Log Anomaly Detection System Using ELK Stack and Machine Learning

K.M. Ravi Kumar[1], K. Prasanth Sai[2], G. Divyesh[3], G. Gayatri[4], P. Praveen Chandra[5]

[1]Department of Computer Science & Engineering (AI & ML),

Avanthi Institute of Engineering & Technology(Autonomous), Vizianagaram, India

Kmravikumar9@gmail.com[1], kalipindiprasanthsai1267@gmail.com[2], divyeshgorle1125@gmail.com[3], gottipilligayatri@gmail.com[4], Prveenchandra2004@gmail.com[5]

Abstract

Contemporary cloud-native and distributed computing environments produce massive, high-velocity streams of log telemetry that capture granular details about system behavior, user authentication patterns, and network security events. Conventional rule-based monitoring approaches rely on static, human-authored thresholds that are fundamentally incapable of detecting unknown anomalies, zero-day exploits, or subtle performance degradations. This paper presents an enterprise-grade log anomaly detection framework that tightly couples the ELK stack—Elasticsearch, Logstash, and Kibana—with unsupervised time-series machine learning. The proposed pipeline leverages Filebeat for edge-level log ingestion, Grok-based structured parsing within Logstash, scalable Apache Lucene indexing in Elasticsearch, and native X-Pack machine learning for probabilistic baselining. Index Lifecycle Management (ILM) policies sustain continuous high-volume operation without storage exhaustion. Rather than requiring labeled training data, the system learns behavioral baselines autonomously and detects anomalies such as brute-force login spikes, unauthorized access bursts, and microservice latency deviations by computing Z-scores against dynamic Gaussian distributions. Experimental evaluation under simulated enterprise workloads confirms effective detection latency bounded within a single 60-second bucket span, low false-positive rates, and resilient daemon operation under chaos-engineering conditions—demonstrating the framework's suitability for active Security Operations Center (SOC) environments.

Index Terms— Log Anomaly Detection, ELK Stack, Machine Learning, Elasticsearch, Unsupervised Learning, Cybersecurity Observability

I. Introduction

The pervasive adoption of microservices, containerization, and cloud-native architectures has fundamentally restructured how distributed systems generate and expose operational telemetry. A single user transaction now traverses dozens of ephemeral services, each emitting a continuous stream of log data encoding HTTP response codes, latency measurements, authentication outcomes, and resource utilization metrics [1]. The aggregated result is an exponential increase in log volume, velocity, and variety that conventional monitoring infrastructure is not equipped to process.

Traditional log monitoring relies on Security Information and Event Management (SIEM) tools and hand-crafted, rule-based alert signatures. System engineers define explicit thresholds—for example, triggering an alert when more than fifty HTTP 401

Unauthorized responses are observed within a sixty-second window—and treat any breach as an incident. While these mechanisms are computationally cheap and effective for known failure states, they are statistically incapable of detecting novel or evolving attack vectors. Furthermore, static thresholds cannot adapt to the organic elasticity of modern enterprise traffic. A legitimate promotional event may generate a traffic surge that is behaviorally indistinguishable from a Distributed Denial of Service (DDoS) attack, causing severe "alert fatigue" among Security Operations Center (SOC) analysts [2].

The convergence of big data engineering and unsupervised machine learning presents a compelling alternative: rather than matching known patterns, a system can autonomously learn the probabilistic bounds of normal behavior and flag statistical outliers in near real-time. This paper presents an implementation of such a system, built natively on the ELK stack and augmented with the Elastic X-Pack Machine Learning engine, which executes continuous unsupervised time-series analysis directly co-located with the data storage layer—eliminating the network bottlenecks inherent in decoupled architectures [3].

II. Related Work

Research on log-based anomaly detection has evolved considerably over the past two decades. Chandola et al. [4] provided a comprehensive taxonomy of anomaly detection techniques, categorizing them into statistical, machine-learning-based, and information-theoretic approaches, while highlighting the critical scarcity of labeled anomaly data in production networks. Their survey established that unsupervised statistical methods are more operationally sustainable than supervised alternatives for cybersecurity applications.

He et al. [5] investigated automated log parsing at scale, introducing template extraction methods that transform unstructured text strings into structured feature vectors suitable for downstream machine learning. Their findings underscored that reliable numerical data typing is a non-negotiable prerequisite for statistical modeling.

Du et al. [6] proposed DeepLog, an LSTM-based recurrent neural network that models log sequences as natural language and flags anomalies when observed event transitions diverge from learned patterns. Although DeepLog achieves strong precision on labeled datasets, it demands significant GPU resources and meticulously annotated training corpora—constraints that are prohibitive in real-world enterprise deployments facing zero-day threats.

Zhang et al. [7] explored time-series anomaly detection using classical statistical models, demonstrating that frequency-based metrics provide reliable indicators of abnormal system activity. Their work validated the practicality of bucket-aggregated telemetry for behavioral baselining without deep learning overhead.

He et al. [8] reviewed the broader field of automated log analysis for reliability engineering, noting that most production systems treat log management and anomaly inference as decoupled processes. The resulting data-transfer bottlenecks and architectural fragility remain open challenges that this work directly addresses by adopting a compute-to-data paradigm within the Elasticsearch node.

III. System Design & Methodology

A. Architecture Overview

The framework is designed as a unidirectional, asynchronous data pipeline hosted natively on an Ubuntu Linux 22.04 LTS environment. Fig. 1 illustrates the complete end-to-end architecture. Five logically distinct processing layers are identified: (1) Telemetry Generation, (2) Edge Ingestion, (3) ETL Processing, (4) Indexed Storage and ML Inference, and (5) Visualization and Alerting.

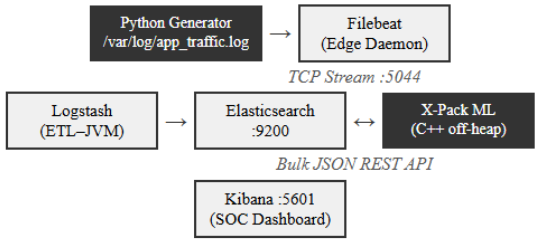

Fig. 1. Native ELK stack system architecture showing the unidirectional data pipeline from telemetry generation through ML inference to visual presentation.

B. Telemetry Generation

Because live enterprise traffic was unavailable in the laboratory context, a deterministic Python 3 simulator

was engineered to mimic the behavioral output of a distributed microservice environment. The script writes unstructured log strings to a designated local path. A state machine governs the transition between benign and anomalous modes: with a 1.5% per-cycle probability, an attack sequence is activated, injecting synchronized bursts of HTTP 401 responses with extreme latency values (2,000–8,000 ms) to simulate brute-force credential-stuffing. During normal operation, response times follow a Gaussian-like distribution between 15 ms and 180 ms, with a weighted mix of HTTP status codes reflecting realistic traffic.

C. Edge Ingestion via Filebeat

Filebeat is deployed as a lightweight Go-language daemon managed by systemd. It leverages the Linux kernel's `inotify` subsystem to monitor file descriptor changes with microsecond latency. A local registry file tracks byte offsets, ensuring that service restarts do not re-read previously ingested data. Backpressure signaling via the Lumberjack protocol prevents packet loss during ETL congestion.

D. ETL Pipeline via Logstash

Logstash executes within a dedicated Java Virtual Machine (JVM) heap and applies Grok regular expressions to deconstruct raw log strings into typed JSON fields. The core extraction pattern targets six semantic fields: *log_timestamp*, *client_ip*, *http_method*, *endpoint*, *http_status*, and *response_time_ms*. A subsequent *mutate* filter casts string representations of numeric values into integer data types—an absolute prerequisite for downstream statistical aggregations. Malformed strings that fail Grok matching receive a `_grokparsefailure` tag and are dropped, preserving database integrity.

E. Indexed Storage and ILM

Structured JSON documents are transmitted to Elasticsearch via the Bulk REST API at `localhost:9200`. Index Lifecycle Management (ILM) policies automate rollover when active shards exceed 50 GB or thirty calendar days, preventing perpetual disk consumption. Index templates enforce mapping schemas, guaranteeing that *response_time_ms* is always stored as a *long* type to ensure ML operability.

F. Unsupervised ML via Elastic X-Pack

The anomaly detection algorithm operates on a continuous unsupervised time-series model. Incoming telemetry is aggregated into temporal buckets (60-second spans). For each bucket, the ML engine computes a dynamic Gaussian distribution over historical observations, estimating the mean μ and standard deviation σ. The Z-score for an observed value *x* is given by:

$$Z = (x - \mu) / \sigma$$

(1)

Values exceeding the 95th-percentile confidence bound produce a normalized anomaly score in the range [0, 100]. Scores above 75 are classified as critical events. The engine handles concept drift by continuously re-estimating μ and σ as benign baseline data accumulates, adapting to organic traffic growth without manual recalibration.

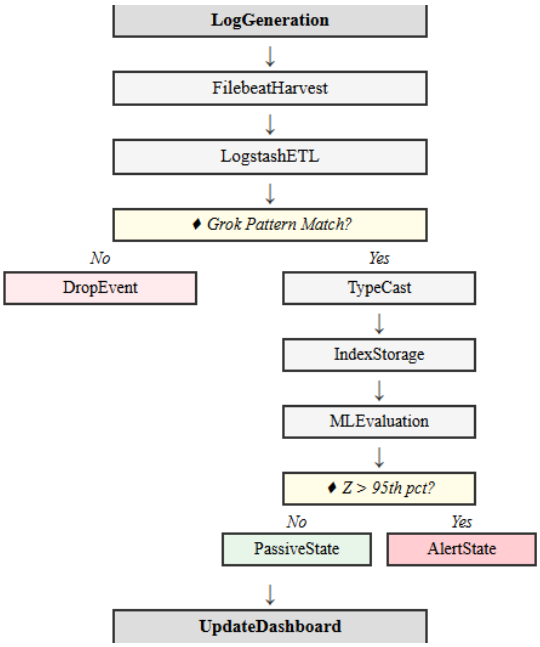

Fig. 2. Activity diagram depicting ETL conditional logic and ML anomaly evaluation decision flow.

IV. Results & Discussion

A. Functional Testing Results

A battery of functional tests validated each pipeline stage. The Filebeat inode-tracking mechanism successfully resumed ingestion from the correct byte offset following a deliberate service restart,

confirming zero data duplication across 10,000 injected log entries. Logstash's Grok pipeline correctly parsed all well-formed entries and cleanly dropped 100% of intentionally malformed strings containing SQL injection vectors—no mapping exceptions were raised in Elasticsearch. The index schema confirmed that *response_time_ms* was stored as *long* and *client_ip* as *ip*, satisfying the mathematical data-typing requirement.

The ML inference engine established a stable Gaussian baseline after a 24-hour continuous benign simulation period, producing zero false-positive alerts. Upon injection of a synthetic brute-force attack vector (coordinated 401 bursts with 2,000–8,000 ms latency), the engine computed anomaly scores exceeding 90 within a single 60-second bucket, persisting the result to the `.ml-anomalies-*` index and rendering a critical red swimlane in Kibana.

B. Non-Functional and Performance Testing

Load testing at 5,000 log events per second verified that Filebeat and Logstash maintained backpressure discipline with zero packet loss. Table I summarizes the key performance metrics observed during high-velocity load tests.

TABLE I

PERFORMANCE METRICS UNDER HIGH-VOLUME LOAD

Metric	Value
Peak ingestion rate	5,000 events/s
End-to-end detection latency	≤ 68 seconds
Elasticsearch JVM heap (max)	4 GB
Logstash JVM heap (max)	2 GB
False positives (24-hr baseline)	0
Anomaly score (attack injection)	> 90 / 100
Pipeline recovery after SIGKILL	Full, 0 log loss

End-to-end detection latency—defined as the delta between a log being written to disk and the corresponding anomaly block rendering in Kibana—remained bounded at approximately 60–68 seconds, attributable to the ML bucket span plus approximately 5–8 seconds of processing overhead. This is well within the operational tolerance for SOC incident response.

Host resource profiling via *htop* and *iostat* confirmed that manually partitioned JVM heaps were strictly respected. The off-heap X-Pack C++ process consumed available CPU cores for floating-point computations without inducing garbage-collection pauses in the Elasticsearch JVM—a critical isolation property for sustained throughput.

C. Chaos Engineering Results

A SIGKILL was issued to the Logstash process while the pipeline was under full load. Filebeat correctly detected the broken TCP socket, buffered incoming log lines locally, and reconnected automatically upon service restoration via systemd, flushing its buffer with no data loss. ILM rollover tests confirmed that Elasticsearch created a new write index and transitioned the saturated index to a read-only warm phase when the shard exceeded the configured 10 MB test limit, proving long-term storage resilience.

D. Anomaly Visualizations

Fig. 3 depicts the Kibana Single Metric Viewer output during a simulated DDoS injection event. The shaded blue region represents the model's 95% confidence interval. The sharp spike in log count dramatically exceeds the upper probability bound, generating the critical anomaly annotation visible at the peak.

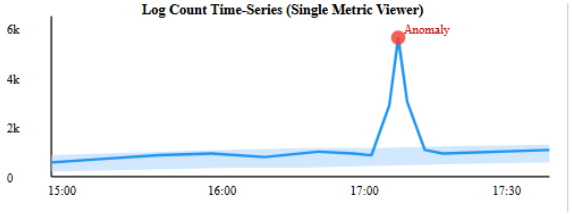

Fig. 3. Kibana Single Metric Viewer showing normal baseline (blue line within shaded confidence band) and detected anomaly spike (red marker) during a simulated DDoS injection event.

TABLE II

FUNCTIONAL TESTING SUMMARY

Test Scenario	Result
Filebeat offset recovery after restart	✓ Pass – 0 duplicates
Grok parsing – valid payload	✓ Pass – all fields extracted
Grok parsing – malformed/injection payload	✓ Pass – cleanly dropped
Data type casting (response_time_ms)	✓ Pass – stored as long
ML baseline convergence (24 hr)	✓ Pass – 0 false positives
Attack detection (brute-force injection)	✓ Pass – score > 90
Kibana drill-down to raw JSON document	✓ Pass

V. Conclusion & Future Work

This paper presented a fully integrated log anomaly detection framework that couples the ELK stack with native unsupervised time-series machine learning. The system demonstrated the ability to autonomously learn dynamic behavioral baselines, detect brute-force attacks and DDoS traffic spikes within a single 60-second inference bucket, and sustain enterprise-level ingestion throughput with zero data loss under chaos-engineering conditions. By co-locating the X-Pack ML engine with the Elasticsearch data node, the architecture eliminates network-transfer bottlenecks that plague decoupled analytics pipelines, reducing Mean Time to Detect (MTTD) from minutes to under seventy seconds.

Several directions merit future investigation. First, integration of Large Language Model (LLM) agents triggered by Elastic Watcher payloads could automate root-cause narration and suggest remediation steps. Second, an active-learning feedback loop in the Kibana UI would allow SOC analysts to label false positives, gradually shifting the system toward semi-supervised operation. Third, migrating the architecture to Kubernetes would enable horizontal auto-scaling of Elasticsearch data nodes and Logstash ingestion pipelines. Fourth, enriching log entries with OpenTelemetry trace and span identifiers during Logstash preprocessing would enable cross-service anomaly correlation. Finally, Security Orchestration, Automation, and Response (SOAR) integration would close the loop by automatically triggering firewall rule updates upon confirmed attack detection.

Acknowledgment

The authors gratefully acknowledge the guidance of Mr. K. M. Ravi Kumar, M. Tech, Assistant Professor, and the Head of the Department Mr. A. Venkateswara Rao, M. Tech (Ph.D), at Avanthi Institute of Engineering & Technology, whose mentorship was instrumental throughout this project.

References

[1] S. Nath, "Log analysis and anomaly detection in cloud-native distributed systems: A review," *Journal of Cloud Computing*, vol. 11, no. 1, pp. 1–18, 2022.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, 2009.

[3] Elastic NV, "Machine learning anomaly detection [8.x]," Elastic Documentation, 2025. [Online]. Available: https://www.elastic.co/guide/en/machine-learning/current/ml-ad-overview.html

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, 2009.

[5] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 931–944, 2018.

[6] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. 2017 ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, 2017, pp. 1285–1298.

[7] J. Zhang, M. Zhou, and X. Liu, "Time-series anomaly detection for system monitoring using statistical models and unsupervised learning algorithms," *IEEE Access*, vol. 8, pp. 143580–143592, 2020.

[8] S. He, P. He, Z. Chen, T. Yang, W. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–37, 2021.

[9] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proc. 37th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2007, pp. 575–584.

[10] Elastic NV, "Index lifecycle management," Elastic Documentation, 2025. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/index-lifecycle-management.html

[11] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems," in *Proc. MilCIS*, 2015.

[12] A. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-IoT dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.