Case Study: EKS Cluster Hardening & Runtime Security Monitoring (Anonymized)

Category: Cloud & Infrastructure Security
Duration: 4 Weeks | **Engagement Type:** Kubernetes Security Audit & Runtime Detection Setup
Tools: AWS EKS, Kube-Bench, Kubescape, Falco, Fluent Bit, Prometheus, CloudWatch, OPA Gatekeeper

---

Context

A fintech client running over 60+ microservices on Amazon EKS observed irregular API throttling and pod restarts in production.

Their security team sought a **cluster-level security audit** to identify potential misconfigurations, privilege escalations, and runtime threats that might affect compliance and uptime.

The assessment targeted:

- **EKS node & control-plane security posture**
- **Namespace isolation, RBAC review, and network segmentation**
- **Runtime detection and audit logging**
- **Pod compliance with CIS benchmarks**

Approach

The audit aligned with the **CIS Kubernetes Benchmark v1.7** and **NSA-CISA Kubernetes Hardening Guide (v1.2)**.

1. **Cluster Enumeration & RBAC Mapping**
    - Exported cluster role bindings and service account mappings using `kubectl get` & AWS CLI scripts.
    - Identified service accounts with `cluster-admin` privileges.

2. **Configuration & Policy Review**
    - Ran **Kubescape** and **Kube-Bench** scans for 120+ checks on control-plane, node, and workload settings.
    - Reviewed Pod Security Standards (PSS) enforcement and admission control settings.
3. **Runtime Threat Detection Setup**
    - Deployed **Falco** for behavioral anomaly detection (exec shell in containers, privilege escalation, etc.).
    - Integrated Falco → Slack via webhook alerts for instant triage.
4. **Logging & Observability Enhancement**
    - Configured **Fluent Bit DaemonSets** to ship audit and Falco logs to CloudWatch + ELK stack.
    - Added **Prometheus exporters** for API server, scheduler, and node health metrics.

Key Findings

Severity	Count	Highlight
Critical	2	Default service accounts had `cluster-admin` rights
High	4	Pods using `hostPath` volumes with root privileges
Medium	5	Missing `NetworkPolicy` between internal namespaces
Low	7	Plaintext secrets in ConfigMaps; weak PodSecurity labels

---

Remediation Summary

- Enforced **RBAC least privilege** for all service accounts.
- Applied **namespace isolation** using `NetworkPolicy` and restricted ingress rules.

- Deployed **OPA Gatekeeper** with constraints for `runAsNonRoot` and disallowed capabilities.
- Converted plaintext secrets to **KMS-encrypted Kubernetes Secrets**.
- Implemented **Falco** for runtime detection and **Fluent Bit** → **Slack** alert pipeline.
- Established **cluster baseline policy** for future DevOps onboarding.

Outcome

- Achieved **CIS compliance improvement of 42%** (from 55% → 97%)
- Prevented lateral privilege escalation paths between namespaces
- Enabled 24×7 runtime anomaly detection and Slack notifications
- Strengthened posture for SOC 2 compliance readiness
- Delivered re-usable Terraform templates for Falco & Gatekeeper deployment

---

Executive Summary

This engagement transformed an overprivileged EKS cluster into a **secure, observable, and compliant Kubernetes environment**.

The project bridged DevOps and Security by embedding runtime threat detection and enforcing admission policies without affecting developer velocity.

The client now operates with real-time visibility into container-level behavior, automated rule-based blocking for non-compliant workloads, and measurable CIS compliance reporting integrated with their CI/CD pipeline.
