HORIZONSHIELD ORIGINALS · BOOK 20

Kubernetes Security Hardening

Cluster Hardening, RBAC, Runtime Security & Supply Chain

Complete Kubernetes security hardening guide — cluster configuration, RBAC design, network policies, admission controllers, runtime security with Falco, image signing, and CIS benchmark compliance.

HORIZONSHIELD

ADVANCED

Contents

CHAPTER 01

Kubernetes Attack Surface

Kubernetes' power comes from its complexity — and that complexity creates a large attack surface. Understanding the K8s threat model is essential before hardening can begin.

Kubernetes Components & Attack Vectors

Component	Risk	Attack Vector
API Server	Critical — central control plane	Unauthenticated access, excessive RBAC permissions
etcd	Critical — stores all cluster state	Direct etcd access bypasses all K8s auth
kubelet	High — runs on every node	Anonymous auth, exec into privileged pods
Container runtime	High — executes workloads	Container escape to host node
Service accounts	Medium — workload identity	Automounted tokens, excessive RBAC
Secrets	High — sensitive data	Stored in etcd unencrypted, accessible to all pods by default
Network	Medium — pod communication	No network policies = east-west free-for-all

94%
of K8s clusters have at least one critical misconfiguration (Red Hat survey)

60%
have overprivileged service accounts with cluster-admin or equivalent

55%
have no network policies — any pod can reach any other pod

40%
allow privileged pod execution in at least one namespace

CHAPTER 02

RBAC Design & Least Privilege

Kubernetes RBAC (Role-Based Access Control) is the primary authorization mechanism. Poorly designed RBAC is the most common source of privilege escalation paths in Kubernetes environments.

RBAC Best Practices

```
# AVOID: Binding cluster-admin to service accounts
# This gives every pod in the namespace full cluster access
kubectl create clusterrolebinding DANGEROUS   --clusterrole=cluster-admin   --serviceaccount=default:default

# SECURE: Create minimal Role for specific needs
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: production
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
  # NOT: ["*"] — never use wildcards in production RBAC
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: production
subjects:
- kind: ServiceAccount
  name: my-app-sa
  namespace: production
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

RBAC Audit

```
# Find all cluster-admin bindings (should be minimal)
kubectl get clusterrolebindings -o json | jq '.items[] |
  select(.roleRef.name=="cluster-admin") |
  {name: .metadata.name, subjects: .subjects}'

# Check what a service account can do
kubectl auth can-i --list --as=system:serviceaccount:prod:my-app

# Install rbac-police or rakkess for visual RBAC analysis
kubectl krew install rakkess
kubectl rakkess
```

CHAPTER 03

Network Policies & Runtime Security

Without network policies, every pod can reach every other pod — a flat network that turns any compromised workload into a lateral movement launchpad. Runtime security detects malicious behavior during execution.

Implementing Default-Deny Network Policy

```
# 1. Default deny all ingress and egress in namespace
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: production
spec:
  podSelector: {}  # Applies to all pods
  policyTypes: [Ingress, Egress]
  # No rules = deny all

# 2. Allow specific traffic (e.g., app -> database)
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-app-to-db
  namespace: production
spec:
  podSelector:
    matchLabels: {app: database}
  ingress:
  - from:
    - podSelector:
        matchLabels: {app: backend}
    ports:
    - port: 5432
```

Falco Runtime Security Rules

```
# Detect shell spawned in container (common post-exploit action)
- rule: Shell Spawned in Container
  desc: A shell was spawned in a container
  condition: spawned_process and container and
             proc.name in (shell_binaries)
  output: "Shell spawned (user=%user.name container=%container.name
           image=%container.image.repository cmd=%proc.cmdline)"
  priority: WARNING
  tags: [container, shell, mitre_execution]

# Detect sensitive file access
- rule: Read Sensitive File
  desc: Detect reads of sensitive files (e.g., /etc/shadow)
  condition: open_read and sensitive_files and not
             (proc.name in (allowed_binaries))
  output: "Sensitive file opened (file=%fd.name proc=%proc.name)"
  priority: ERROR
```

CHAPTER 04

K8s Supply Chain & Image Security

Container image supply chain attacks inject malicious code before your workloads even start. Securing the image build, distribution, and runtime pipeline prevents compromised images from reaching production.

Image Security Pipeline

```
# 1. Build with minimal base image (distroless)
FROM gcr.io/distroless/java17-debian12
# No shell, no package manager, no attack surface

# 2. Scan every image before push (GitHub Action)
- name: Scan image with Trivy
  uses: aquasecurity/trivy-action@master
  with:
    image-ref: myapp:${{ github.sha }}
    severity: CRITICAL,HIGH
    exit-code: 1  # Fail build if critical CVE found

# 3. Sign images with Cosign (keyless signing)
cosign sign --yes myregistry.io/myapp:v1.0

# 4. Verify signature before deployment (admission webhook)
cosign verify --certificate-identity=ci@company.com   --certificate-oidc-issuer=https://token.actions.githubusercontent.

# 5. Enforce signed images via Kyverno policy
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-signed-images
spec:
  validationFailureAction: Enforce
  rules:
  - name: verify-image-signature
    match:
      resources: {kinds: [Pod]}
    verifyImages:
    - imageReferences: ["myregistry.io/*"]
      attestors:
      - entries:
        - keyless:
            subject: "ci@company.com"
```

HS

HORIZONSHIELD ORIGINALS

Complete Security Library

HorizonShield Originals are practitioner-written guides designed for real-world implementation — from SMB security foundations to advanced threat hunting, cloud architecture, and regulatory compliance.

AVAILABLE IN THIS SERIES

01 SMB Cybersecurity Guide 2026
02 Penetration Testing Checklist
03 Incident Response Playbook
04 DORA Compliance Guide
05 Cloud Security Hardening
06 Zero Trust Architecture
07 Threat Hunting SOC Operations
08 DevSecOps Implementation
09 Web Application Security
10 Network Security Fundamentals
11 Linux & Systems Security
12 Social Engineering & Human Hacking
13 Malware Analysis & Reverse Engineering
14 Digital Forensics & Incident Response
15 Applied Cryptography
16 Compliance Audit & Risk Management
17 AI & LLM Security Guide
18 OT & ICS Security
19 Mobile Security
20 Kubernetes Security
21 Bug Bounty Methodology
22 Security Architecture for Startups

horizonshield.net/library

Download all 16 books · Access labs · Track your progress

horizonshield.net