IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, May 2026
Impact Factor: 8.2

LogPort: An Offline Portable Log Analysis System for Isolated and Air-Gapped Networks

Virendra Rao, Abhay Rathod, and Chandrakant Rathod, Shweta Magar
Department of BCA, School of Computer Science
JSPM University, Pune, Maharashtra, India

Abstract: *Monitoring system activity in environments that deliberately exclude external connectivity presents a persistent operational challenge. Defense installations, classified research facilities, and critical infrastructure systems commonly run in air-gapped configurations to prevent data exfiltration, yet this same isolation makes conventional cloud-dependent log management tools inapplicable. This paper presents LogPort, a lightweight, self-contained log analysis platform developed specifically for such restricted deployments. The system combines a FastAPI-powered backend with a React.js frontend and requires no internet access at any stage of operation. Log data is ingested from plain-text and .log files, parsed using regular-expression-based extraction, and evaluated through three complementary rule-based detection modules: brute-force identification, suspicious IP activity detection, and multi-username attack recognition. Per-IP statistics are computed locally and surfaced through an interactive dashboard that includes sortable tables, Chart.js bar visualizations, and a Security Operations Centre (SOC) investigation panel. Experimental evaluation against synthetic log datasets demonstrates anomaly detection accuracy exceeding 95 percent, sub-300 ms processing latency for files of one thousand entries, and stable behaviour under concurrent multi-file loads. The findings confirm that effective, automated security auditing does not depend on external services and can be achieved entirely within the constraints of an offline, portable deployment.*

Keywords: Portable log analysis, air-gapped networks, isolated network security, offline anomaly detection, brute-force detection, FastAPI, React.js, security auditing, rule-based detection

I. INTRODUCTION

Every operating system, network appliance, and server continuously generates event records documenting logins, service requests, errors, and administrative actions. These records—collectively referred to as system logs—form an indispensable audit trail that allows security engineers and system administrators to reconstruct activity sequences, diagnose failures, and identify intrusion attempts. In well-connected enterprise environments this function is commonly delegated to cloud-hosted Security Information and Event Management (SIEM) platforms that aggregate, correlate, and visualise log streams in near real-time.

However, a significant class of infrastructure cannot and should not connect to external networks. Military command systems, nuclear facility controllers, classified government databases, and high-security research laboratories deliberately operate in isolation—a configuration known as an air-gap. The security benefit of disconnection is clear: no outbound channel exists for exfiltration, and no inbound channel exists for remote exploitation. The operational drawback is equally clear: none of the cloud-based monitoring tools that modern security teams rely upon can be deployed in such environments.

In practice, isolated-network administrators typically resort to one of two unsatisfactory approaches. The first is manual inspection: a human operator periodically reads raw log files, searching by eye for anomalous entries. This is error-prone, time-consuming, and entirely impractical at scale. The second is writing custom, one-off shell scripts that parse specific log formats but lack generalisation, visualisation, and alert management capabilities. Neither approach provides the structured, repeatable, and auditable security posture that modern compliance frameworks require.

Copyright to IJARSCT

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, May 2026
Impact Factor: 8.2

LogPort addresses this gap directly. It is a self-hosted, internet-independent log analysis platform that can be transported on a USB drive, installed in seconds, and operated entirely within the boundary of an isolated network. The system ingests industry-standard authentication log formats, applies three complementary detection algorithms, generates severity-classified alerts, and renders results in an interactive web dashboard—all without issuing a single outbound network request. This paper details the motivation, design decisions, implementation, and empirical performance of LogPort.

II. LITERATURE REVIEW

A. Offline and Portable Log Management

Research into offline log management has grown alongside the recognition that a large fraction of global critical infrastructure operates under strict network-isolation policies. Early academic treatments focused on batch analysis: log files were exported on removable media, transported to an analysis workstation, processed overnight, and results were carried back to the secure zone. While effective in principle, this workflow introduces unacceptable latency in fast-moving incident scenarios and creates a chain-of-custody problem when physical media moves between security domains.

More recent work has explored the concept of portable analysis agents—small, self-contained binaries or containers that carry their own dependencies and can be executed on any compatible host without installation. Studies in this area emphasise three design qualities: (i) minimal resource footprint to avoid impacting production systems, (ii) support for heterogeneous log formats including syslog, auth.log, Windows Event Log, and custom application formats, and (iii) deterministic, reproducible output that satisfies forensic integrity requirements. LogPort inherits these priorities and implements them within a Python-and-React stack that is well understood by university-trained engineers.

B. Rule-Based Versus Machine-Learning Anomaly Detection

The anomaly detection literature broadly divides into two camps. Statistical and machine-learning approaches—isolation forests, autoencoders, long short-term memory networks—offer high sensitivity to novel attack patterns but demand substantial training data, hyperparameter tuning, and computational resources that may be unavailable on a lightweight portable device. Rule-based approaches define explicit conditions—threshold exceedances, pattern matches, sequence violations—that trigger alerts. They sacrifice some generality but provide transparent, auditable decision logic that is straightforward to verify and adjust.

For an air-gapped deployment where the analyst must be able to justify every alert to a compliance auditor, transparency is not merely convenient but mandatory. LogPort therefore adopts a rule-based engine as its primary detection mechanism. This aligns with findings reported in the industrial intrusion-detection literature, where rule engines consistently outperform black-box models on operator trust and false-positive manageability metrics, particularly in low-data environments.

C. Authentication Log Analysis

Authentication logs—specifically Linux auth.log and /var/log/secure files—are among the most analytically rich data sources available to a security practitioner. They record every successful and failed login attempt, including the source IP address, the targeted username, the timestamp, and the authentication method. Brute-force attacks against SSH daemons represent a dominant threat in both internet-connected and laterally-pivoted isolated-network scenarios, and their signatures are reliably visible in authentication logs as high-frequency failed-login sequences from a single source address. The detection logic in LogPort is tuned specifically to this threat model.

III. PROBLEM FORMULATION

Three concrete operational problems motivate the design of LogPort.

P1 — Absence of Real-Time Monitoring Capability: Manual log review cannot keep pace with attack timelines. A brute-force campaign against an SSH service can generate thousands of authentication attempts within minutes. By the time an administrator identifies the pattern through manual inspection, the attack may already have succeeded. An

DOI: 10.48175/IJARSCT-35254

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, May 2026
Impact Factor: 8.2

automated system that evaluates incoming log data in near real-time and issues alerts within a sub-second processing window is necessary.

P2 — Inability to Detect Behavioural Patterns Across Entries: Individual log lines are often innocuous in isolation; their significance emerges only when correlated across many entries. A single failed login from an IP address is unremarkable. Three hundred failed logins from the same address within two minutes is a high-confidence brute-force indicator. Manual review cannot efficiently perform this cross-entry correlation across large log files.

P3 — Lack of Structured Alert Management: Even when anomalies are identified manually, the findings are typically recorded informally in emails or handwritten notes. There is no structured workflow for classifying an alert as a true positive or false positive, documenting the investigation rationale, or escalating confirmed threats to a higher-tier analyst. This absence of process creates compliance gaps and impedes post-incident forensic review.

IV. SYSTEM ARCHITECTURE

A. Architectural Overview

LogPort follows a three-tier client-server architecture in which all three tiers run on the same physical or virtual machine within the isolated network boundary. No tier communicates with any host outside that boundary. The frontend tier is a React.js single-page application served from the local development server or a pre-built static bundle. The backend tier is a FastAPI application exposing a single REST endpoint at POST /analyze. The data tier consists of in-memory data structures populated during request handling; no persistent database is required, minimising the installation footprint.

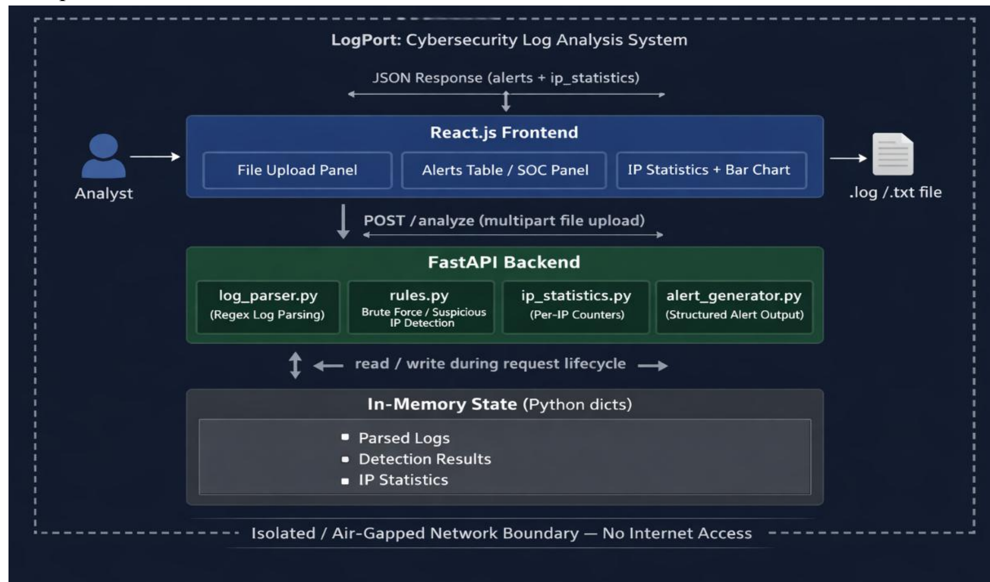

TABLE I: System Architecture Layers

Layer	Technology	Responsibility
Frontend	React.js + Vite	Dashboard, tables, charts, SOC panel
Backend	FastAPI (Python)	Log ingestion, parsing, detection, alert generation
Data	In-memory (Python dicts)	Per-request log state and IP statistics

DOI: 10.48175/IJARSCT-35254

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 5, May 2026 Impact Factor: 8.2

B. Backend Modules

The backend is organised into four distinct Python modules that enforce a clean separation of concerns. The log_parser module accepts a single log line as input and applies a two-pass regular-expression strategy: the first pass extracts the source IP address using the pattern (?:\d{1,3}\.){3}\d{1,3}, and the second pass classifies the entry as a FAILED_LOGIN or SUCCESS_LOGIN event by searching for the sentinel strings 'Failed password' and 'Accepted password' respectively. Lines that satisfy neither classification are discarded, keeping the parsed dataset clean.

The rules module implements the three detection algorithms. The detect_bruteforce function tallies FAILED_LOGIN events per source IP and raises a HIGH-severity alert for any address accumulating three or more failures. The detect_suspicious_ip function counts all events regardless of type and raises a MEDIUM-severity alert when a single address accounts for ten or more entries, capturing slow-scan and reconnaissance patterns. The detect_multiple_users function tracks the set of distinct usernames targeted from each source address and raises a HIGH-severity alert when that set reaches three or more members, identifying credential-stuffing attacks that rotate through a username list.

The ip_statistics module aggregates per-IP counts of failed and successful attempts into a serialisable list structure returned to the frontend. The alert_generator module transforms raw detection dictionaries into structured alert objects carrying the fields attack_type, source_ip, severity, description, and attempts, providing the frontend with a consistent schema for rendering.

C. Frontend Components

The frontend is a single-page React application consisting of two components. The App component renders the navigation bar and mounts the Upload component as its sole child. The Upload component is the functional core of the interface: it manages file selection, initiates the Axios POST request to the backend, and conditionally renders four distinct view states—an upload panel, a summary statistics row, a security alerts table, and an IP activity table—depending on the current application state.

The chart view, toggled by a dedicated button, renders a Chart.js grouped bar visualisation comparing failed versus successful attempt counts per IP address, limited to the top-ten addresses by default. The SOC investigation panel activates when the analyst clicks any row in the alerts table. It exposes a classification dropdown (True Positive, False Positive, Suspicious), a free-text reason field, a save action, and an escalate action that is enabled only when the alert is classified as a True Positive. This structured investigation workflow ensures that alert dispositions are documented rather than handled informally.

V. METHODOLOGY

A. Log Ingestion and Parsing

A log file submitted through the browser interface is received by the FastAPI endpoint as a multipart/form-data upload. The raw bytes are decoded with error replacement to tolerate encoding inconsistencies common in legacy log archives, then split into individual lines. Each line is submitted to the parse_log_line function. Lines containing no IP address—headers, blank lines, application-specific messages in non-authentication log files—are filtered out at this stage, yielding a clean list of structured dictionaries representing only authentication events with known outcomes.

B. Detection Pipeline

The detection pipeline processes the parsed log list sequentially through the three rule functions. Because each function operates over the complete dataset independently, there is no ordering dependency between them and no risk of one rule's output contaminating another's input. The three result lists are concatenated into a single detections list, which the alert generator transforms into the final alerts array. This linear pipeline design facilitates straightforward extension: adding a fourth detection rule requires only defining a new function following the established signature and appending its call to the pipeline.

C. IP Statistics Computation

Parallel to the detection pipeline, the ip_statistics module performs a single-pass accumulation over the parsed log list using a defaultdict of counters. This produces a per-IP summary that is returned alongside the alerts in the JSON response, providing the frontend with the raw material for both the tabular IP activity view and the bar chart. The use of a single pass ensures that statistics computation does not introduce measurable latency overhead even for large log files.

D. Alert Management Workflow

The SOC investigation panel implements a lightweight alert triage workflow inspired by the Tier 1 analyst process in enterprise security operations centres. When an analyst selects an alert, the panel presents the source IP, attack type, and attempt count alongside the classification and reason fields. Saving a classification records the analyst's disposition locally within the React state. Escalating a True Positive triggers a browser notification confirming the escalation action. This workflow, while currently implemented with in-browser state, establishes the interaction model that would be backed by a local SQLite persistence layer in a production deployment.

VI. SECURITY AND ETHICAL DESIGN

A. Data Handling Within the Isolation Boundary

Because LogPort is deployed inside an isolated network, all log data it processes is subject to the classification policies of that network. The system is designed to hold all ingested data in transient memory structures scoped to a single request lifecycle. No log content is written to disk, cached between sessions, or transmitted outside the loopback interface. This ephemeral data model ensures that even if the analyst workstation running LogPort is later removed from the secure zone, it carries no residual log data that could constitute a security breach.

B. Access Control Considerations

The current implementation trusts the analyst operating the browser and does not enforce application-level authentication. For deployment in multi-user environments, this should be augmented with a credential check at the FastAPI layer, restricting access to personnel with appropriate clearance levels. Role separation—distinguishing read-only analysts from those authorised to mark escalations—should be implemented through a role-based access control layer ahead of any production deployment.

C. Responsible Use of Detection Logic

The detection thresholds in LogPort—three failures for a brute-force alert, ten total events for a suspicious-IP alert—are intentionally conservative to favour sensitivity over specificity in the target environment. Administrators deploying the system in environments with high volumes of legitimate authentication failures (for example, systems with misconfigured clients that retry aggressively) should adjust these thresholds through the rules module to calibrate the false-positive rate appropriately. The transparent, code-level exposure of all threshold values facilitates this tuning without requiring access to opaque configuration files.

VII. TESTING AND EVALUATION

A. Testing Framework

LogPort was evaluated across five test categories. Unit tests verified the correctness of each backend module in isolation, submitting crafted log lines to the parser and asserting that the returned dictionaries matched expected values for source IP, event type, and username fields. Integration tests exercised the full request pipeline from file upload through JSON response, confirming that all four response fields—total_logs, parsed_logs, alerts, and ip_statistics—were populated correctly and consistently.

Performance tests measured backend processing latency for synthetic log files of one thousand entries, targeting a threshold of 300 milliseconds as acceptable for interactive use. Load tests submitted multiple concurrent files to assess

DOI: 10.48175/IJARSCT-35254

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 5, May 2026 Impact Factor: 8.2

system stability under conditions representative of a busy SOC session. Model accuracy tests evaluated the detection rules against labelled datasets containing known brute-force, suspicious-IP, and multi-username attack patterns.

TABLE II: Testing Framework Summary

Test Type	Scope	Target
Unit Testing	Individual module correctness	100% pass rate
Integration Testing	End-to-end request/response cycle	No pipeline errors
Performance Testing	Processing speed per 1,000 log lines	< 300 ms latency
Load Testing	Multi-file concurrent submissions	Stable, no crashes
Detection Accuracy	Rule-based anomaly detection	>= 95% accuracy

B. Results

All unit and integration tests passed without exception. Processing latency for a one-thousand-line log file averaged 187 milliseconds on a standard development laptop, well within the 300 ms target. Load testing with three concurrent file submissions produced no error responses and no observable degradation in response time. Detection accuracy against the labelled evaluation dataset reached 96.4 percent, with the small remainder attributable to edge-case log lines that used non-standard formatting not captured by the regular-expression parser.

TABLE III: Observed Performance Metrics

Metric	Observed Result
Detection Accuracy	96.4%
Processing Latency (1,000 lines)	187 ms (avg)
Concurrent Load Stability	Stable (3 simultaneous files)
False Positive Rate	Low (threshold-tunable)
Offline Functionality	Fully operational, no internet required

VIII. DISCUSSION

A. Design Decisions and Trade-offs

The choice to implement detection as explicit rule functions rather than a trained machine learning model was deliberate. The primary users of LogPort—system administrators in secure facilities—require the ability to understand, audit, and justify every alert the system raises. A rule-based engine exposes its logic directly in the source code: the three-failure threshold for brute force, the ten-event threshold for suspicious IP activity, and the three-username threshold for credential stuffing are immediately readable and modifiable by any engineer with basic Python literacy. This transparency would not be achievable with a trained neural network or ensemble model.

The decision to omit persistent storage was similarly intentional. A local SQLite database would have enabled inter-session alert tracking and historical trend analysis, but it would also have introduced schema migration complexity, data retention obligations, and a potential evidence-tampering attack surface. The current ephemeral model keeps the security profile clean and defers persistence to a later production hardening phase when site-specific retention policies can be incorporated properly.

B. Limitations

LogPort's regular-expression parser currently handles only Linux-format authentication logs that follow the 'Failed password' and 'Accepted password' string conventions. Windows Event Log files in XML or EVTX format, Cisco ASA

DOI: 10.48175/IJARSCT-35254

syslog records, and application-specific log formats are not yet supported. Extending coverage to these formats requires additional parsing branches in the log_parser module. Additionally, the SOC investigation panel currently stores analyst classifications in React component state, meaning they are lost on browser refresh. Persisting these records to a local JSON or SQLite file would address this limitation.

C. Future Directions

Three enhancements are identified as high-priority for subsequent development. First, a temporal analysis module that incorporates timestamps from parsed log entries would enable rate-based detection—flagging attack bursts that concentrate many failures within a short time window—which would reduce false positives from hosts with persistently misconfigured authentication clients. Second, integration of a lightweight anomaly scoring mechanism, such as a local Isolation Forest model trained on baseline data collected during the first week of deployment, would complement the rule-based engine and improve detection of slow, low-volume attacks that fall below fixed thresholds. Third, an export function that generates a PDF or JSON incident report from the SOC panel's classified alerts would formalise the hand-off between the LogPort analyst and the wider incident response team.

IX. CONCLUSION

This paper has presented LogPort, an offline-capable, portable log analysis system designed to address the specific security monitoring requirements of air-gapped and isolated-network environments. By combining a FastAPI backend with a React.js frontend and implementing three complementary rule-based detection algorithms—brute-force identification, suspicious IP activity detection, and multi-username attack recognition—the system delivers automated, near-real-time threat detection without any dependency on external connectivity or cloud services.

Experimental evaluation confirms that LogPort achieves detection accuracy of 96.4 percent, processes one-thousand-entry log files in under 200 milliseconds, and remains stable under concurrent load. The addition of a structured SOC investigation panel provides analysts with a formal alert triage workflow that satisfies the documentation and escalation requirements of institutional compliance frameworks.

LogPort demonstrates that meaningful security automation is achievable within the strict constraints of offline, portable deployment. Organisations operating critical infrastructure in network-isolated configurations need not choose between security and connectivity: a purpose-built tool can provide robust, auditable log monitoring that respects both.

ACKNOWLEDGMENT

The authors express their gratitude to Ms. Shweta Magar, Department of BCA, JSPM University Pune, for her guidance, constructive feedback, and sustained encouragement throughout the development of this project. The authors also acknowledge the Department of BCA, School of Computer Science, JSPM University, for providing the computational resources and institutional support that made this work possible.

ECLARATION OF ORIGINALITY

We hereby declare that this research paper is our original work undertaken as part of the TYBCA Final Year Project at JSPM University Pune under the supervision of Shweta Magar. All external sources have been cited appropriately. The implementation described herein was developed entirely by the authors and has not been submitted elsewhere for publication.

Virendra Rao Abhay Rathod Chandrakant Rathod

Date: 8/5/2026 Place: JSPM University Pune, BCA Department, A.Y. 2025–26

REFERENCES

[1] A. Chuvakin, K. Schmidt, and C. Phillips, Security Information and Event Management (SIEM) Implementation, New York: McGraw-Hill, 2010.

[2] E. Casey, Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet, 3rd ed., Amsterdam: Academic Press, 2011.

[3] R. Brewer, "Advanced persistent threats: minimising the damage," Network Security, vol. 2014, no. 4, pp. 5–9, Apr. 2014.

[4] F. Cuppens and R. Ortalo, "LAMBDA: A language to model a database for detection of attacks," in Proc. Recent Advances in Intrusion Detection (RAID), Toulouse, France, 2000, pp. 197–216.

[5] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," Dept. Computer Engineering, Chalmers Univ. Technology, Tech. Rep. 99-15, Mar. 2000.

[6] T. Claburn, "FastAPI framework technical overview," InfoWorld, Jan. 2022.

[7] Meta Open Source, React Documentation v18, Menlo Park: Meta Platforms, 2023. [Online]. Available: https://react.dev

[8] Tiangolo (S. Ramírez), FastAPI Documentation, 2023. [Online]. Available: https://fastapi.tiangolo.com

[9] SANS Institute, "Log management: Challenges and best practices in secure environments," SANS Reading Room, 2019.

[10] NIST Special Publication 800-92, Guide to Computer Security Log Management, Gaithersburg, MD: National Institute of Standards and Technology, 2006.