Case Study: Security Observability & Incident Response Automation (ElastAlert2 + CloudWatch + Slack)

Category: Automation & Tooling
Duration: 3 Weeks | **Engagement Type:** Security Monitoring & Response Automation
Tech Stack: OpenSearch (ELK), ElastAlert2, AWS CloudWatch, Lambda, SNS, Slack Webhooks, Python

---

Context

A global SaaS platform operating across multiple AWS accounts lacked real-time visibility into its security posture.

While logs were centralized in **OpenSearch**, the absence of active alerting and correlation meant that:

- 4xx/5xx spikes went unnoticed until after incidents,
- AWS WAF false positives or real attacks weren't triaged in time,
- and engineers relied solely on dashboards — *not actionable alerts*.

The objective was to design a **complete observability and incident response pipeline** — connecting OpenSearch, CloudWatch, and Slack to automatically surface security anomalies, prioritize them, and notify relevant teams with rich context.

Approach

The solution combined **ElastAlert2**, **AWS CloudWatch metrics**, and **Slack-based response workflows** to deliver proactive detection, contextual alerting, and auto-triage.

1. Log Normalization & Indexing

- Segregated logs by index patterns (`uc-be-*`, `us-fe-*`, `waf-logs-*`) for application, API, and WAF sources.
- Implemented **log enrichment pipelines** in OpenSearch ingest nodes to tag logs with environment, severity, and source application metadata.

- Introduced time-based index rotation with ILM policies for efficient retention (30-day hot → 90-day warm).

2. ElastAlert2 Deployment & Rule Configuration

- Deployed **ElastAlert2** as a containerized service with health monitoring via systemd.
- Created custom YAML rule packs for:
    - **Application Anomalies** – spikes in 4xx/5xx error codes per namespace.
    - **WAF Alerts** – label-based matches for `awswaf:managed:aws:core-rule-set:*` triggers.
    - **Authentication Anomalies** – repeated login failures from a single IP or region.
    - **Timeout or Latency Thresholds** – sustained latency >1s in key routes.
- Added grouping and throttling to prevent alert floods during large-scale incidents.

3. CloudWatch Integration

- Linked WAF metrics (`BlockedRequests`, `CountedRequests`) and ALB target health metrics with **CloudWatch Alarms**.
- Configured **Lambda functions** to push alarm context to SNS → Slack in structured JSON format.
- Implemented correlation: WAF blocks + 5xx spikes = escalated "Critical" event in Slack.

4. Slack & Response Automation

- Created dedicated Slack channels:
    - `#4xx-error-logs` for client-side errors
    - `#5xx-error-logs` for backend issues
    - `#waf-alerts` for rule-based triggers
- Configured **interactive Slack messages** with buttons for:

    - Marking an alert as acknowledged
    - Triggering a re-scan (via Lambda invocation)
    - Opening related OpenSearch dashboard view
- Used emojis & tags (🔥, 🛡️, ⚠️) to visually indicate severity.

5. Incident Correlation Layer

- Built a Python microservice (`incident-correlator.py`) that:
    - Merged logs from CloudWatch + OpenSearch + WAF sources
    - Applied deduplication logic
    - Created a unified “incident object” stored in S3 with metadata (status, timestamps, tags)
    - Posted daily summary reports to `#security-digest` channel

Architecture Overview

```
OpenSearch (Logs) → ElastAlert2 → SNS → Slack Alerts

        ├──→ CloudWatch Metrics → Lambda → SNS

        └──→ Python Correlator → S3 (Incidents) → Daily Slack Digest
```

- **Latency:** <10 seconds from detection to alert delivery
- **Average Alert Volume:** 180/day → 35 actionable after deduplication
- **Retention:** 90 days warm + S3 archive for audits

Key Findings & Metrics

Metric	Before	After Implementation
Average Detection Lag	2–6 hours	<10 seconds

Alert Fatigue	High (manual log parsing)	Reduced by 80%
WAF False Positives	20–25/day	3–4/day
Mean Time to Acknowledge (MTTA)	~40 mins	~6 mins

Security & Operational Impact

- Unified monitoring pipeline spanning **WAF, backend logs, and app performance.**
- Enabled **real-time Slack triage** with contextual intelligence.
- Standardized alert workflows across brands and environments.
- Provided **executive visibility** through summarized daily Slack reports.
- Reduced response time dramatically — incidents are now detected, correlated, and assigned *automatically.*

Executive Summary

This engagement transformed the client's passive logging setup into a **proactive incident response ecosystem**.

Through ElastAlert2 rule engineering, CloudWatch metric correlation, and Slack automation, the team achieved:

- **Continuous security awareness** across all production environments
- **Faster, data-driven responses** to anomalies and attack attempts
- **Reduced operational overhead** and improved collaboration between DevOps, Security, and QA teams

By building correlation between logs and metrics, the platform now sustains a **self-alerting architecture** — ensuring no critical event goes unnoticed or unanalyzed.

Deliverables

- 20+ ElastAlert2 rule YAMLs with annotations
- CloudWatch alarm templates and Lambda SNS connectors
- Python “Incident Correlator” script with S3 archiving
- Slack integration workflow (channels, webhooks, alert schema)
- Documentation for alert tuning and new rule onboarding
