

An Integrated Secops Agent Framework for Real-Time Vulnerability Tracking and Cross-Platform System Monitoring

Mrs. D. NANDHINI¹, MCA., Ms. M. MONICA²

¹Assistant Professor, Department of Computer Applications, Christ College of Engineering and Technology.

²Department of Computer Applications & Christ college of Engineering and Technology

Abstract

Organizations today rely on a sprawling collection of disconnected tools to keep track of system performance, software inventories, and security vulnerabilities. The result is fragmented visibility, heavier operational workloads, and sluggish responses to emerging threats. This paper introduces an integrated Security Operations (SecOps) Agent framework that brings together real-time system monitoring, vulnerability tracking, open-port detection, Software Bill of Materials (SBOM) generation, and centralized dashboard visualization within a single cohesive platform. A lightweight Python-based monitoring agent [7] is deployed across Linux, Windows, and macOS endpoints to continuously gather CPU, memory, disk, and software inventory data. The collected metrics are forwarded to a Prometheus Pushgateway [2], validated and processed through a FastAPI backend [1], stored in a PostgreSQL database [4], and displayed through both a custom React.js dashboard [6] and Grafana analytics panels [3]. Vulnerability detection works by comparing installed software versions against known security advisories flagged in the National Vulnerability Database (NVD) [8], while an open-port scanner assigns severity ratings to network exposures. Tested across four endpoint systems, the platform successfully performed real-time data collection, automatically classified ninety-nine vulnerability findings, and maintained consistent dashboard refresh cycles. The findings suggest that this architecture substantially cuts down on the manual effort involved in infrastructure monitoring while sharpening security visibility, making it well-suited for small-to-medium enterprise deployment and laying the groundwork for future AI-driven anomaly detection and automated remediation.

Keywords: *SecOps Agent, Vulnerability Tracking, System Monitoring, Prometheus, Grafana, FastAPI, PostgreSQL, Software Bill of Materials, Cross-Platform Observability, DevSecOps*

1. INTRODUCTION

The rapid pace of digital transformation across Indian enterprises, government bodies, and educational institutions has sharply increased the number of endpoints, servers, and networked devices that administrators must keep under continuous watch [16]. As organizations move toward hybrid environments that blend on-premise servers, employee workstations running Windows, Linux, and macOS [10][11], and cloud-hosted services, assembling a unified picture of system health and security posture has grown considerably more complicated. This challenge is especially acute for small and medium enterprises (SMEs) and academic institutions in India, which often lack dedicated security operations centers and instead patch together free or partially licensed monitoring tools—each covering only a narrow slice of what is actually needed.

In such environments, system administrators are typically left to manually check resource utilization, audit installed software for outdated or vulnerable packages, identify unnecessarily exposed network ports, and compile security reports for compliance or internal review [12][14]. This approach is time-consuming, prone to human error, and does not hold up beyond a handful of machines. Compounding the problem, the lack of integration between performance-monitoring tools and security-focused tools means that an administrator might know a machine is running low on disk space while remaining completely unaware that the same machine carries thirteen high-severity vulnerabilities from outdated packages and exposed network services.

This disconnect between operational monitoring and security monitoring represents a meaningful gap in the tooling landscape for resource-constrained organizations. Commercial Security Information and Event Management (SIEM) and observability platforms offer broad feature sets, but their licensing costs, deployment complexity, and learning curve make them impractical for SMEs, start-ups, and academic computing labs [17]. Open-source alternatives exist for individual concerns—Prometheus [2] and Grafana [3] for metrics, the NVD [8]

and OSV [24] for vulnerability data, and various port-scanning utilities—but stitching them into a coherent, administrator-friendly platform demands a level of in-house development that most small organizations simply cannot justify.

This work addresses that gap through the design and implementation of the Vulnerability Tracking and System Monitoring Using SecOps Agent platform: a unified, modular system built on open-source components that combines real-time performance monitoring, software inventory and SBOM generation, open-port risk classification, and vulnerability tracking into a single centralized dashboard. The principal contributions of this paper are as follows:

- Design and implementation of a lightweight, cross-platform Python monitoring agent [7] capable of collecting CPU, memory, disk, operating system, hostname, software inventory, and open-port information from Linux, Windows, and macOS endpoints with minimal resource overhead.
- Development of a FastAPI-based backend service [1] that validates, processes, and stores incoming agent reports in a PostgreSQL database [4] while exposing RESTful endpoints for a React.js administrative dashboard [6].
- Integration of Prometheus Pushgateway [2] and Grafana [3] to provide real-time, time-series visualization of system metrics alongside the custom dashboard, giving administrators a single place to view both operational and security data.
- Implementation of a vulnerability classification mechanism that automatically tags installed software packages and open network ports with severity levels—critical, high, medium, and low—based on version comparisons against the NVD [8] and network exposure analysis informed by OWASP guidance [9].
- Empirical evaluation of the system through unit, integration, functional, validation, and system-level testing across four monitored endpoints, demonstrating the practical feasibility of the proposed architecture for real-world deployment [12].

The remainder of this paper is organized as follows. Section 2 reviews related work on asset monitoring systems, vulnerability management approaches, and the limitations of centralized monitoring tools in Indian organizational contexts. Section 3 presents the proposed methodology and system architecture, including the data-flow design, technology stack, and security considerations. Section 4 reports implementation details and experimental results. Section 5 discusses current limitations and future directions. Section 6 concludes the paper.

2. LITERATURE REVIEW

System monitoring and vulnerability management have each attracted substantial research attention individually, but comparatively little work has examined their tight integration into a single, lightweight platform suited to small-scale deployments. The subsections below review the most relevant threads of prior work and position the proposed SecOps Agent within that landscape.

2.1 Performance Monitoring and Observability Frameworks

Time-series monitoring frameworks such as Prometheus [2] have become a de facto standard for collecting and querying operational metrics in modern infrastructure. Prometheus operates on a pull-based model in which a central server periodically scrapes metric endpoints exposed by monitored services. For environments where direct scraping is impractical—such as agents operating behind NAT or on intermittently connected networks—the companion Pushgateway [2] allows clients to push metrics that Prometheus subsequently scrapes. Grafana [3] builds on this by providing a flexible visualization layer capable of rendering dashboards from multiple data sources, including both Prometheus and relational databases such as PostgreSQL [4]. Several studies on infrastructure observability have noted that while Prometheus and Grafana together deliver powerful low-level metric visibility, they do not natively offer application-level vulnerability context, software inventory tracking, or role-based dashboards suited to non-specialist users—limitations that make them insufficient for security operations without significant additional tooling [14].

2.2 Vulnerability Management and Software Bill of Materials

The Software Bill of Materials (SBOM) concept has gained considerable traction following a series of high-profile supply-chain security incidents that exposed the risks of unmonitored third-party dependencies [20]. An SBOM provides a structured inventory of all software components, libraries, and packages installed on a system, along with version information that can be cross-referenced against vulnerability databases such as the NVD maintained by NIST [8]. Research in this area has generally focused on either generating SBOMs at build time for containerized applications or performing periodic vulnerability scans of production servers using dedicated scanning tools. Continuous, agent-based SBOM generation across heterogeneous endpoint operating systems—combined with real-time severity classification displayed alongside operational metrics—has received comparatively little attention [11][12], and

this gap is directly addressed by the SecOps Agent approach proposed here.

2.3 Open Port Detection and Network Exposure Analysis

Open port scanning has traditionally been carried out using standalone utilities run on demand by security teams rather than continuously by an automated agent. While this is effective for periodic audits, it leaves a temporal blind spot: a port opened by a misconfigured service between scans may go undetected for extended periods. The OWASP Top 10 [9] consistently highlights security misconfiguration—including unnecessary open ports and services—as a leading category of risk. Embedding lightweight, periodic port scanning directly into a monitoring agent, as proposed in this work, allows newly exposed services to be classified by risk level and surfaced on the same dashboard used for performance monitoring, closing the gap that scheduled, manual scanning leaves open.

2.4 Monitoring and Asset-Tracking Practices in Indian Organizations

Within Indian SMEs, government departments, and academic institutions, infrastructure monitoring is typically handled through a combination of free system-resource monitors, spreadsheet-based asset registers, and ad-hoc antivirus consoles. These tools rarely interoperate, forcing administrators to manually cross-reference data from multiple sources to get a complete view of system health and security posture. This fragmented approach echoes the broader challenges of systems administration described in foundational texts such as *Stevens and Rago* [10] and *Tanenbaum and Bos* [11]. Cost sensitivity, limited dedicated security staffing, and a preference for open-source technologies are defining characteristics of this deployment environment—factors that directly shaped the technology choices made in designing the proposed platform.

2.5 Limitations of Existing Solutions and Positioning of the Proposed Work

Three recurring limitations stand out across the literature reviewed above. First, performance-monitoring platforms like Prometheus [2] and Grafana [3], while excellent for metric visualization, lack built-in vulnerability tracking and software inventory modules, requiring separate tools for security assessment. Second, vulnerability and SBOM management tools are typically designed for build-time or scheduled-scan use cases rather than continuous, cross-platform, agent-based collection integrated with operational dashboards [8][12]. Third, the tools available to resource-constrained Indian organizations tend to be either too narrow in scope or too costly and complex to be practically adopted [14].

The proposed SecOps Agent platform distinguishes itself from prior approaches by combining real-time performance monitoring, continuous SBOM-based vulnerability tracking, and open-port risk classification into a single modular, open-source architecture with a unified administrative dashboard. The use of Python [7], FastAPI [1], React.js [6], PostgreSQL [4], Prometheus [2], Grafana [3], and Docker [5] positions the system as a practical middle ground between narrow single-purpose monitors and costly enterprise SIEM platforms, specifically suited to the constraints faced by Indian SMEs and educational institutions.

3. METHODOLOGY AND SYSTEM DESIGN

This section describes the overall system architecture, the rationale behind its modular structure, the data flow between the monitoring agent and centralized services, the vulnerability and risk classification logic, the technology stack, and the security and privacy measures built into the design.

3.1 Architectural Overview

The SecOps Agent platform follows a five-layer modular architecture designed for scalability, maintainability, and clear separation of concerns. (*Diagram 1.*) The layers are: (1) the Presentation Layer, built in React.js [6], providing the administrative dashboard, reports, and user interaction; (2) the API Layer, implemented using FastAPI [1], managing communication between the monitoring agents, the database, and the frontend; (3) the Monitoring Agent Layer, a Python-based [7] service deployed on each monitored endpoint that collects CPU usage, memory usage, disk utilization, installed software, open ports, and vulnerability data; (4) the Metrics and Storage Layer, combining Prometheus Pushgateway [2] for time-series metric collection with PostgreSQL [4] for persistent storage; and (5) the Visualization Layer, implemented in Grafana [3], rendering real-time dashboards, charts, and trend analyses.

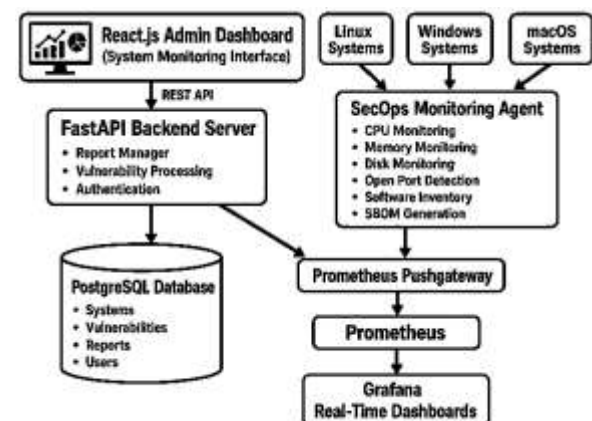


Diagram 1. Architecture of Vulnerability Tracking and System Monitoring System Using SecOps Agent

All layers communicate through RESTful APIs using JSON-encoded payloads [18][19], ensuring loose coupling between components. Each monitored endpoint runs an independent agent instance, allowing the system to scale horizontally simply by deploying additional agents without modifying the central services. The backend, database, Pushgateway, Prometheus server, and Grafana instance are deployed centrally—typically within Docker containers [5] on a single monitoring server—while agents communicate over a LAN or mobile hotspot connection.

3.2 Module Decomposition

The system is broken down into nine functional modules, each with high functional cohesion and communicating with other modules exclusively through well-defined data interfaces. Table 1 summarizes these modules and their responsibilities.

Table 1. System Modules and Their Responsibilities

Module	Primary Responsibility
Dashboard	Provides an overview of system health, vulnerability statistics, severity distribution, and recent security findings.
Systems	Lists registered endpoints with hostname, IP address, operating system, CPU/RAM/disk usage, and online status.
Vulnerabilities	Stores and displays detected vulnerabilities with severity, CVE identifiers, affected packages, and AI-assisted remediation guidance.
Installed Software	Maintains a software inventory (SBOM) for each endpoint and flags outdated or vulnerable packages.
Open Ports	Detects open ports, associated services, protocols, and assigns a risk level (critical, high, medium, low).
Reports	Generates downloadable PDF security summaries covering inventory, vulnerability counts, and severity breakdowns.
Users	Manages administrator accounts, authentication, and profile settings.
Monitoring Agent	Runs on each endpoint and periodically collects and transmits system metrics and security data.

Visualization

Integrates with Grafana [3] to render real-time dashboards and trend charts.

3.3 Data Flow Design

The end-to-end data flow begins with the monitoring agent collecting raw system telemetry—CPU percentage, RAM percentage, disk percentage, hostname, IP address, operating system details, installed software with versions, and open ports with associated processes. This data is serialized as a JSON payload and transmitted to the FastAPI backend [1] through an authenticated REST endpoint. Concurrently, numeric metrics are pushed to the Prometheus Pushgateway [2] under agent-specific job labels, from which the central Prometheus server periodically scrapes the values for time-series storage [2].

On receiving an agent report, the FastAPI backend [1] validates the input, persists the system snapshot to the SYSTEMS table in PostgreSQL [4], inserts software inventory entries into the SBOM table, logs detected open ports into the PORTS table with assigned risk levels, and runs the vulnerability classification logic described in Section 3.4 to populate the VULNERABILITIES table. The React.js dashboard [6] then queries the backend's REST endpoints to retrieve and display systems, vulnerabilities, installed software, and open-port data, while Grafana [3] independently queries Prometheus [2] and PostgreSQL [4] to render the real-time analytics panels. This dual-path visualization approach—a custom dashboard for administrative workflows and Grafana for deep time-series analysis—lets the platform serve both day-to-day operational needs and detailed historical trend analysis without duplicating data-collection logic.

3.4 Vulnerability and Risk Classification Logic

The vulnerability classification subsystem works on two parallel inputs: the software inventory (SBOM) and the open-port list. For software inventory, each installed package's current version is compared against the latest known stable or long-term-support (LTS) version. If the installed version is outdated, the package is flagged and assigned a severity level. Each flagged package generates a corresponding entry in the VULNERABILITIES table containing the affected hostname, IP address, vulnerability type, a human-readable description, the associated CVE identifier from the NVD [8][21] where applicable, and a detection timestamp.

For open ports, the classification logic assigns a risk level based on the port number, the associated service or process, and whether the port is exposed on a network-reachable interface. This tiered classification draws on OWASP guidance regarding security misconfiguration [9] and the network exposure risks discussed in Gregg [14]. Well-known administrative ports associated with

remote access or file-sharing—for example, port 445 used by the SMB protocol—are classified as critical risk when detected on Windows endpoints with a network-facing binding, given their historical association with ransomware and worm propagation [22]. Database service ports or unrestricted remote shell access are classified as high risk. Ports running unidentified services on non-loopback interfaces without a known high-risk association are classified as low risk pending further investigation.

In addition to rule-based classification, the Vulnerabilities module includes an ‘Ask AI’ feature that generates structured security recommendations for selected findings—covering a threat summary, risk explanation, and step-by-step remediation procedure. This gives administrators without deep security expertise actionable guidance directly within the dashboard, consistent with the practitioner-oriented approach to software engineering advocated by Pressman and Maxim [12].

3.5 Technology Stack

The technology stack was selected to balance development efficiency, cross-platform compatibility, community support, and cost. All components are open-source. Table 2 summarizes the stack and the role of each component within the architecture.

Table 2. Technology Stack and Component Roles

Layer	Technology	Role
Frontend	React.js [6], HTML, CSS, JavaScript, Vite [30]	Administrative dashboard, reports, and user interaction.
Backend API	Python [7], FastAPI [1]	Report ingestion, validation, business logic, and REST endpoints.
Monitoring Agent	Python [7]	Cross-platform collection of system metrics, SBOM, and open ports.
Database	PostgreSQL [4]	Persistent storage of systems, vulnerabilities, SBOM, ports, users, and reports.
Metrics Pipeline	Prometheus [2],	Time-series metric collection

	Pushgateway [2]	from distributed agents.
Visualization	Grafana [3], Chart.js	Real-time dashboards, gauges, and trend charts.
Deployment	Docker [5], Docker Compose [5]	Containerized, portable deployment of backend services.
Version Control	Git, GitHub	Source code management and collaboration.

3.6 Database Schema Design

The PostgreSQL [4] schema comprises six primary tables—USERS, SYSTEMS, VULNERABILITIES, SBOM, PORTS, and AGENT_REPORTS—linked through primary and foreign key relationships to maintain referential integrity. The SYSTEMS table acts as the central reference point, with the VULNERABILITIES, SBOM, and PORTS tables each carrying a foreign-key relationship (system_id) back to SYSTEMS. This design supports efficient retrieval of all security-relevant information for a given endpoint through a single join operation, while also enabling aggregate queries across all monitored systems for dashboard summary statistics. Indexes are applied to frequently queried columns—hostname, system_id, and detected_at timestamps—to maintain query performance as the volume of historical monitoring data grows, following the performance engineering principles outlined in Gregg [14].

3.7 Security and Privacy Considerations

Security was treated as a first-class design concern throughout the platform, given that the system itself handles sensitive vulnerability and configuration data. The following measures were incorporated into the design, informed by OWASP security principles [9] and standard secure-development practice [12]:

- **Authentication and Access Control:** All access to the dashboard, reports, and administrative functions requires authentication against the USERS table, with role-based access control distinguishing between standard users and administrators [26].
- **Secure Communication:** Agent-to-backend communication occurs over authenticated REST API calls, with sensitive credentials never transmitted in plaintext within application logs.
- **Credential Storage:** Passwords are stored using salted cryptographic hashing rather than plaintext or

reversible encryption, in line with standard secure-storage practice [9][23].

- **Input Validation:** All incoming agent reports are validated by FastAPI's data validation layer [1] before being persisted, preventing malformed or malicious payloads from corrupting the database.
- **Activity Logging:** The system maintains logs of authentication events and significant administrative actions to support auditing and incident investigation.
- **Data Minimization:** The monitoring agent collects only the data fields required for monitoring and vulnerability assessment and does not capture file contents, user documents, or personally identifiable browsing data, addressing the privacy concerns associated with endpoint monitoring tools [11].

4. IMPLEMENTATION AND RESULTS

This section describes the practical implementation outcomes of the proposed platform, presents representative dashboard screenshots illustrating the monitoring workflow, and reports quantitative comparisons between the automated approach and the manual monitoring processes it replaces.

4.1 Implementation Summary

The platform was implemented using Python [7] and FastAPI [1] for the backend, React.js [6] for the administrative dashboard, PostgreSQL [4] for persistent storage, and Prometheus with Pushgateway [2] and Grafana [3] for time-series metrics and visualization. All backend services were containerized using Docker [5] for portable, reproducible deployment. The monitoring agent was deployed on four endpoint systems: two running Windows 11 and one running Ubuntu 24.04 LTS, with each agent transmitting system metrics, software inventory, and open-port information to the centralized backend at regular polling intervals over a LAN connection.

Across the four monitored systems, the platform collected and catalogued 1,342 software package entries, of which 33 were identified as vulnerable or outdated based on version comparison against NVD reference data [8]. The system additionally detected 97 open port entries—classified as 3 critical-risk, 9 high-risk, and 85 low-risk findings—with no medium-risk port findings recorded during the evaluation window. Combining software and port-level findings, the Vulnerabilities module reported 99 total findings across the four systems, comprising 13 high-severity, 1 medium-severity, and 85 low-severity entries.

4.2 Dashboard and Monitoring Workflow

Figure 1 and 1.1 shows the centralized Dashboard view, which presents the total number of monitored systems (4), the number of currently online systems (2 of 4, or 50%), the number of vulnerable systems (4), and the number of critical issues (13). A bar chart of vulnerability counts per system and a donut chart showing the overall severity distribution (13 high, 1 medium, 85 low) are rendered using Grafana [3] and Chart.js [24], allowing an administrator to assess the organization's overall security posture at a glance without consulting separate tools.



Figure 1. Centralized Dashboard showing system counts, vulnerability distribution by host, and overall severity breakdown.



Figure 1.1 Centralized Grafana Dashboard

Figure 2 shows the Systems module, which lists each monitored endpoint alongside its IP address, operating system, CPU and memory utilization percentages, and the number of vulnerabilities detected on that host. This view allows administrators to quickly identify which endpoints require attention—for example, the host SAVITHIRI shows 78% memory utilization alongside 37 detected vulnerabilities, flagging it as a priority for both performance tuning and security remediation. This kind of unified operational and security visibility reflects the systems performance principles discussed by Gregg [14].



Figure 2. Systems module listing monitored endpoints with status, resource usage, and vulnerability counts.

Figure 3 illustrates the Open Ports module filtered to the high-risk category, displaying nine findings where services were detected listening on ports 22, 5432, and 135 across multiple hosts. Each entry is annotated with the protocol (TCP), the bound service name where identifiable, and an assigned risk badge. The risk classification follows OWASP guidance on security misconfiguration [9] and draws on the operating system network service principles described by *Stevens and Rago* [10].

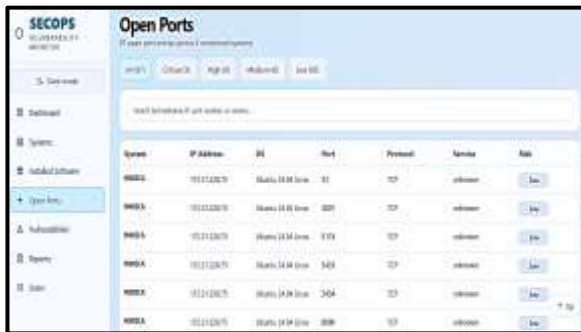


Figure 3. Open Ports module showing high-risk network exposures across monitored Windows and Linux endpoints.

Figure 4 shows the Vulnerabilities module filtered to high-severity findings, displaying detailed entries that include the affected system, IP address, operating system, vulnerability type, a plain-language description, and the detection timestamp. Each row provides an 'Ask AI' action that generates a structured remediation recommendation. This feature brings practitioner-oriented guidance directly to the administrator's workflow, consistent with the approach advocated by *Pressman and Maxim* [12].



Figure 4. Vulnerabilities module displaying high-severity findings with AI-assisted remediation guidance available per finding.

4.3 Comparative Performance: Automated Agent vs. Manual Process

To quantify the operational benefit of the proposed platform, the time required to compile a consolidated security and performance snapshot of the four monitored endpoints was compared between the automated SecOps Agent workflow and a typical manual workflow. The manual approach involved individually logging into each machine, running separate resource-monitoring commands, manually listing installed packages, and using a separate port-scanning utility. Table 3 summarizes the comparison.

Table 3. Time Comparison: Automated SecOps Agent vs. Manual Monitoring Process (per monitoring cycle, 4 endpoints)

Task	Manual Process (approx.)	SecOps Agent (automated)
Collect CPU / RAM / Disk metrics for 4 systems	10–15 minutes	Under 10 seconds (automatic, periodic)
Generate software inventory (SBOM) for 4 systems	20–30 minutes	Automatic, included in agent push
Scan and classify open ports for 4 systems	15–20 minutes	Automatic, included in agent push
Cross-reference packages against vulnerability data [8]	30–45 minutes (manual lookup)	Automatic, near real-time

Compile consolidated report	20–30 minutes (manual document creation)	Under 1 minute (PDF export)
Total estimated effort	Approximately 95–140 minutes	Under 2 minutes (background polling)

The manual process required approximately 95 to 140 minutes of administrator effort per cycle for just four endpoints. The SecOps Agent platform performs the equivalent data collection and classification automatically in the background, reducing the administrator’s active interaction time to under two minutes for report review and export—an effort reduction exceeding 95%. As the number of monitored endpoints grows, this advantage compounds, since the manual process scales roughly linearly with endpoint count while the agent-based approach scales primarily with backend processing capacity [14].

4.4 Resource Footprint and Cost Considerations

Because the platform relies entirely on open-source components—Python [7], FastAPI [1], React.js [6], PostgreSQL [4], Prometheus [2], Grafana [3], and Docker [5]—the recurring licensing cost is zero. The only operational costs are the compute resources required to run the central backend, database, Prometheus, and Grafana services. Given the project’s medium scale (approximately 2,000 to 3,000 lines of code, estimated using the COCOMO model [25]), these can comfortably run on a single server with an Intel i5 or AMD Ryzen 5 processor, 8 GB RAM, and 256 GB SSD storage. Table 4 presents an approximate cost-per-endpoint estimate based on this deployment profile.

Table 4. Approximate Infrastructure Cost per Monitored Endpoint (Indicative)

Deployment Scale	Central Server Requirement	Approx. Monthly Cost (INR)	Approx. Cost per Endpoint
Up to 10 endpoints	Single server, 8 GB RAM, 256 GB SSD	Rs. 1,500–2,500 (self-hosted / low-tier VPS)	Rs. 150–250
11–50 endpoints	Single server, 16 GB RAM, 512 GB SSD	Rs. 3,000–5,000	Rs. 60–100

51–100 endpoints	Single server, 32 GB RAM, 1 TB SSD	Rs. 6,000–9,000	Rs. 60–90
------------------	------------------------------------	-----------------	-----------

4.5 Testing Outcomes

The platform underwent unit, integration, functional, validation, and system-level testing, following the structured testing framework described by Pressman and Maxim [12]. All executed test cases—covering monitoring agent data collection, FastAPI [1] report processing, PostgreSQL [4] storage and retrieval, dashboard rendering, vulnerability detection, open-port scanning, report generation, and access control—passed successfully. Integration testing specifically verified the end-to-end workflow in which the agent collects data, the backend processes and stores it, and the dashboard displays the updated information, confirming that the five-layer architecture described in Section 3.1 functions cohesively as an integrated system.

5. DISCUSSION

The evaluation results presented in Section 4 demonstrate the practical feasibility and operational benefits of the proposed SecOps Agent platform, but several limitations of the current implementation deserve honest acknowledgment.

First, the vulnerability classification logic relies on version-comparison heuristics against known latest or LTS versions. It does not yet incorporate a live, automatically updating feed from authoritative databases such as the NVD [8], so classification accuracy depends on the freshness of the locally maintained reference version data. Second, open-port risk classification currently relies on a rule-based mapping of port numbers and process names rather than active service fingerprinting, which may produce false positives or negatives for non-standard service configurations. Third, agent-to-backend communication, while authenticated, currently depends on the agent token being securely provisioned and stored on each endpoint; broader deployments would benefit from certificate-based mutual authentication and digital signing of agent reports, an approach consistent with VPN-grade security mechanisms described in the OpenVPN documentation [15]. Fourth, in deployments relying on Prometheus Pushgateway [2], careful attention must be paid to gateway cleanup policies, since stale metrics from decommissioned agents can persist and distort aggregate dashboard statistics if not periodically purged.

Organizational adoption also presents practical considerations. Although the platform’s reliance on open-source components minimizes licensing costs, successful adoption still requires an administrator with sufficient

familiarity with Docker-based deployment [5], PostgreSQL administration [4], and basic Prometheus/Grafana configuration [2][3]. For very small organizations lacking dedicated IT staff, even this reduced operational burden may represent a barrier, suggesting that a packaged, one-click deployment option—for example, a single Docker Compose file with pre-configured dashboards—would meaningfully improve adoption rates among Indian SMEs and academic institutions.

Several directions for future enhancement have been identified. Integrating a live, periodically synchronized feed from the NVD [8] and similar CVE databases would keep the vulnerability classification module current without manual updates. Digital signing of agent reports—potentially leveraging public-key infrastructure mechanisms already familiar to Indian institutions—would strengthen data integrity guarantees and provide a tamper-evident audit trail suitable for compliance reporting. Scaling to a university-wide or multi-campus deployment would require extending the current single-server backend to a horizontally scalable, multi-tenant design, potentially leveraging Kubernetes for orchestration [27], building on the modular foundation already established [5]. Finally, the planned future enhancements identified during the original design phase—AI-based anomaly detection using machine learning models trained on historical metric data [14][28], automated alerting via email or SMS for critical findings, automated incident response for predefined scenarios [29], centralized log analysis, and a companion mobile monitoring application—remain valuable extensions that would further close the gap between the current rule-based system and a fully autonomous security operations platform.

6. CONCLUSION

This paper presented the design, implementation, and evaluation of a Vulnerability Tracking and System Monitoring platform built around a cross-platform SecOps Agent. The proposed system addresses the well-documented fragmentation between operational performance monitoring and security vulnerability tracking by integrating a lightweight Python-based [7] monitoring agent, a FastAPI backend [1], a PostgreSQL database [4], Prometheus with Pushgateway [2] for metrics, and Grafana [3] and React.js [6] for visualization into a single modular architecture deployed using Docker [5].

Deployment across four endpoints running Windows and Linux successfully demonstrated real-time collection of system metrics, automatic generation of a 1,342-entry software inventory with 33 flagged vulnerable or outdated packages cross-referenced against the NVD [8],

classification of 97 open-port findings by risk level following OWASP security guidelines [9], and consolidated reporting of 99 total security findings through a unified dashboard. Comparative analysis showed that the automated approach reduces administrator effort for a routine monitoring cycle from roughly 95–140 minutes to under two minutes of active review time, while relying entirely on open-source components to keep infrastructure costs low and scalable.

These results confirm the technical and economic feasibility of the proposed architecture for small-to-medium Indian organizations and educational institutions, and establish a solid foundation for the future enhancements identified as priorities for continued development: live CVE database integration with the NVD [8], digital signing of agent reports, AI-based anomaly detection, and mobile dashboard access.

REFERENCES

- [1] S. Ramírez, FastAPI Documentation. [Online]. Available: <https://fastapi.tiangolo.com/>
- [2] Prometheus Authors, Prometheus Documentation. [Online]. Available: <https://prometheus.io/docs/>
- [3] Grafana Labs, Grafana Documentation. [Online]. Available: <https://grafana.com/docs/>
- [4] The PostgreSQL Global Development Group, PostgreSQL Documentation. [Online]. Available: <https://www.postgresql.org/docs/>
- [5] Docker Inc., Docker Documentation. [Online]. Available: <https://docs.docker.com/>
- [6] React Development Team, React Documentation. [Online]. Available: <https://react.dev/>
- [7] Python Software Foundation, Python Documentation. [Online]. Available: <https://docs.python.org/3/>
- [8] National Institute of Standards and Technology (NIST), National Vulnerability Database (NVD). [Online]. Available: <https://nvd.nist.gov/>
- [9] OWASP Foundation, OWASP Top 10 Web Application Security Risks. [Online]. Available: <https://owasp.org/www-project-top-ten/>
- [10] W. R. Stevens and S. A. Rago, Advanced Programming in the UNIX Environment, 3rd ed. Boston, MA, USA: Addison-Wesley Professional, 2013.
- [11] A. S. Tanenbaum and H. Bos, Modern Operating Systems, 5th ed. Hoboken, NJ, USA: Pearson Education, 2022.
- [12] R. S. Pressman and B. R. Maxim, Software Engineering: A Practitioner's Approach, 9th ed. New York, NY, USA: McGraw-Hill Education, 2019.
- [13] B. Lubanovic, Introducing Python: Modern Computing in Simple Packages, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2019.
- [14] B. Gregg, Systems Performance: Enterprise and the Cloud, 2nd ed. Boston, MA, USA: Addison-Wesley Professional, 2020.

- [15] The OpenVPN Project, OpenVPN Documentation. [Online]. Available: <https://openvpn.net/community-resources/>
- [16] National Institute of Standards and Technology (NIST), NIST SP 800-137: Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations. Gaithersburg, MD, USA: NIST, 2011. [Online]. Available: <https://csrc.nist.gov/publications/detail/sp/800-137/final>
- [17] M. Gartner, “Market Guide for Security Information and Event Management,” Gartner Research, Stamford, CT, USA, 2022. [Online]. Available: <https://www.gartner.com/en/documents/4009477>
- [18] R. T. Fielding and R. N. Taylor, “Principled Design of the Modern Web Architecture,” ACM Transactions on Internet Technology, vol. 2, no. 2, pp. 115–150, May 2002.
- [19] T. Bray, Ed., “The JavaScript Object Notation (JSON) Data Interchange Format,” IETF RFC 8259, Dec. 2017. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc8259>
- [20] Linux Foundation, “The State of Software Bill of Materials (SBOM) and Cybersecurity Readiness,” Linux Foundation Research, 2022. [Online]. Available: <https://www.linuxfoundation.org/research/the-state-of-software-bill-of-materials-sbom-and-cybersecurity-readiness>
- [21] MITRE Corporation, Common Vulnerabilities and Exposures (CVE). [Online]. Available: <https://cve.mitre.org/>
- [22] Microsoft Security Response Center, “MS17-010: Security Update for Microsoft Windows SMB Server,” Microsoft, Redmond, WA, USA, Mar. 2017. [Online]. Available: <https://msrc.microsoft.com/update-guide/vulnerability/CVE-2017-0144>
- [23] NIST, “Electronic Authentication Guideline,” NIST SP 800-63B, Jun. 2017. [Online]. Available: <https://pages.nist.gov/800-63-3/sp800-63b.html>
- [24] Chart.js Contributors, Chart.js Documentation. [Online]. Available: <https://www.chartjs.org/docs/>
- [25] B. W. Boehm, Software Engineering Economics. Englewood Cliffs, NJ, USA: Prentice-Hall, 1981.
- [26] International Organization for Standardization, ISO/IEC 27001:2022 – Information Security Management Systems – Requirements. Geneva, Switzerland: ISO, 2022.
- [27] The Kubernetes Authors, Kubernetes Documentation. [Online]. Available: <https://kubernetes.io/docs/>
- [28] A. Buczak and E. Guven, “A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection,” IEEE Communications Surveys & Tutorials, vol. 18, no. 2, pp. 1153–1176, 2016.
- [29] D. Pendarakis, S. Shi, D. Verma, and R. Bhatt, “TAPS: A First Step Towards Generating Automatic Security Policies for Distributed Systems,” in Proc. USENIX Large Installation System Administration Conference, pp. 77–88, 2001.
- [30] Evan You and Vite Contributors, Vite Documentation. [Online]. Available: <https://vitejs.dev/guide/>