of several interconnected modules that coordinate the reviewing process and provide developers with organized feedback.

A. System Overview

The framework begins when a developer initiates a review through a supported GitHub-integrated action or the Command-Line Interface (CLI). It also supports automatic review initiation through configured GitHub events, such as commits and pull requests. After receiving a review request, the system retrieves the required repository information and prepares the relevant source-code context for analysis.

The repository analysis module examines project files, identifies supported programming languages, and collects information about the project structure and dependencies. Based on the selected review scope, the system prepares the relevant files and code context. Where supported, syntax parsing can provide additional structural information.

The prepared context is then provided to two specialized agents, the Security Auditor Agent and the Software Architect Agent, which perform their analysis independently and in parallel. Their findings are forwarded to the Review Moderator Agent, which consolidates the results, identifies duplicate or conflicting observations, and organizes the final findings according to priority.

The consolidated review is presented through a locally hosted web dashboard and can also be accessed through the CLI. Developers can inspect the findings and download reports in PDF or HTML format.

B. Repository Integration and Analysis Module

The repository integration module connects the framework with GitHub to obtain project information and code changes. It uses authorized GitHub access to retrieve repository files, commit information, pull-request details, and other required metadata.

The repository analysis component examines the available project information to identify supported programming languages, source files, project structure, and dependencies. This information helps the system understand the development context before initiating the specialized review process.

The framework is designed to support different programming languages through automatic language detection. The actual language coverage depends on the parsers and analysis capabilities implemented in the system.

Developers can select the scope of analysis according to their requirements. The proposed system supports repository-level reviewing as well as focused analysis of selected changes, files, or modules.

C. Context Preparation Module

The context preparation module collects and organizes the information required by the reviewing agents. It identifies relevant source-code sections and provides surrounding code context so that the agents can examine the selected code more meaningfully.

Where supported, an Abstract Syntax Tree (AST) or concrete syntax tree can be generated using suitable language parsers. This representation helps identify structural elements such as functions, classes, imports, and relationships within the source code.

The prepared context is then shared with the specialized agents. This process aims to reduce unnecessary information and provide consistent input for parallel analysis.

D. Multi-Agent Reviewing Module

The proposed framework contains three specialized agents, each responsible for a particular reviewing task.

1) Security Auditor Agent:

The Security Auditor Agent focuses on identifying potential security vulnerabilities and unsafe coding practices. It examines source-code context for issues such as improper input validation, injection vulnerabilities, insecure command execution, unsafe handling of sensitive information, and other security-related weaknesses within the supported analysis scope.

2) Software Architect Agent:

The Software Architect Agent examines the structural and design-related aspects of the source code. Its responsibilities include identifying code organization issues, excessive complexity, duplication, maintainability concerns, readability problems, and possible inconsistencies in software design.

3) Review Moderator Agent:

The Review Moderator Agent receives the findings generated by the two specialist agents. It compares their observations, identifies duplicate findings, handles conflicting conclusions, and organizes the results into a consolidated review. It also ensures that reported findings contain relevant explanations and supporting code references.

The Security Auditor and Software Architect agents are designed to operate in parallel, while the Review Moderator processes their results after the specialist analysis is completed.

E. Risk Prioritization and Report Generation

After consolidation, the findings are organized according to their estimated severity, confidence, and supporting evidence. A risk-prioritization mechanism can be used to help developers identify issues that may require earlier attention. The scoring method and its parameters will need to be defined and evaluated during implementation.

Each reported finding is intended to contain the relevant file and line location, issue title, explanation, category, estimated severity, confidence, supporting code evidence, and suggested correction or verification step.

The framework does not automatically modify source-code files or apply patches. Instead, it presents recommendations that developers can inspect and decide whether to implement.

F. Local Execution and Developer Interface

The proposed framework is designed to be installed separately from the developer's project repository. Its installation process is intended to configure the agent application and required software dependencies, allowing the same installation to be reused across multiple projects.

Developers provide their own Large Language Model (LLM) API credentials and authorize the required GitHub access. The API credentials are configured separately and are not intended to be included in the distributed installation package.

The orchestrator and reviewing agents execute locally, while internet connectivity is required for communication with GitHub services and the external LLM provider.

The framework provides two developer interfaces:

1. **Web Dashboard:** A locally hosted web interface that displays review progress, individual agent activities, identified findings, and consolidated results.
2. **Command-Line Interface (CLI):** A terminal-based interface through which developers can initiate reviews and interact with the reviewing system.

The final reports can be accessed through the dashboard and downloaded in PDF or HTML format. Optional review-history functionality can also be used to retain previous reports and compare findings across subsequent reviews.

Overall, the proposed architecture aims to provide a structured and reusable environment for automated code vulnerability reviewing while maintaining developer control over the final review and modification process.

V. METHODOLOGY

The proposed Automated Code Vulnerability Reviewing system follows a structured workflow consisting of multiple stages. These stages cover repository access, project analysis, context preparation, multi-agent reviewing, result consolidation, and report generation.

The proposed methodology consists of the following stages:

1) System Installation and Configuration:

The developer installs the Multi-Agent Reviewing System separately from the project repository. The installation process is designed to configure the required application components and software dependencies. The developer then provides an LLM API key and authorizes the required GitHub access permissions.

2) Review Initiation:

A review can be initiated automatically through configured GitHub events, such as commits and pull requests, or manually through the developer interface. The developer can select the required repository and define the review scope according to the requirements.

3) Repository Retrieval and Analysis:

The system connects to GitHub using authorized access and retrieves the required repository information. It examines the project structure, identifies supported programming languages, locates relevant source files, and collects available dependency information.

4) Source-Code Context Preparation:

The system identifies the files and code sections relevant to the selected review scope. It collects surrounding code context and, where supported, uses an Abstract Syntax Tree (AST) or concrete syntax tree to obtain additional structural information.

5) Parallel Specialist Analysis:

The prepared source-code context is provided to the Security Auditor Agent and Software Architect Agent. Both agents perform their respective analyses independently and in parallel. The Security Auditor examines potential security vulnerabilities, while the Software Architect evaluates code organization, maintainability, readability, and design-related concerns.

6) Review Moderation and Consolidation:

After both specialist agents complete their analysis, their findings are forwarded to the Review Moderator Agent. It compares the reported issues, identifies duplicate observations, handles conflicting conclusions, and consolidates the results into a structured format.

7) Risk Prioritization:

The consolidated findings are organized according to estimated severity, confidence, and supporting evidence. A risk-

prioritization mechanism is intended to help developers identify issues that may require immediate attention. Its scoring factors and weights will be established and evaluated during implementation.

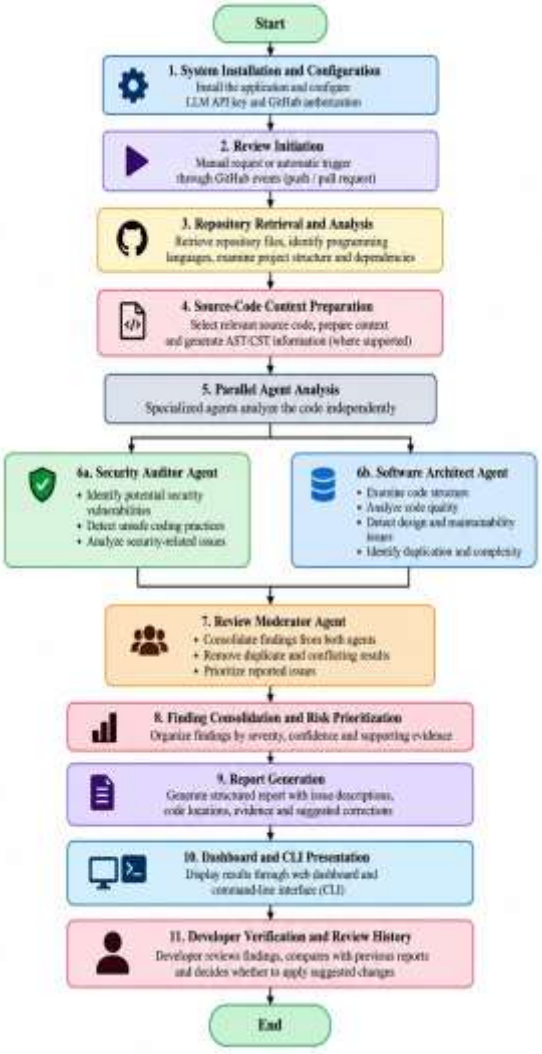
8) Report Generation and Presentation:

The system generates a structured review report containing issue titles, descriptions, relevant file and line locations, supporting evidence, priority levels, and suggested corrections. The results are presented through the local web dashboard and CLI. Reports are also intended to be downloadable in PDF and HTML formats.

9) Review History and Developer Verification:

If review-history functionality is enabled, previous reports can be retained for comparison with subsequent reviews. Findings can be examined to identify issues that remain present, have disappeared from later reviews, or have newly appeared. Developers remain responsible for verifying the findings and applying any required code changes.

Through these stages, the proposed methodology aims to provide an organized and repeatable approach to automated code vulnerability reviewing while maintaining human oversight throughout the process.



VI. TOOLS AND TECHNOLOGIES

The proposed Automated Code Vulnerability Reviewing system requires a combination of repository integration tools, programming technologies, AI services, code-parsing tools, and developer-interface components. The following technologies are considered for developing the proposed framework.

A. GitHub Integration:

GitHub will serve as the primary repository integration platform. Its APIs and supported event mechanisms will be used to retrieve repository information, access source-code changes, and initiate reviews through configured commit and pull-request events. Developers will authorize the required repository permissions before using the system.

B. Programming Language and Backend:

A backend programming environment such as Python or JavaScript can be used to implement the orchestrator, repository analysis module, and agent communication workflow. The final programming language will be selected based on library availability, integration requirements, and implementation suitability.

C. Large Language Model API:

An external Large Language Model (LLM) API will provide the language-model capabilities required by the Security Auditor Agent, Software Architect Agent, and Review Moderator Agent. Developers will configure their own API credentials. The initial implementation is intended to use one model provider, while the architecture may support additional providers in future versions.

D. Multi-Agent Orchestration:

A multi-agent orchestration framework or a custom coordination mechanism can be used to manage agent execution, parallel analysis, communication, and result consolidation. The final implementation will use clearly defined agent responsibilities and structured outputs to maintain consistency between the reviewing components.

E. Source-Code Parsing and Analysis:

Language-specific parsers or Tree-sitter can be considered for identifying source-code structures and generating syntax representations. These representations can help the system prepare relevant context for supported programming languages before sending information to the reviewing agents.

F. Web Dashboard and Command-Line Interface:

A locally hosted web dashboard will provide a visual interface for initiating reviews, monitoring agent activities, viewing findings, and accessing reports. A Command-Line Interface (CLI) will provide an additional terminal-based method for interacting with the system. Both interfaces will communicate with the same local orchestrator.

G. Report Generation and Storage:

The report-generation component will organize consolidated findings into structured output. PDF and HTML formats are intended for downloadable reports. An optional local review-history mechanism may retain previous reports and findings to support comparisons across subsequent reviews.

H. Security and Credential Management:

The framework will require secure handling of LLM API credentials and GitHub authorization. Credentials should be stored separately from project source files and excluded from version control. Repository permissions should follow the principle of least privilege, while sensitive source-code information and model-provider data-handling policies must be considered during processing.

The final technology stack will be determined during implementation. The selected tools, libraries, model versions, and configurations will be documented to support reproducibility and future evaluation.

VII. EVALUATION PLAN AND DISCUSSION

The proposed Automated Code Vulnerability Reviewing framework will be evaluated after implementation to assess its ability to identify potential vulnerabilities, analyse software architecture, and generate structured code-review reports. The evaluation will focus on the accuracy, reliability, efficiency, and usefulness of the multi-agent reviewing approach.

A. Evaluation Dataset

A collection of software repositories and code changes written in the programming languages supported by the system will be selected for evaluation. The dataset should include examples containing known security vulnerabilities, coding issues, and code changes without known issues. Repository-level information, source files, and dependency details will be considered wherever relevant to the review.

B. Comparative Evaluation

The proposed framework will be compared with selected code-review approaches, including conventional static analysis tools and a single-agent LLM-based review approach. This comparison will help examine whether assigning separate responsibilities to specialized agents and using a Review Moderator can improve the organization and reliability of review findings.

The Software Architect Agent will be evaluated based on its ability to identify software design and code quality issues, including poor code organization, duplicated code, high complexity, readability problems, and maintainability concerns. The findings generated by the agent will be reviewed by human evaluators and compared with manually identified issues. Each finding will be assessed for relevance, correctness, clarity, and usefulness of the recommendation. This evaluation will help determine whether the agent provides meaningful suggestions for improving software structure and maintainability.

C. Evaluation Metrics

The following metrics are proposed for evaluating the system:

1. **Precision:** Measures the proportion of reported issues that are valid findings.
2. **Recall:** Measures the proportion of known issues successfully identified by the system.
3. **F1-Score:** Provides a combined measure of precision and recall.
4. **False Positive Rate:** Measures how frequently the system reports issues that are not genuine vulnerabilities or defects.
5. **Duplicate Finding Rate:** Evaluates the moderator's ability to identify and remove repeated findings.
6. **Localization Accuracy:** Measures whether reported issues correctly identify the relevant files and code locations.
7. **Review Time:** Measures the time required to complete a review.
8. **API Usage and Cost:** Estimates the external LLM API usage and associated processing cost.

D. Multi-Agent Evaluation

The contribution of each agent will be examined separately. The Security Auditor Agent will be evaluated for security-related findings, while the Software Architect Agent will be assessed for architecture and code-quality observations. The Review Moderator Agent will be evaluated based on its ability to consolidate findings, remove duplicates, handle conflicting observations, and assign appropriate priorities.

A comparison between the complete multi-agent workflow and configurations without the moderator or other selected components may also be performed to understand the contribution of individual modules.

E. Expected Discussion

The proposed framework is intended to provide a more organized reviewing process by combining security-focused analysis with architecture and code-quality assessment. Parallel execution of the specialist agents may help separate different review responsibilities, while the moderator is intended to produce a consolidated report.

However, the framework may also introduce additional processing time and API costs because multiple agents are involved. Its effectiveness will depend on the quality of repository context, supported programming languages, LLM responses, and the accuracy of the moderation process.

The actual performance of the proposed system can only be established through implementation and experimental evaluation. Therefore, no quantitative performance results or claims of improved accuracy are presented in this paper at this stage.

VIII. LIMITATIONS AND SECURITY CONSIDERATIONS

Although the proposed Automated Code Vulnerability Reviewing framework aims to support automated code review through a multi-agent approach, certain limitations and security considerations must be addressed during its implementation and evaluation.

A. Limitations

1. **LLM Reliability:** Large Language Models may generate incorrect, incomplete, or unsupported findings. Therefore, the reported issues require developer verification.
2. **False Positives and False Negatives:** The framework may report issues that are not actual vulnerabilities or fail to identify existing security weaknesses.
3. **Programming Language Support:** The analysis capability depends on the availability and quality of language-specific parsing and analysis support. Unsupported languages or constructs may reduce the completeness of the review.
4. **Processing Time and API Cost:** Coordinating multiple agents may increase processing time and external API usage compared with a single-agent approach.
5. **Repository Context Limitations:** Large repositories may contain more source code and dependency information than can be processed in a single analysis request. Context selection and processing limits may therefore affect the findings.
6. **External Service Dependency:** The proposed system requires an internet connection for GitHub communication and external LLM API access. Service interruptions or API limitations may affect review execution.

B. Security Considerations

1. **Source-Code Privacy:** Repository code and relevant context may be transmitted to an external LLM provider for analysis. Developers must consider the provider's data-handling policies and avoid submitting confidential information without authorization.
2. **API Key Protection:** LLM API keys and GitHub credentials must be stored securely and must not be exposed in source code, logs, or generated reports.
3. **GitHub Permissions:** The framework should follow the principle of least privilege. Read-only repository access should be used by default, with additional permission required only when a feature such as publishing review comments is enabled.
4. **Prompt Injection Risks:** Repository files, comments, and documentation may contain instructions intended to manipulate an LLM. Such content must be treated as untrusted input rather than instructions that override the system's review rules.
5. **Human Verification:** The framework is designed to provide recommendations and suggested corrections. Developers should verify findings before making changes. The system should not automatically modify source files or apply code patches.
6. **Report and History Protection:** If review history is enabled, stored reports may contain sensitive code information. Local storage and access to these reports should therefore be appropriately protected.

Addressing these limitations and security considerations will be necessary to make the proposed framework reliable and suitable for practical software development environments.

IX. FUTURE SCOPE

The proposed Automated Code Vulnerability Reviewing framework can be extended in several ways to improve its analysis capabilities, usability, and integration with software development workflows.

A. Additional Specialized Agents

The framework can be extended by introducing additional agents with specific responsibilities, such as a Performance Analysis Agent, Testing Agent, and Dependency Analysis Agent. These agents could examine performance-related issues, test coverage, and vulnerabilities in third-party dependencies.

B. Expanded Programming Language Support

Future versions can provide broader programming language support by integrating additional language-specific parsers and analysis techniques. This would allow the framework to review repositories developed using a wider range of programming languages and technologies.

C. Advanced Vulnerability Detection

The Security Auditor Agent can be improved through the integration of additional security analysis techniques, vulnerability databases, and updated security guidelines. This may help identify a wider range of security weaknesses and provide more relevant recommendations.

D. Improved Review History and Comparison

The review history feature can be enhanced to compare findings across multiple reviews. This would help developers identify issues that are no longer detected in subsequent reviews, remain present, appear for the first time, or occur repeatedly in subsequent reviews.

E. Enhanced GitHub Integration

The framework can be extended with additional GitHub workflow integrations, including continuous integration and continuous deployment (CI/CD) pipelines. This could allow code reviews to become part of the development workflow whenever supported repository events occur.

F. Multiple LLM Provider Support

Future versions may support multiple LLM providers, allowing developers to select an appropriate model based on cost, performance, availability, and analysis requirements.

G. Improved Remediation Assistance

The framework may provide more detailed remediation guidance and optional patch suggestions. Any future capability to apply changes should include developer approval, validation, and appropriate safeguards to prevent unintended modifications.

These extensions could make the framework more flexible and useful for different software development environments while maintaining its primary objective of assisting developers through automated and structured code reviewing.

X. CONCLUSION

This paper presented the proposed Automated Code Vulnerability Reviewing framework, which aims to assist developers in reviewing software repositories through a GitHub-integrated, role-based multi-agent architecture. The framework combines repository analysis, source-code context preparation, and specialized review agents to examine security vulnerabilities and software architecture-related concerns.

The proposed system consists of three agents: the Security Auditor Agent, the Software Architect Agent, and the Review Moderator Agent. The first two agents perform their respective analyses in parallel, while the moderator consolidates their findings, removes duplicates, handles conflicting observations, and prioritizes the reported issues. The framework also proposes a local execution environment with a web-based dashboard and command-line interface, providing developers with access to structured review reports.

Unlike systems that directly modify source code, the proposed framework focuses on identifying potential issues and providing recommendations for developer verification. Its planned integration with GitHub is intended to support both manual and event-triggered code reviews.

The paper also discussed the proposed methodology, evaluation approach, limitations, and security considerations. Since the framework is currently at the proposal stage, its practical effectiveness, accuracy, and performance have not yet been experimentally established. Future implementation and testing will be required to validate its capabilities and determine its usefulness in real-world software development environments.

ACKNOWLEDGMENT

The authors acknowledge the guidance of the Department of Artificial Intelligence and Machine Learning, Zeal Polytechnic, Pune, and the faculty members supporting the project. The authors also acknowledge the research publications and technical documentation consulted during the preparation of this paper.

REFERENCES

[1] J. Yu, P. Liang, Y. Fu, A. Tahir, M. Shahin, C. Wang, and Y. Cai, “An insight into security code review with LLMs: Capabilities, obstacles and influential factors,” *arXiv preprint arXiv:2401.16310*, 2024. [Online]. Available: https://arxiv.org/abs/2401.16310

[2] R. I. T. Jensen, V. Tawosi, and S. Alamir, “Software vulnerability and functionality assessment using LLMs,” *arXiv preprint arXiv:2403.08429*, 2024. [Online]. Available: https://arxiv.org/abs/2403.08429

[3] T. Sharanarthi and S. Polineni, “Multi-Agent LLM Collaboration for Adaptive Code Review, Debugging, and Security Analysis,” in *Proc. 2025 International Conference on Mechatronics, Robotics, and Artificial Intelligence (MRAI)*, 2025, doi: 10.1109/MRAI65197.2025.11135756.

[4] V. Belozerov, P. J. Barclay, and A. Sami, “Secure coding with AI—from detection to repair,” *Empirical Software Engineering*, vol. 31, art. no. 93, 2026, doi: 10.1007/s10664-026-10812-8.

[5] Tree-sitter, “Introduction,” *Tree-sitter Documentation*. [Online]. Available: https://tree-sitter.github.io/tree-sitter/. [Accessed: Oct. 29, 2026].

[6] OWASP Foundation, “OWASP Top 10 for Large Language Model Applications,” OWASP GenAI Security Project, 2026. [Online]. Available: https://genai.owasp.org/llm-top-10/