JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Ai-Code-Reviewer

Masud Sayyed
M.Sc.CS (Cybersecurity)
Department of Advanced Computing
Nagindas Khandwala College, Malad, India

Abstract : Modern software development requires high code quality, strong security practices, and efficient debugging processes. However, manual code reviews and traditional debugging methods are often time-consuming and may fail to detect hidden vulnerabilities or inefficient coding patterns. With the increasing complexity of software systems and the growing use of third-party libraries, developers require intelligent tools that can assist in identifying issues early in the development lifecycle. To address these challenges, this project proposes an AI Code Reviewer & Debugging Assistant, a MERN stack-based application designed to automate code analysis and improve software security.

IndexTerms **- Cybersecurity, Automation, Code Analysis**

I. INTRODUCTION

The AI Code Reviewer & Debugging Assistant is designed to streamline the software development lifecycle by improving both code quality and security during the development process. The primary objective of this project is to automatically analyze source code, detect bugs, identify security vulnerabilities, and provide intelligent suggestions for improvement. By integrating artificial intelligence with modern development tools, the system helps developers write cleaner, safer, and more efficient code while reducing the time required for manual code reviews and debugging.

The system is built using a modern full-stack architecture. The frontend is developed using React.js, Tailwind CSS, Redux, and JavaScript, providing a responsive and user-friendly interface for developers to submit and review code analysis results. The backend is implemented with Node.js and Express.js, while MongoDB is used as the database to store analysis results, project data, and user information.

To enhance the analysis capabilities, the platform integrates several advanced technologies and security tools. These include Gemini API for intelligent code review and debugging suggestions, GitHub API for repository integration, npm audit, Snyk API, GitHub CodeQL, and OWASP Dependency-Check for detecting vulnerabilities in dependencies and source code. The application is deployed using Vercel for the frontend and Render or Heroku for the backend, ensuring scalability and easy accessibility. For testing and API validation, Postman is used, while GitHub with Webhooks enables version control and automated integration with repositories.

Overall, this project provides an AI-powered automated code review and debugging environment that assists developers in identifying errors, improving code efficiency, and strengthening application security, ultimately contributing to faster and more reliable software development.

II. Background

In modern software development, writing efficient, secure, and error-free code is a critical requirement. As software systems become more complex, developers face increasing challenges in identifying bugs, maintaining code quality, and ensuring application security. Traditional code review processes are often manual, time-consuming, and dependent on the experience of developers or reviewers. This can lead to overlooked vulnerabilities, inefficient debugging, and delays in the development lifecycle.

With the rapid growth of open-source libraries and third-party dependencies, security risks have also increased significantly. Many applications rely on external packages that may contain hidden vulnerabilities,

outdated components, or insecure configurations. If these issues are not detected early, they can expose applications to cyber threats such as data breaches, injection attacks, and unauthorized access. Therefore, integrating automated security analysis and vulnerability detection tools has become essential in modern development practices.

Recent advancements in Artificial Intelligence (AI) and automated software analysis have introduced new opportunities to improve the code review process. AI-based tools can analyze code patterns, detect logical errors, suggest improvements, and help developers fix bugs more efficiently. Additionally, integrating security scanning tools such as dependency vulnerability checkers and static code analysis frameworks can further enhance the security posture of applications.

The AI Code Reviewer & Debugging Assistant project is developed to address these challenges by combining AI-driven code analysis with automated security scanning. By integrating technologies such as AI-based code review, dependency vulnerability detection, and repository integration, the system provides developers with a centralized platform for analyzing code quality, detecting potential bugs, and identifying security risks. This approach aims to reduce manual effort, improve development efficiency, and promote secure coding practices throughout the software development lifecycle.

III. Purpose of the Study

The proposed system is a MERN Stack-based AI application designed to automate code review, vulnerability detection, and debugging assistance for developers. The platform integrates Artificial Intelligence with modern security analysis tools to evaluate source code for logical errors, performance issues, and security vulnerabilities. Using the GPT API, the system performs AI-powered code review by analyzing code structure, efficiency, readability, and potential bugs, while also suggesting improvements to enhance overall code quality.

In addition to AI analysis, the system incorporates integrated security scanning mechanisms such as Static Application Security Testing (SAST), dependency vulnerability checks, and secret detection to identify insecure coding practices and exposed credentials within the codebase. Detected vulnerabilities are further analyzed through a vulnerability scoring and mapping framework, where issues are assigned severity scores based on the Common Vulnerability Scoring System (CVSS) and mapped to well-known security standards such as OWASP Top 10, MITRE ATT&CK, and SANS Top 25. This helps developers understand the impact and classification of each detected issue.

IV. Scope of the Project

he scope of the AI Code Reviewer & Debugging Assistant project is to develop an intelligent platform that automates code review, vulnerability detection, and debugging support for software developers. The system focuses on improving code quality, development efficiency, and application security by integrating artificial intelligence with automated security analysis tools within a MERN stack-based environment.

The project covers the development of a platform capable of analyzing source code using AI-based techniques to detect logical errors, inefficient coding patterns, and readability issues. By integrating the GPT API, the system will provide intelligent feedback and suggestions to help developers improve their code. The system will also incorporate static application security testing (SAST), dependency vulnerability scanning, and secret detection to identify security risks within the application code and third-party libraries.

Another important scope of the project includes vulnerability classification and risk assessment, where identified issues will be assigned severity levels using the Common Vulnerability Scoring System (CVSS) and mapped to recognized security standards such as OWASP Top 10, MITRE ATT&CK, and SANS Top 25. This enables developers to better understand the potential impact of vulnerabilities and prioritize fixes effectively.

The platform will also support automation and integration with version control systems, particularly GitHub, where scans can be automatically triggered through GitHub Webhooks whenever a new pull request or code update occurs. Additionally, the system will generate detailed security and analysis reports in PDF or HTML format that can be used for documentation, compliance, and auditing purposes.

Abbreviation sand Acronyms

- **AI** – Artificial Intelligence
- **API –** Application Programming Interface
- **MERN** – MongoDB, Express.js, React.js, Node.js
- **SAST** – Static Application Security Testing
- **CVSS** – Common Vulnerability Scoring System
- **OWASP** – Open Worldwide Application Security Project

- **MITRE** – MITRE Corporation (Cybersecurity Knowledge Base Organization)
- **SANS** – SysAdmin, Audit, Network, and Security Institute
- **GUI** – Graphical User Interface
- **CI/CD** – Continuous Integration / Continuous Deployment
- **JSON** – JavaScript Object Notation
- **IDE** – Integrated Development Environment

3.1Population and Sample

The population for this study includes **software developers, cybersecurity professionals, and students involved in software development and security testing** who require tools to improve code quality and detect vulnerabilities in applications. It also includes **software projects and repositories** that are analyzed using the AI Code Reviewer & Debugging Assistant system. These users represent individuals who regularly perform code reviews, debugging, and security analysis during the software development lifecycle.

The sample for this study consists of **selected source code repositories and code snippets** used to evaluate the performance and effectiveness of the proposed system. A limited number of repositories or code samples containing different programming patterns, dependencies, and potential vulnerabilities are analyzed using the developed platform. Additionally, a group of **developers or students** may be selected to test the system and provide feedback on its functionality, accuracy, and usability. This sample helps in assessing how effectively the system detects vulnerabilities, suggests improvements, and supports developers in improving code quality and security.

3.2 Data and Sources of Data

The data used in this study primarily consists of **source code files, software repositories, and vulnerability information** required for analyzing code quality and security risks. The collected data includes different programming code samples, dependency information, and security vulnerability data that are used to test and evaluate the performance of the **AI Code Reviewer & Debugging Assistant** system.

The primary source of data is **public and private GitHub repositories**, where source code is obtained for analysis through the GitHub API. These repositories provide real-world programming examples that help in identifying coding issues, vulnerabilities, and dependency-related risks. Additional data related to known vulnerabilities is obtained from **security databases and scanning tools** such as npm audit, Snyk API, and GitHub CodeQL, which provide information about insecure libraries, outdated packages, and potential security threats.

Furthermore, the system also generates **analysis results and vulnerability reports** as part of the data during the scanning process. These results include detected bugs, security issues, vulnerability severity scores (CVSS), and recommended remediation steps. The generated reports are stored in the **MongoDB database**, which serves as a secondary source of data for evaluating the effectiveness of the system and analyzing the overall performance of the proposed solution.

3.3 Theoretical framework

The theoretical framework of the **AI Code Reviewer & Debugging Assistant** is based on the integration of **Artificial Intelligence, software engineering principles, and cybersecurity analysis techniques** to automate code review and vulnerability detection. Traditional software development relies heavily on manual code reviews and debugging processes, which can be time-consuming and prone to human error. By applying AI-based analysis and automated security scanning, the system aims to enhance the efficiency and reliability of code evaluation during the software development lifecycle.

One of the key theoretical foundations of this system is **Static Application Security Testing (SAST)**. SAST techniques analyze source code without executing the program to detect potential vulnerabilities such as insecure coding patterns, misconfigurations, and dependency-related issues. Tools like dependency scanners and static analysis engines are integrated into the framework to identify vulnerabilities early in the development process, reducing the risk of security breaches in deployed applications.

Another important concept used in the framework is **Artificial Intelligence–based code analysis**, where machine learning and natural language processing techniques are applied to understand code structure, logic, and context. Through the use of AI models such as the GPT API, the system can interpret programming code, detect inefficient logic, and provide intelligent suggestions to improve code quality and readability.

The framework also incorporates **vulnerability assessment and classification models** such as the **Common Vulnerability Scoring System (CVSS)** for assigning severity scores to identified issues. Additionally, detected vulnerabilities are mapped to widely recognized cybersecurity standards including **OWASP Top 10, MITRE ATT&CK, and SANS Top 25**, which provide structured guidelines for understanding common security threats and weaknesses.

Overall, the theoretical framework combines concepts from **AI-driven analysis, secure software development practices, and automated vulnerability assessment** to create an integrated system capable of improving code quality, identifying security risks, and assisting developers in producing secure and efficient software.

Equations

In the **AI Code Reviewer & Debugging Assistant**, certain analytical and scoring methods can be represented using mathematical expressions to evaluate vulnerability severity, code quality, and system efficiency.

1. Vulnerability Severity Score (CVSS-Based Representation): The severity of a vulnerability can be represented using a simplified scoring model derived from the Common Vulnerability Scoring System (CVSS).

$$Vs=(C+I+A)\times E$$

Where:

- **Vs** = Vulnerability Severity Score
- **C** = Confidentiality impact
- **I** = Integrity impact
- **A** = Availability impact
- **E** = Exploitability factor

This equation helps estimate the overall risk level of a detected vulnerability.

2. Code Quality Score

The quality of the analyzed code can be evaluated by considering multiple parameters such as readability, efficiency, and error count.

$$CQ = R+Ef+S / B+1$$

Where:

- **CQ** = Code Quality Score
- **R** = Readability score
- **Ef** = Efficiency factor
- **S** = Security score
- **B** = Number of detected bugs

This measures reliability of the scanning framework.

4. Vulnerability Density

This metric helps measure the number of vulnerabilities relative to the size of the codebase.

$$VD = V / LOC$$

Where:

- VD = Vulnerability Density
- V = Total number of vulnerabilities detected
- LOC = Lines of Code

Lower vulnerability density indicates more secure code.

RESEARCH METHODOLOGY

The research methodology for the **AI Code Reviewer & Debugging Assistant** focuses on designing, developing, and evaluating an automated system that integrates artificial intelligence with security scanning tools to improve code quality and detect vulnerabilities. The methodology follows a **system development and experimental evaluation approach**, where the system is implemented using modern web technologies and tested using real code repositories and sample datasets.

3.1 Research Design

The research design for the **AI Code Reviewer & Debugging Assistant** follows a **development-based research approach** that focuses on designing, implementing, and evaluating an intelligent system for automated code review and vulnerability detection. The study involves analyzing existing code review and security scanning tools, identifying their limitations, and proposing a system that integrates artificial intelligence with automated security analysis. The design emphasizes improving code quality, detecting vulnerabilities, and providing remediation suggestions through an AI-powered framework.

3.2 System Development Approach

The system is developed using a **modular and iterative development approach**. The architecture is based on the **MERN stack (MongoDB, Express.js, React.js, and Node.js)** to ensure scalability and maintainability. The development process includes requirement analysis, system design, implementation, integration of AI and security tools, testing, and deployment. Each module of the system—such as code submission, AI analysis, vulnerability scanning, and report generation—is developed and integrated step by step to ensure proper functionality.

3.3 Data Collection Method

The data used for analysis is collected from **source code repositories, sample code snippets, and dependency data**. Public and private repositories from GitHub are used as the primary source of code samples. The system collects information related to coding patterns, dependencies, and potential vulnerabilities. Additionally, vulnerability databases and security scanning tools provide information about known vulnerabilities in third-party libraries and software components.

3.4 Tools and Technologies Used

The system utilizes several technologies to implement the proposed solution. The frontend is developed using **React.js, Tailwind CSS, and Redux**, while the backend is implemented using **Node.js and Express.js** with **MongoDB** as the database. AI-based analysis is performed using the **GPT API**, and repository integration is handled using the **GitHub API**. Security scanning tools such as **npm audit, Snyk API, and GitHub CodeQL** are integrated to detect vulnerabilities in dependencies and source code. Additional tools such as **Postman and Jest** are used for testing and validating the system.

3.4.1 System Testing and Evaluation

The developed system is tested using various code samples and repositories to verify its functionality and accuracy. Testing includes **functional testing, API testing, and performance evaluation** to ensure the system correctly analyzes code and detects vulnerabilities. The evaluation focuses on parameters such as vulnerability detection capability, accuracy of AI suggestions, scanning speed, and usability of generated reports. The results are analyzed to determine the effectiveness of the system in improving code quality and identifying security risks.

3.4.2 Comparison of the Models

The proposed system is compared with existing code review and vulnerability scanning tools based on several criteria such as **automation, AI integration, vulnerability detection capability, remediation suggestions, and reporting features**. Traditional tools typically perform static analysis or dependency scanning separately, while the proposed system integrates **AI-driven code analysis with multiple security scanning tools in a single platform**. This integrated approach improves the efficiency of code review, provides intelligent debugging suggestions, and generates comprehensive compliance reports, making it more effective for modern secure software development practices.

IV. RESULTS AND DISCUSSION

4.1 Results of Descriptive Statics of Study Variables

This section presents the findings obtained from the implementation and testing of the **AI Code Reviewer & Debugging Assistant**. The results demonstrate how effectively the system analyzes source code, detects vulnerabilities, and provides intelligent remediation suggestions. The developed platform integrates AI-based code analysis with automated security scanning tools, allowing developers to identify coding issues and security risks during the development process. The discussion focuses on the system's ability to improve code quality, detect vulnerabilities, and provide structured reports that help developers understand and resolve issues efficiently.

The descriptive statistics summarize the key variables observed during the evaluation of the system. These variables include the number of code samples analyzed, vulnerabilities detected, types of vulnerabilities identified, and the effectiveness of AI-generated recommendations. The results show that the system successfully identifies coding issues and security vulnerabilities across different code samples and repositories.

During testing, multiple code repositories and sample files were analyzed using the integrated scanning tools. The system detected various categories of vulnerabilities such as insecure dependencies, exposed secrets, and potential coding errors. The vulnerability severity levels were classified using CVSS scoring, and the issues were mapped to recognized security frameworks such as OWASP Top 10 and SANS Top 25.

The analysis also indicates that the AI-powered code review module effectively provided suggestions for improving code readability, efficiency, and security. The generated reports included detailed information about detected vulnerabilities, severity levels, and recommended remediation steps. These descriptive results demonstrate that the proposed system can assist developers in identifying issues early in the development lifecycle and improving the overall security and quality of software applications.

REFERENCES

[1]Pressman, R. S., & Maxim, B. R. (2019). Software Engineering: A Practitioner's Approach (9th ed.). McGraw-Hill Education.
[2]Sommerville, I. (2016). Software Engineering (10th ed.). Pearson Education.
[3]Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.
[4]Martin, R. C. (2011). The Clean Coder. Prentice Hall.
[5]McGraw, G. (2006). Software Security: Building Security In. Addison-Wesley.
[6]Torrens, P. (2020). Practical Vulnerability Management. No Starch Press.
[7]Hunt, A., & Thomas, D. (1999). The Pragmatic Programmer. Addison-Wesley.
[8]Flanagan, D. (2020). JavaScript: The Definitive Guide (7th ed.). O'Reilly Media.
[9]OWASP Foundation, "OWASP Top Ten – Security Risks." Accessed: Jan. 2025. [Online]. Available: https://owasp.org/www-project-top-ten/
[10]Mozilla Developer Network, "JavaScript Documentation." Accessed: Jan. 2025. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript
[11]W3Schools, "Web Development Tutorials." Accessed: Jan. 2025. [Online]. Available: https://www.w3schools.com
[12]GitHub Documentation, "Secure Coding & Repository Management." Accessed: Jan. 2025. [Online]. Available: https://docs.github.com
[13]Stack Overflow, "Developer Community Knowledge Base." Accessed: Jan. 2025. [Online]. Available: https://stackoverflow.com
[14]National Institute of Standards and Technology (NIST), "National Vulnerability Database (NVD)." Accessed: Jan. 2025. [Online]. Available: https://nvd.nist.gov
[15]MITRE Corporation, "Common Vulnerabilities and Exposures (CVE)." Accessed: Jan. 2025. [Online]. Available: https://cve.mitre.org