IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510146

# AI-Powered Penetration Testing and Vulnerability Remediation for GitHub Repositories

**Mrs. Bindu K.P[1], Gagana R[2], Kushal K R[3], M G Sahana[4], and M Harshit Pramod[5]**

Assistant Prof., Dept of CSE, K.S.School of Engineering & Management, Bengaluru, India[1]

Student, Dept of CSE, K.S.School of Engineering & Management, Bengaluru, India[2–5]

**Abstract**: AI-Powered Penetration Testing and Vulnerability Remediation for GitHub Repositories presents PentestAI, a web platform that scans a developer's GitHub repositories for security vulnerabilities, verifies the serious findings in an isolated sandbox and proposes fixes as pull requests. The system combines a Large Language Model (Google Gemini) that analyses the most security-relevant source files with a deterministic pattern-matching pass that detects committed secrets such as API keys and private keys.

High and critical findings are re-checked with CodeQL and Semgrep inside network isolated Docker containers to reduce false positives. A fixer service generates a patch with a code model, re-scans the patched code twice and opens a pull request only if the vulnerability is gone and no new issue is introduced. The application is built with TanStack Start, React and Supabase, and supports weekly automated scans, vulnerability lifecycle tracking, reports and an AI security assistant. The results show that the platform turns a manual, expensive security audit into a fast, repeatable and verifiable process for everyday developers.

**Keywords:** Penetration Testing, Large Language Models, Static Application Security Testing, Vulnerability detection, Secret Detection, Automated Program Repair, Sandboxing

## I. INTRODUCTION

Software is written and published faster than ever, but security review has not kept pace. Students, startups and open-source maintainers routinely push code to GitHub without any security testing, so common flaws such as SQL injection, cross-site scripting, broken access control and leaked credentials reach production unnoticed. Manual penetration testing is slow and expensive, while many automated scanners either need deep configuration or produce so many false positives that developers learn to ignore them.

Large Language Models (LLMs) make code analysis far easier to use, but their output is not deterministic and recent studies show that they overestimate their own accuracy on realistic vulnerability benchmarks [3]. The proposed system, PentestAI, therefore uses an LLM to find likely vulnerabilities across a repository and then verifies each serious finding with established static analysis tools inside an isolated sandbox before presenting it to the user. Users sign in with GitHub, import repositories, run scans, review verified findings and request AI-generated fixes that are delivered as pull requests for human review.

Primary features of the system include:

- **AI-Based Repository Scanning:** The scanner ranks files by security relevance, selects up to 18 files (250 KB) and asks Google Gemini to report vulnerabilities with severity, CWE, CVSS score, location and remediation.
- **Deterministic Secret Detection:** Regular-expression rules detect AWS keys, GitHub tokens, Google, OpenAI, Stripe and Slack keys, private key blocks and .env values before the AI pass, so a leaked credential is never missed.
- **Sandbox Verification:** High and critical findings are re-checked with CodeQL and Semgrep in a Docker container with no network access, a read-only file system and strict resource limits.
- **Verified AI Fixes:** A fixer generates a patch, re-scans the patched code twice and opens a GitHub pull request only when the vulnerability is gone and no new issue appears.
- **Continuous Monitoring:** Repositories can be scanned weekly with an email digest, and each vulnerability is tracked across scans with a full timeline, including regressions.
- **Security Assistant and Reports:** A chat assistant answers questions using the user's own scan data, and reports can be exported as JSON or PDF

 This work is licensed under a Creative Commons Attribution 4.0 International License 



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026
DOI: 10.17148/IJARCCE.2026.1510146

## II. RELEVANT LITERATURE

### A. LLMs for Vulnerability Repair

Pearce et al. showed that off-the-shelf code LLMs can repair security bugs without task specific training: together, the models repaired all of their synthetic and hand-crafted scenarios, but producing functionally correct fixes for real-world bugs remained difficult [1]. Their earlier study found that code suggested by GitHub Copilot was frequently vulnerable [5]. This motivates verifying every AI-generated patch before it is offered to a developer.

### B. Combining LLMs with Static Analysis

IRIS combines LLMs with CodeQL by letting the model infer taint specifications. On 120 validated Java vulnerabilities, CodeQL alone detected 27 while IRIS with GPT-4 detected 55, and it also reduced false discoveries [2]. The result shows that LLMs and static analysers are complementary, which is the basis of the scan-then-verify design used in this work.

### C. Limits of Learned Vulnerability Detectors

Ding et al. introduced the PrimeVul dataset and found that a 7B model scoring 68.26% F1 on BigVul scored only 3.09% F1 on PrimeVul; larger models, including GPT-4, did not give meaningful gains under the strictest settings [3]. Zhou et al. similarly report emerging but limited results for LLM-based detection [6]. Model output must therefore be treated as a candidate, not a verdict.

### D. Secret Leakage in Public Repositories

Meli et al. found secret leakage in more than 100,000 public GitHub repositories, with thousands of new, unique secrets leaked every day [4]. Because leaked keys are high impact and follow recognisable formats, deterministic pattern matching is a more reliable first line of defence than an AI model alone.

## III. SYSTEM DESIGN AND METHODOLOGY

The system is split into two independent halves that communicate only through signed HTTP messages. The web application, built with TanStack Start, React and TypeScript, handles sign-in, repository management, results and reports, and stores data in Supabase PostgreSQL with Row Level Security. The sandbox, deployed with Docker Compose, contains a Caddy HTTPS proxy, a Fastify job API, a Redis queue, a persistent worker and one-shot runner and fixer containers. The application never runs untrusted code, and the sandbox never holds a database credential: results return through callbacks signed with HMAC-SHA256 and are bound to the job id that the application issued, so a forged or replayed callback changes nothing.

 This work is licensed under a Creative Commons Attribution 4.0 International License 

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510146

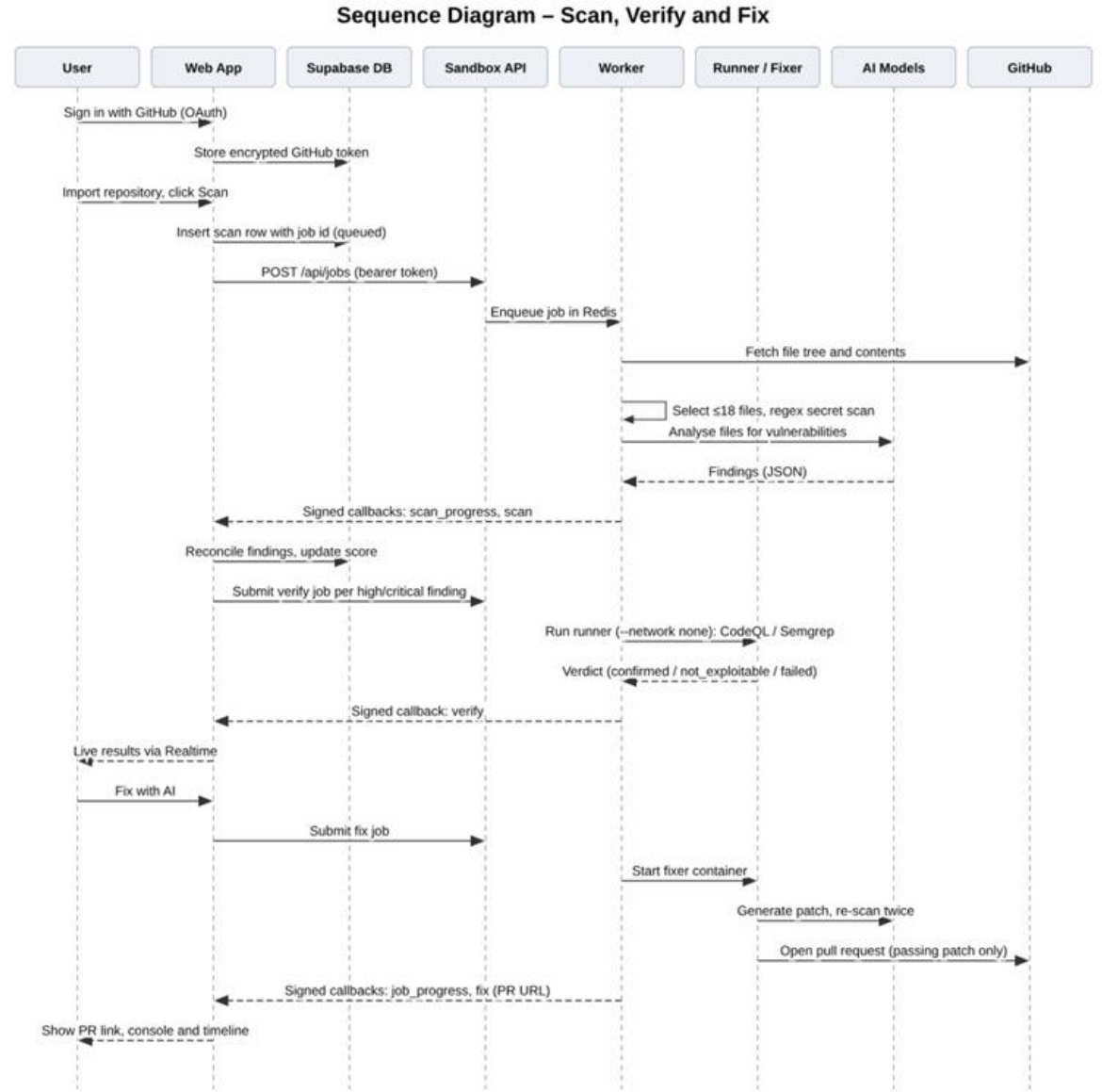



Fig 1. Sequence diagram of the scan, verify and fix workflow

### A. Scanning Stage

Scanning is the first phase of the workflow, as shown in Fig. 1. It consists of five main steps:

**1. Trigger:** The user starts a scan. The application checks repository ownership, rejects a second concurrent scan, decrypts the user's GitHub token (stored with AES-256 GCM), creates the scan record with its job id and submits the job to the sandbox.

**2. File Selection:** The worker fetches the repository tree through the GitHub API, excludes generated and vendored folders, minified files and lockfiles, and ranks the remaining files by security relevance, with secret-bearing files such as .env first. At most 18 files and 250 KB are selected.

**3. Secret Detection:** Deterministic patterns scan the selected files for leaked credentials. Any leaked secret is reported as a critical finding and caps the repository's security score.

**4. LLM Analysis:** Google Gemini 2.5 Flash analyses the files and returns structured findings with title, severity, CWE, CVSS score, file location, explanation and remediation. All model output is shape-checked and truncated before storage, because it is treated as untrusted input.

**5. Persistence and Live Progress:** The worker reports progress at each stage through signed callbacks. The application reconciles new findings with existing ones, so a vulnerability keeps one record across scans, and streams log lines to the browser with Supabase Realtime.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510146

**B. Verification and Remediation Phase**

This phase confirms the serious findings and helps the developer fix them:

**1. Sandbox Verification:** Each high or critical finding becomes a verification job. The worker clones the repository through a git credential helper, so the token never appears in command arguments or in .git/config, and mounts the code read-only into a runner container with no network, dropped capabilities and a hard timeout.

**2. Static Analysis Verdict:** CodeQL with its security-extended queries verifies JavaScript, TypeScript and Python code; Semgrep with bundled offline rules covers other languages and acts as a fallback. Results are matched to the CWEs expected for the finding's category, and the finding is marked confirmed, not exploitable or failed.

**3. Patch Generation:** On request, a fixer container builds a baseline by scanning the unpatched code twice, then asks a code model (Qwen3-Coder via OpenRouter) for a patch.

**4. Fix Gate:** The patched code is re-scanned with the same analyser used for normal scans. The patch passes only if the target finding is gone and no new medium, high or critical issue appears, and it must pass two independent re-scans. A rejected patch is returned to the model with the re-scan's findings, up to three attempts.

**5. Pull Request and Monitoring:** Only a passing patch is committed to a new branch and opened as a pull request. Repositories opted into weekly scanning are re-scanned automatically, findings missed in two consecutive scans that read their file are resolved, returning ones are reopened as regressions, and a weekly digest is emailed to the user.

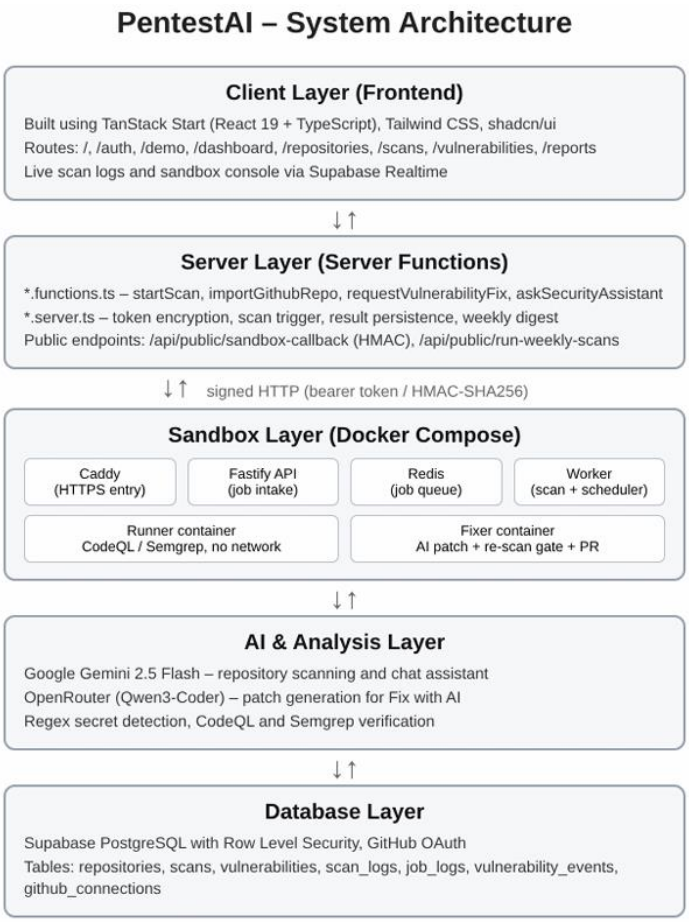



Fig 2. High level design

## IV. RESULTS AND DISCUSSION

Testing confirmed that the system completes the full workflow, from GitHub sign-in and repository import to scanning, sandbox verification and AI-assisted fixing. The scanner consistently prioritised security-sensitive files such as authentication handlers, API routes, database queries and configuration files, and the deterministic secret scanner flagged committed credentials in .env files and source code even when the AI analysis did not mention them. Live progress and sandbox console output appeared in the browser as each stage ran.

DOI: 10.17148/IJARCCE.2026.1510146

Verification in the sandbox separated findings that static analysis could confirm from those it could not, and kept probe crashes distinct from clean results, so a failed check was never presented as a safe one. The fix gate rejected patches that left the vulnerability in place or introduced new issues, and only patches that passed two re scans were opened as pull requests; the developer's code was never modified directly. The core logic is covered by an automated Vitest suite of about 184 test cases, including token encryption, callback authentication and binding, file selection, secret detection, the fix gate and vulnerability lifecycle rules.

The main limitations are that each scan samples at most 18 files, the LLM results are not deterministic between runs, and verification is a second static analysis pass rather than live exploitation; AI patches are re-scanned but not compiled or executed. Overall, the system reduced the effort of a first security review to a few clicks and showed strong potential for continuous use by small teams.

## V. CONCLUSION AND FUTURE WORK

PentestAI provides a practical approach to automated security testing of GitHub repositories by combining LLM-based analysis, deterministic secret detection, sandboxed static-analysis verification and gated AI remediation. It reduces manual effort, lowers false positives compared with AI-only scanning, keeps untrusted code isolated from the application and database, and leaves every change under the developer's control through pull requests.

Future enhancements include scanning more files per run and running CodeQL and Semgrep during the scan itself, adding safe dynamic verification and automatic test execution for AI patches, supporting GitLab and Bitbucket, and posting results as pull request comments on every push.

## VI. ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to the Department of Computer Science and Engineering, K. S. School of Engineering and Management (KSSEM), Bengaluru, for providing an excellent academic and research environment for this work. We extend our heartfelt thanks to our project guide, Mrs. Bindu K.P, Assistant Professor, Department of Computer Science and Engineering, for her valuable guidance, encouragement, and continuous support throughout the preparation of this paper.
We also acknowledge the contributions of the cybersecurity research community and the authors of the reviewed studies whose work provided valuable insights and foundation for this survey and proposed system.

## REFERENCES

[1]. Guo Fangzhou. Design and Implementation of an Intrusion Testing System Based on MVC Framework [D]. Beijing University of Posts and Telecommunications, 2019.
[2]. Cheng Siqiang, Liu Jianxun, Peng Zhenlian, et al. A Study on Code Classification Based on CodeBERT [J]. Computer Engineering and Application, 2023,59(24):277-288.DOI:10.3778/j.issn.1002-8331.2209-0402.
[3]. Duan A, JR R C R.Multi-label Text Classification on TextCNN fused BiLSTM with Attention Mechanism [J]. IAENG International Journal of Computer Science,2025,52(2):
[4]. Chang Jiaqi. Research on Software Vulnerability Detection Methods Based on Deep Learning and Attention Mechanism [D]. Anhui: Anhui University, 2023.
[5]. Zhuang Qiaohui. Research on multi-scale target detection and recognition methods for complex scenes based on deep learning [J]. Information Recording Materials, 2025,26(02):208-211
[6]. Deng Yanning. Nonlinear phase noise compensation method based on machine learning [D]. Beijing University of Posts and Telecommunications, 2022.
[7]. Afraei S, Shahriar K, Madani H S. Developing intelligent classification models for rock burst prediction after recognizing significant predictor variables, Section 1: Literature review and data preprocessing procedure [J].Tunnelling and Underground Space Technology incorporating Trench less Technology Research,2019,83324-353
[8]. Wartschinski L, Noller Y, Vogel T, etal. VUDENC: vulnerability detection with deep learning on a natural codebase for Python [J]. Information and Software Technology, 2022, 144:106809.
[9]. Huang Zhuxing, Yang Yan, Liu Yuting. Research on stock prediction model based on multi-dimensional emotional feature fusion of financial news and trading features.
[10]. Chen Li. Design of Digital Education Platform Based on Python Programming Language [J]. Electronic Technology, 2024,53(01):202-203.

 This work is licensed under a Creative Commons Attribution 4.0 International License 



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510146

[11]. H. Pearce, B. Tan, B. Ahmad, R. Karri and B. Dolan-Gavitt, "Examining zero-shot vulnerability repair with large language models," in Proc. 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2023.

[12]. Z. Li, S. Dutta and M. Naik, "IRIS: LLM-assisted static analysis for detecting security vulnerabilities," in Proc. International Conference on Learning Representations (ICLR), 2025.

[13]. Y. Ding, Y. Fu, O. Ibrahim, C. Sitawarin, X. Chen, B. Alomair, D. Wagner, B. Ray and Y. Chen, "Vulnerability detection with code language models: How far are we?," in Proc. IEEE/ACM 47th International Conference on Software Engineering (ICSE), Ottawa, Canada, 2025.

[14]. M. Meli, M. R. McNiece and B. Reaves, "How bad can it git? Characterizing secret leakage in public GitHub repositories," in Proc. Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 2019.

[15]. H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt and R. Karri, "Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions," in Proc. 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2022.

[16]. X. Zhou, T. Zhang and D. Lo, "Large language model for vulnerability detection: Emerging results and future directions," in Proc. 46th IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), Lisbon, Portugal, 2024.

[17]. M. H. P., K. K. R., G. R., M. G. S., B. K. P., "PentestAI: AI-Powered Penetration Testing for GitHub Repositories," Undergraduate Research Project Report, K. S. School of Engineering and Management, Bengaluru, India, 2026.

 This work is licensed under a Creative Commons Attribution 4.0 International License 