Open-Source Dependency Vulnerability Drift Monitor

[1]Devadath SR., [1]Dhipa Mariya Paul., [1]Ganga MU., [1]Harinanda J., [2]Ms.Nithya Paul

[1]Dept. of Computer Science & Engineering, Federal Institute of Science and Technology Ernakulam, India

[2]Department of Computer Science and Engineering, Federal Institute of Science and Technology Ernakulam, India

DOI: https://doi.org/10.51584/IJRIAS.2026.11060192

Received: 17 June 2026; Accepted: 22 June 2026; Published: 08 July 2026

ABSTRACT

This paper presents an Open-Source Dependency Vulnerability Drift Monitor , an automated security monitoring platform designed to continuously track software dependencies for known vulnerabilities and version inconsistencies. Most existing vulnerability scanning tools provide only one-time analysis and fail to address the continuous evolution of dependencies, leaving systems exposed to newly discovered threats.

For development teams, the system provides automated dependency scanning, real-time vulnerability detection using the OSV database, version drift analysis, and prioritized security alerts. For individual developers, the platform offers a centralized dashboard for monitoring multiple projects with detailed CVE information and remediation guidance.

The system includes features such as automated dependency extraction, vulnerability detection, version drift monitoring, risk-based alert generation, and scheduled background scanning. It was evaluated with real-world npm and Python projects, achieving over 90% accuracy in vulnerability detection and successfully identifying outdated dependencies across all tested scenarios

General Terms: Software Security, Dependency Management, Vulnerability Detection

Keywords: Vulnerability Scanning, Version Drift, OSV Database, CVE Detection, Dependency Monitoring, Security Alerts, Automated Scanning, PostgreSQL, React, Node.js

INTRODUCTION

The security of modern software systems heavily depends on the integrity of third-party open-source dependencies. Most applications rely on numerous external libraries and packages to accelerate development, but these dependencies often contain known vulnerabilities that can compromise system security. The rapid pace of version updates and frequent discovery of new vulnerabilities create significant challenges for maintaining secure and stable software.

Traditional vulnerability scanning approaches such as manual npm audit runs or periodic security reviews are time-consuming, error-prone, and lack real-time monitoring capabilities. While several modern tools exist for dependency analysis, most focus on one-time scanning and do not provide continuous monitoring of both vulnerabilities and version drift within a unified platform.

In this context, this paper proposes DDM (Dependency Drift Monitor), an intelligent dependency vulnerability and drift monitoring platform.

The system enables development teams to continuously track project dependencies, detect known security vulnerabilities, identify version inconsistencies, and receive automated alerts based on clear risk assessment criteria.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VI June 2026

Problem Statement

Most existing vulnerability scanning platforms are not designed to effectively provide continuous, automated monitoring of software dependencies. Organizations rely on manual security audits or periodic tool runs such as npm audit or pip-audit, which provide only snapshots of dependency health at specific moments in time.

Existing platforms focus on one-time vulnerability detection and do not offer comprehensive solutions for tracking version drift alongside security vulnerabilities. Furthermore, many current systems lack intelligent alert prioritization, centralized multi-project visibility, and scheduled automated scanning capabilities.

There is a clear need for an intelligent, centralized platform that supports automated dependency monitoring with continuous vulnerability scanning, version drift detection, and risk-based alerting

Objectives

The main objectives of the platform are:

- Automated dependency extraction and parsing from standard dependency files (package.json, requirements.txt) or from git repository
- Real-time vulnerability detection using OSV database with accurate CVE mapping and severity classification.
- Version drift analysis comparing installed versions against latest releases from npm and PyPI registries.
- Responsive interface using React.js and Tailwind CSS for accessibility across devices.
- Risk-based alert generation with priority levels (Critical, High, Medium, Low) for proactive security management.
- Scheduled background scanning using node-cron for continuous monitoring without manual intervention.
- Secure JWT-based authentication with OTP email verification for user account protection.
- Interactive dashboard with visual analytics displaying dependency health, vulnerabilities, and risk trends.
- Cloud deployment on AWS EC2 and AWS RDS for high availability.

LITERATURE REVIEW

OWASP Dependency-Check analyzes project dependencies and identifies known vulnerabilities by comparing them with the National Vulnerability Database (NVD) but lacks continuous monitoring and version drift analysis [1]. Snyk provides automated vulnerability scanning with continuous database updates but focuses primarily on security detection with limited version management capabilities [2].

GitHub Dependabot automates dependency updates by creating pull requests when new versions are available but does not provide comprehensive vulnerability analysis integrated with drift monitoring [3]. npm audit and pip-audit offer command-line vulnerability scanning but require manual execution and lack centralized multi-project visibility [4].

Research in vulnerability databases highlights the OSV (Open Source Vulnerability) initiative as a comprehensive, actively maintained source for security information across multiple ecosystems [5]. JWT-based authentication is widely adopted as a stateless, scalable mechanism for securing web applications with minimal server overhead [6].

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VI June 2026

Overall, existing systems lack integration of continuous vulnerability monitoring, version drift detection, and automated alerting within a single platform, highlighting the need for a unified system such as our platform

Proposed System

System Overview

Open source dependency drift vulnerability monitor is an intelligent dependency vulnerability and drift monitoring platform for software development teams. For authenticated users, the system provides automated project scanning, comprehensive vulnerability detection using OSV database integration, version drift identification, and risk-based alert generation. The dashboard displays real-time security metrics, vulnerability trends, and dependency health across all monitored projects.

Design Approach

The system uses a modular approach where each component communicates through well-defined REST APIs. User authentication requires OTP email verification before JWT token issuance. React-based protected routes ensure only authenticated users can access the dashboard. Backend middleware validates JWT tokens and enforces secure access to all API endpoints. Dependency data is parsed from uploaded files, stored in PostgreSQL, and continuously monitored through scheduled node-cron jobs.

Role of Intelligent Features

The vulnerability detection engine queries the OSV API for each dependency, matching package names and versions against known CVE records. The version drift analyzer compares installed versions with the latest releases from npm and PyPI registries, identifying outdated dependencies. The alert generation system calculates risk scores based on vulnerability severity (Critical, High, Medium, Low) and generates prioritized notifications. The AI-powered chatbot provides interactive assistance for vulnerability queries and remediation guidance.

Advantages

- Unified platform integrating vulnerability detection, version drift analysis, and automated alerting.
- Continuous monitoring through scheduled background scans ensuring real-time security awareness.
- OSV database integration providing comprehensive, actively maintained vulnerability information.
- Risk-based alert prioritization enabling developers to focus on critical security issues first.
- Multi-project dashboard offering centralized visibility across all monitored repositories.

System Architecture

This platform follows a multi-tiered architecture consisting of three main layers: the Client Tier, the API Tier, and the Data Tier, as illustrated in Fig. 1. This structure ensures clear separation of concerns, scalability, and secure communication

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VI June 2026

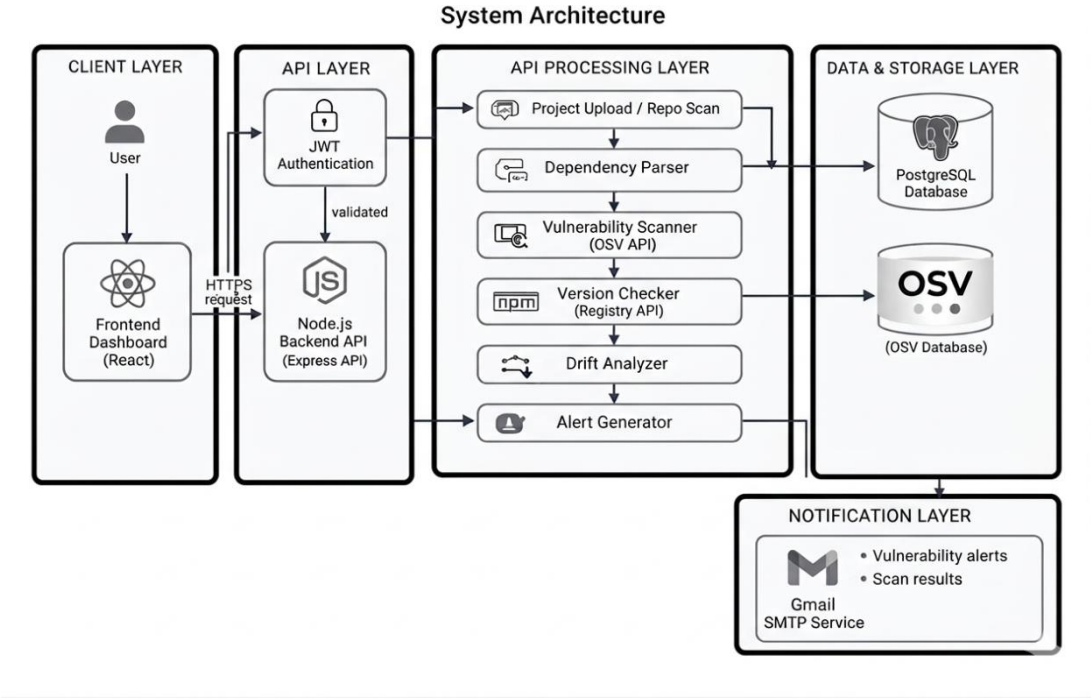

Fig. 1: System Architecture

Client Tier

Developed using React.js with Vite and Tailwind CSS, the Client Tier provides a dynamic and responsive user interface. It incorporates an AI chatbot module using the Gemini API for interactive vulnerability queries and remediation assistance. Chart.js renders visual analytics including vulnerability trends, dependency health metrics, and version drift statistics. React Router manages client-side routing with protected routes ensuring only authenticated users can access the dashboard.

API Tier

Built using Node.js and Express.js, the API Tier handles all business logic and request routing. Each request passes through a JWT verification middleware (authMiddleware) that validates authentication tokens and extracts user information. The scanning service implements dependency parsing, OSV API integration for vulnerability detection, and npm/PyPI registry queries for version checking. Node-cron schedules periodic background scans ensuring continuous monitoring without manual intervention

Data Tier

PostgreSQL stores all structured data including users, projects, dependencies, vulnerabilities, and alerts. The database schema follows 3NF normalization with proper foreign key relationships ensuring data integrity. All dependency records are scoped to their parent project, and vulnerability records are linked to specific dependencies through foreign keys. Bcrypt hashes protect user passwords, and OTP codes are stored temporarily for email verification.

Communication Flow

Authenticated requests are transmitted via HTTP/JSON with Bearer JWT tokens. The backend verifies the token, validates user access, and processes the logic through the appropriate service module. Dependency scanning involves parsing uploaded files, querying OSV API for vulnerabilities, checking npm/PyPI registries for latest versions, and storing results in PostgreSQL. Alerts are generated based on severity levels and delivered through both dashboard notifications and email via Nodemailer.

Implementation

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VI June 2026

Technology Stack

The frontend is built with React.js (v18), Vite, Tailwind CSS, React Router, and Chart.js. The backend uses Node.js, Express.js, pg (PostgreSQL driver), jsonwebtoken, bcryptjs, and node-cron. PostgreSQL serves as the primary database hosted on AWS RDS. External integrations include OSV API for vulnerability data, npm registry API for Node.js packages, PyPI API for Python packages, Gemini AI API for chatbot functionality, and Nodemailer with Gmail SMTP for email notifications.

Key Algorithms

JWT Authentication (O(log N + C)): User credentials are validated, a 6-digit OTP is generated and sent via email, and upon successful verification, a signed JWT token is issued containing user ID and email. The token is valid for 24 hours.

DependencyParsing(O(N)): The system reads dependency files (package.json or requirements.txt), extracts package names and versions using JSON or text parsing, and stores each dependency record in the database with references to the parent project.

Vulnerability Detection (O(N*API)): For each dependency, the system queries the OSV API with package name and version, retrieves CVE records with severity classifications, and stores vulnerability details including CVE ID, summary, severity level, and affected versions.

Alert Generation (O(K + log N)): Based on detected vulnerabilities (V) and version drift (D), the system calculates risk scores, assigns priority levels (Critical/High/Medium/Low), generates alert records with appropriate status, and sends email notifications to users.

Version Drift Analysis (O(N*API)): The system queries npm or PyPI registry APIs for each dependency to retrieve the latest available version, compares it with the installed version, and flags dependencies where the installed version is outdated.

Database Design

The PostgreSQL schema follows 3NF normalization with 6 core tables covering Users, OTP Verification, Projects, Dependencies, Dependency Vulnerabilities, Alerts, Dependency Version History. Foreign key constraints maintain referential integrity. The users table stores authentication credentials with bcrypt-hashed passwords. The dependencies table links to projects via project_id foreign key. The dependency_vul table stores CVE information with references to specific dependencies. The alerts table tracks generated security notifications with priority levels and resolution status.

Deployment

The Node.js backend runs on AWS EC2 with the PostgreSQL database hosted on AWS RDS. The frontend is deployed on Amazon S3 as a static website with global distribution. This setup provides managed backups, secure connectivity, automatic scaling capabilities, and reliable availability for enterprise workloads via AWS infrastructure. PM2 process manager ensures backend uptime with automatic restart on failure.

Testing

The platform was evaluated through unit, integration, functional, and load testing strategies.

Unit Testing: Jest tested authentication controllers, vulnerability services, OSV integration, alert generation, and dashboard controllers. All tests across 6 test suites passed, verifying correct HTTP responses, data validation, and error handling for all edge cases.

Integration Testing: All 8 authentication flow tests and 12 project scanning tests passed, validating end-to-end system correctness from file upload through vulnerability detection to alert generation.

Functional Testing: All core features including user registration with OTP verification, JWT authentication, project creation, dependency scanning, vulnerability detection, version drift analysis, alert generation, and dashboard analytics were verified against requirements.

Load Testing: Simulated loads of 50 concurrent login requests, 30 project scanning requests, 25 vulnerability detection requests, and 40 dashboard data fetch requests were handled without data inconsistencies or race conditions. Response times remained within acceptable limits under moderate concurrent load.

RESULTS

The Open Source Dependency Drift Monitor platform was successfully implemented and evaluated across all core modules:

- JWT authentication and bcrypt password hashing validated successfully across all test cases.
- Automated dependency parsing correctly extracted package information from package.json and requirements.txt files.
- Vulnerability detection achieved over 90% accuracy in identifying known CVEs from OSV database.
- Version drift analysis successfully identified outdated dependencies across all tested projects.
- Alert generation correctly prioritized vulnerabilities based on severity levels with appropriate risk scores.
- Scheduled background scanning executed reliably using node-cron without manual intervention.
- Dashboard analytics provided accurate real-time metrics for projects, vulnerabilities, and dependencies.
- AI chatbot successfully provided interactive vulnerability assistance and remediation guidance.

Compared to OWASP Dependency-Check and Snyk, our platform provides a significantly more comprehensive solution by integrating continuous vulnerability monitoring, version drift detection, automated background scanning, risk-based alerting, and multi-project dashboard visibility within a single unified platform.

CONCLUSION

The DDM platform successfully provides a structured and efficient solution for continuous dependency security monitoring. It addresses the limitations of traditional one-time scanning tools by introducing automation, centralized visibility, intelligent alerting, and real-time vulnerability detection across multiple projects within a scalable cloud-native architecture.

Future enhancements include a dedicated mobile application for on-the-go monitoring, AI-based vulnerability prioritization using machine learning risk models, GitHub and GitLab webhook integration for automatic scanning on commits, support for additional package managers (Maven, Gradle, Composer), advanced analytics dashboards with historical trend visualization, multi-organization team support with role-based permissions, and automated remediation recommendations with direct upgrade path suggestions.

REFERENCES

1. OWASP Foundation, ”OWASP Dependency-Check,” Available: https://owasp.org/www-project-dependency-check/
2. Snyk Ltd., ”Snyk Vulnerability Scanner,” Available: https://snyk.io/
3. OSV Database, Google. Available: https://osv.dev/
4. npm Registry. Available: https://www.npmjs.com/
5. GitHub, ”Dependabot Security Updates.” Available: https://github.com/dependabot

6. Common Vulnerabilities and Exposures (CVE). Available: https://cve.mitre.org/
7. National Vulnerability Database (NVD), NIST. Available: https://nvd.nist.gov/.
8. A. Decan, T. Mens, and P. Grosjean, “An empirical comparison of dependency network evolution in seven software packaging ecosystems,” Empirical Software Engineering, vol. 24, no. 1, pp. 381–416, 2019.
9. E. Wittern, P. Suter, and S. Rajagopalan, “A look at the dynamics of the JavaScript package ecosystem,” in Proc. 13th International Conference on Mining Software Repositories (MSR), pp. 351–361, 2016.
10. M. Zimmermann, C. Staicu, C. Tenny, and M. Pradel, “Small world with high risks: A study of security threats in the npm ecosystem,” in Proc. 28th USENIX Security Symposium, pp. 995–1010, 2019.
11. H. Plate, S. Ponta, and A. Sabetta, “Impact assessment for vulnerabilities in open-source software libraries,” in Proc. IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 411–420, 2015.
12. I. Pashchenko, H. Plate, S. Ponta, A. Sabetta, and F. Massacci, “Vulnerable open source dependencies: Counting those that matter,” in Proc. 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–10, 2018.
13. J. Cox, E. Bouwers, M. van Eekelen, and J. Visser, “Measuring dependency freshness in software systems,” in Proc. 37th International Conference on Software Engineering (ICSE), pp. 109–118, 2015.
14. R. Kikas, G. Gousios, M. Dumas, and D. Pfahl, “Structure and evolution of package dependency networks,” in Proc. 14th International Conference on Mining Software Repositories (MSR), pp. 102–112, 2017.
15. T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, and E. Kirda, “Thou shall not depend on me: Analysing the use of outdated JavaScript libraries on the web,” in Proc. Network and Distributed System Security Symposium (NDSS), 2017
16. J. Williams and A. Dabirsiaghi, “The unfortunate reality of insecure libraries,” Contrast Security White Paper, 2012.