1 CKS

Generated on: 2026-01-13 15:04:35 **Version:** 1.0

1.1 Table of Contents

1.2 Overview

The **Certified Kubernetes Security Specialist (CKS)** exam certifies that candidates have the skills, knowledge, and competency to perform a broad range of best practices for securing container-based applications and Kubernetes platforms during build, deployment, and runtime.

1.3 Exam Overview

Detail	Information
Exam Format	Performance-based (hands-on)
Number of Questions	15-20
Duration	2 hours
Passing Score	67%
Certification Validity	2 years
Cost	$395 USD
Retake Policy	1 free retake
Kubernetes Version	1.30
Prerequisite	Must hold valid CKA certification

1.4 Exam Domains & Weights

Domain	Weight
Cluster Setup	10%
Cluster Hardening	15%
System Hardening	15%
Minimize Microservice Vulnerabilities	20%
Supply Chain Security	20%
Monitoring, Logging and Runtime Security	20%

1.5 Prerequisites

- **Valid CKA certification** (required)
- Strong Kubernetes administration skills
- Understanding of Linux security concepts
- Familiarity with container security

1.6 Study Resources

1.6.1 Official Resources

- CKS Exam Curriculum
- Kubernetes Security Documentation
- Kubernetes Security Best Practices

1.6.2 Recommended Courses

- Kubernetes Security Essentials (LFS260)
- CKS with Practice Tests - Udemy

1.6.3 Practice Resources

- Killercoda CKS Scenarios ⭐ **Highly Recommended**
- killer.sh CKS Simulator - Included with exam registration

1.7 Quick Navigation

- 06 - Monitoring, Logging and Runtime Security
- Sample Practice Questions

1.8 Exam Environment

The CKS exam provides:

- Access to multiple Kubernetes clusters
- `kubectl` with auto-completion enabled
- Access to Kubernetes documentation (kubernetes.io)
- A Linux terminal environment
- Root access via `sudo`
- Security tools pre-installed

1.8.1 Allowed Resources During Exam

- kubernetes.io/docs
- kubernetes.io/blog
- trivy.dev/docs
- falco.org/docs
- aquasecurity.github.io/trivy

1.9 Exam Tips

1. **Master security contexts** - runAsUser, runAsNonRoot, capabilities
2. **Know Network Policies** - Default deny, allow specific traffic
3. **Understand RBAC deeply** - Least privilege principle
4. **Practice with security tools** - Trivy, Falco, AppArmor, seccomp
5. **Know Pod Security Standards** - privileged, baseline, restricted
6. **Practice image scanning** - Trivy for vulnerability detection
7. **Understand audit logging** - Configure and analyze audit logs
8. **Practice on Killercoda** - Free hands-on scenarios

1.10 Security Tools Overview

Tool	Purpose
Trivy	Container image vulnerability scanning
Falco	Runtime security monitoring
AppArmor	Linux security module for access control
seccomp	System call filtering
OPA/Gatekeeper	Policy enforcement
kube-bench	CIS Kubernetes benchmark

1.11 Useful Commands

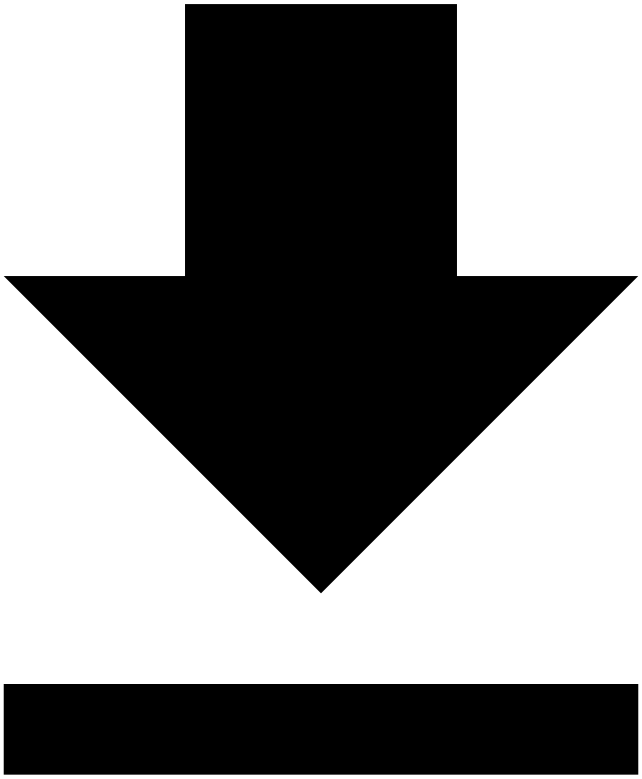
```
# Set alias
alias k=kubectl

# Enable auto-completion
source <(kubectl completion bash)
complete -o default -F __start_kubectl k

# Check API server audit logs
cat /var/log/kubernetes/audit/audit.log | jq .

# Scan image with Trivy
trivy image nginx:1.21

# Check seccomp profiles
ls /var/lib/kubelet/seccomp/

# View AppArmor profiles
aa-status

# Check Pod Security Standards
kubectl label namespace default pod-security.kubernetes.io/
        enforce=restricted

# RBAC verification
kubectl auth can-i --list --as system:serviceaccount:default:mysa
kubectl auth can-i create pods --as jane

# Network Policy testing
kubectl exec -it test-pod -- nc -zv target-service 80
```

1.12 Registration

Register for CKS Exam

> **Note**: You must hold a valid CKA certification before taking the CKS exam.

1.13 Cluster Setup



Download PDF Version

This domain covers securing Kubernetes cluster components and network security.

1.14 Network Policies

1.14.1 Default Deny All Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

1.14.2 Default Deny All Egress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-egress
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Egress
```

1.14.3 Default Deny All Traffic

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

1.14.4 Allow Specific Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend-to-backend
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    - namespaceSelector:
        matchLabels:
          name: production
    ports:
    - protocol: TCP
      port: 8080
```

1.14.5 Allow DNS Egress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-dns-egress
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Egress
  egress:
  - to:
    - namespaceSelector: {}
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - protocol: UDP
      port: 53
    - protocol: TCP
      port: 53
```

1.15 CIS Kubernetes Benchmark

1.15.1 kube-bench

Tool to check Kubernetes against CIS benchmarks.

```
# Run kube-bench
kube-bench run --targets=master
kube-bench run --targets=node
kube-bench run --targets=etcd

# Run specific checks
kube-bench run --targets=master --check=1.1.1

# Output as JSON
kube-bench run --json
```

1.15.2 Common CIS Findings

Check	Description	Fix
1.1.1	API server pod spec permissions	`chmod 600 /etc/kubernetes/manifests/kube-apiserver.yaml`
1.2.1	Anonymous auth disabled	`--anonymous-auth=false`

Check	Description	Fix
1.2.6	RBAC enabled	`--authorization-mode=RBAC`
4.1.1	kubelet service file permissions	`chmod 600 /etc/systemd/system/kubelet.service.d/10-kubeadm.conf`

1.16 Ingress Security

1.16.1 TLS Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: secure-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - secure.example.com
    secretName: tls-secret
  rules:
  - host: secure.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: secure-service
            port:
              number: 443
```

1.16.2 Create TLS Secret

```bash
# Generate self-signed certificate
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout tls.key -out tls.crt -subj "/CN=secure.example.com"

# Create secret
kubectl create secret tls tls-secret --cert=tls.crt --key=tls.key
```

1.17 Cluster Component Security

1.17.1 Secure API Server

```
# /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
  - command:
    - kube-apiserver
    - --anonymous-auth=false
    - --authorization-mode=Node,RBAC
    - --enable-admission-plugins=NodeRestriction,PodSecurityPolicy
    - --audit-log-path=/var/log/kubernetes/audit/audit.log
    - --audit-policy-file=/etc/kubernetes/audit-policy.yaml
    - --encryption-provider-config=/etc/kubernetes/encryption-
        config.yaml
    - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
    - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
```

1.17.2 Secure etcd

```
# /etc/kubernetes/manifests/etcd.yaml
spec:
  containers:
  - command:
    - etcd
    - --cert-file=/etc/kubernetes/pki/etcd/server.crt
    - --key-file=/etc/kubernetes/pki/etcd/server.key
    - --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    - --client-cert-auth=true
    - --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
    - --peer-key-file=/etc/kubernetes/pki/etcd/peer.key
    - --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    - --peer-client-cert-auth=true
```

1.17.3 Secure kubelet

```
# /var/lib/kubelet/config.yaml
authentication:
  anonymous:
    enabled: false
  webhook:
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
authorization:
  mode: Webhook
readOnlyPort: 0
```

1.18 Encryption at Rest

1.18.1 Encryption Configuration

```
# /etc/kubernetes/encryption-config.yaml
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: <base64-encoded-32-byte-key>
      - identity: {}
```

1.18.2 Enable Encryption

```
# Generate encryption key
head -c 32 /dev/urandom | base64

# Add to API server
# --encryption-provider-config=/etc/kubernetes/encryption-
        config.yaml

# Encrypt existing secrets
kubectl get secrets --all-namespaces -o json | kubectl replace -f
        -
```

1.18.3 Verify Encryption

```
# Check if secrets are encrypted in etcd
ETCDCTL_API=3 etcdctl get /registry/secrets/default/my-secret \
  --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key | hexdump -C
```

1.19 GUI Element Security

1.19.1 Disable Kubernetes Dashboard (if not needed)

```
# Delete dashboard
kubectl delete ns kubernetes-dashboard

# Or restrict access with NetworkPolicy
```

1.19.2 Secure Dashboard Access

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: dashboard-policy
  namespace: kubernetes-dashboard
spec:
  podSelector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  policyTypes:
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 10.0.0.0/8
    ports:
    - protocol: TCP
      port: 443
```

1.20 Key Concepts to Remember

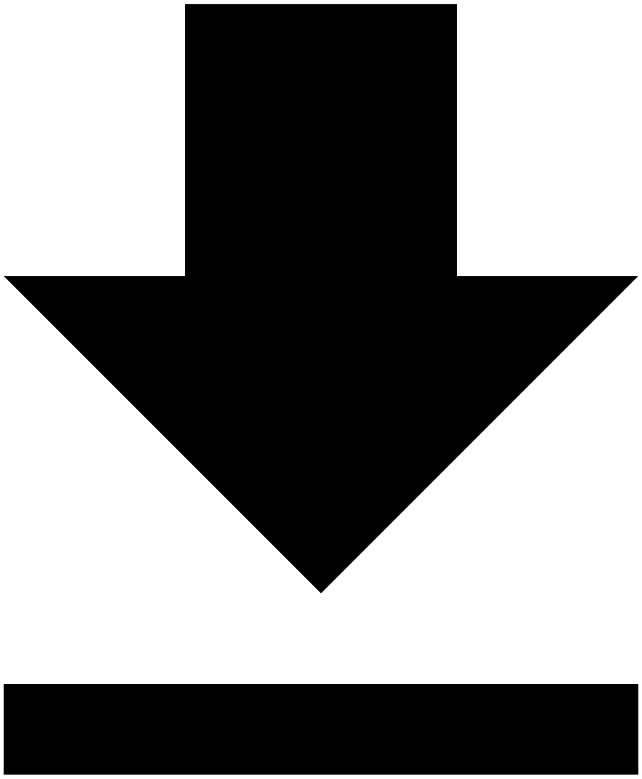
1. **Network Policies** - Default deny, allow specific traffic
2. **CIS Benchmarks** - Use kube-bench to verify compliance
3. **TLS everywhere** - Encrypt all cluster communication
4. **Encryption at rest** - Encrypt secrets in etcd
5. **Disable anonymous auth** - Require authentication

1.21 Practice Questions

1. How do you create a default deny all network policy?
2. What tool checks Kubernetes against CIS benchmarks?
3. How do you enable encryption at rest for secrets?
4. What flag disables anonymous authentication on the API server?
5. How do you verify secrets are encrypted in etcd?

Back to CKS Overview | Next: Cluster Hardening →

1.22 Cluster Hardening



Download PDF Version

This domain covers hardening Kubernetes cluster components and implementing access controls.

1.23 RBAC (Role-Based Access Control)

1.23.1 Role (Namespace-scoped)

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
- apiGroups: [""]
```

```
  resources: ["pods/log"]
  verbs: ["get"]
```

1.23.2 ClusterRole (Cluster-scoped)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["get", "list"]
```

1.23.3 RoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
- kind: ServiceAccount
  name: default
  namespace: kube-system
- kind: Group
  name: developers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

1.23.4 ClusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: managers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
```

```
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

1.23.5 RBAC Commands

```
# Create role
kubectl create role pod-reader --verb=get,list,watch --
        resource=pods -n default

# Create clusterrole
kubectl create clusterrole node-reader --verb=get,list --
        resource=nodes

# Create rolebinding
kubectl create rolebinding read-pods --role=pod-reader --
        user=jane -n default

# Create clusterrolebinding
kubectl create clusterrolebinding read-nodes --clusterrole=node-
        reader --group=developers

# Check permissions
kubectl auth can-i create pods --as jane
kubectl auth can-i list nodes --as jane
kubectl auth can-i --list --as jane
kubectl auth can-i --list --as system:serviceaccount:default:mysa
```

1.23.6 Least Privilege Principle

```
# Bad: Too permissive
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]

# Good: Specific permissions
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list"]
```

1.24 ServiceAccount Security

1.24.1 Create ServiceAccount

```
apiVersion: v1
kind: ServiceAccount
```

```yaml
metadata:
  name: my-sa
  namespace: default
automountServiceAccountToken: false
```

1.24.2 Disable Token Auto-mount

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  serviceAccountName: my-sa
  automountServiceAccountToken: false
  containers:
  - name: app
    image: nginx
```

1.24.3 ServiceAccount Token Projection

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-projected-token
spec:
  serviceAccountName: my-sa
  containers:
  - name: app
    image: nginx
    volumeMounts:
    - name: token
      mountPath: /var/run/secrets/tokens
  volumes:
  - name: token
    projected:
      sources:
      - serviceAccountToken:
          path: token
          expirationSeconds: 3600
          audience: api
```

1.25 Restrict API Access

1.25.1 Disable Anonymous Authentication

```yaml
# /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
  - command:
```

```
    - kube-apiserver
    - --anonymous-auth=false
```

1.25.2 Enable RBAC Authorization

```
# /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
  - command:
    - kube-apiserver
    - --authorization-mode=Node,RBAC
```

1.25.3 Restrict kubelet API

```
# /var/lib/kubelet/config.yaml
authentication:
  anonymous:
    enabled: false
  webhook:
    enabled: true
authorization:
  mode: Webhook
```

1.26 Upgrade Kubernetes

1.26.1 Upgrade Control Plane

```
# Check available versions
apt-cache madison kubeadm

# Upgrade kubeadm
apt-mark unhold kubeadm
apt-get update && apt-get install -y kubeadm=1.30.0-1.1
apt-mark hold kubeadm

# Plan upgrade
kubeadm upgrade plan

# Apply upgrade
kubeadm upgrade apply v1.30.0

# Upgrade kubelet and kubectl
apt-mark unhold kubelet kubectl
apt-get update && apt-get install -y kubelet=1.30.0-1.1
         kubectl=1.30.0-1.1
apt-mark hold kubelet kubectl

systemctl daemon-reload
systemctl restart kubelet
```

1.26.2 Upgrade Worker Nodes

```
# Drain node
kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-
        data

# On worker node
apt-mark unhold kubeadm
apt-get update && apt-get install -y kubeadm=1.30.0-1.1
apt-mark hold kubeadm

kubeadm upgrade node

apt-mark unhold kubelet kubectl
apt-get update && apt-get install -y kubelet=1.30.0-1.1
        kubectl=1.30.0-1.1
apt-mark hold kubelet kubectl

systemctl daemon-reload
systemctl restart kubelet

# Uncordon node
kubectl uncordon <node-name>
```

1.27 Restrict Access to Kubernetes API

1.27.1 API Server Flags

```
# /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
  - command:
    - kube-apiserver
    - --anonymous-auth=false
    - --authorization-mode=Node,RBAC
    - --enable-admission-plugins=NodeRestriction
    - --insecure-port=0
    - --profiling=false
    - --audit-log-path=/var/log/kubernetes/audit/audit.log
```

1.27.2 Network-level Restrictions

```
# Restrict API server access with NetworkPolicy
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: restrict-api-access
  namespace: kube-system
spec:
  podSelector:
```

```
    matchLabels:
      component: kube-apiserver
  policyTypes:
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 10.0.0.0/8
```

1.28 Admission Controllers

1.28.1 Important Security Admission Controllers

Controller	Description
`NodeRestriction`	Limits kubelet permissions
`PodSecurity`	Enforces Pod Security Standards
`ImagePolicyWebhook`	External image validation
`ValidatingAdmissionWebhook`	Custom validation
`MutatingAdmissionWebhook`	Custom mutation

1.28.2 Enable Admission Controllers

```
# /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
  - command:
    - kube-apiserver
    - --enable-admission-plugins=NodeRestriction,PodSecurity
```

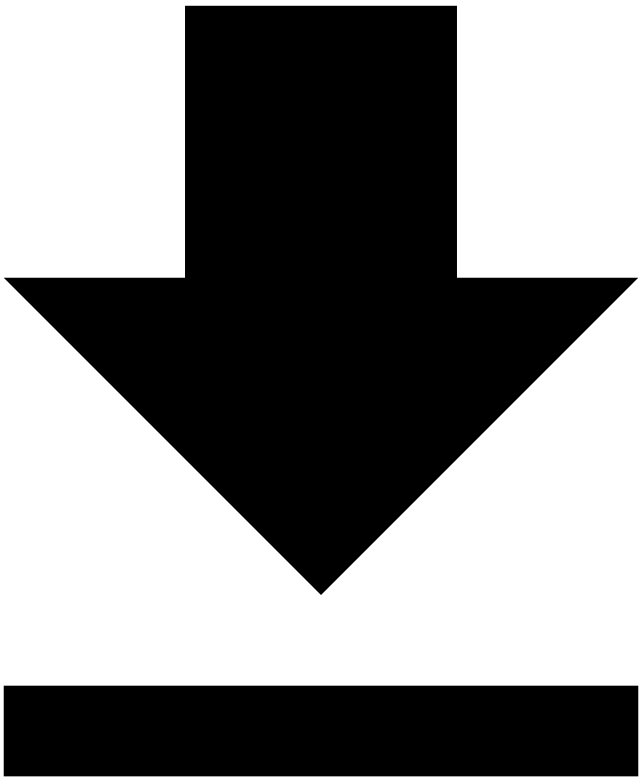
1.29 Key Concepts to Remember

1. **RBAC** - Role, ClusterRole, RoleBinding, ClusterRoleBinding
2. **Least Privilege** - Grant minimum required permissions
3. **ServiceAccount** - Disable auto-mount when not needed
4. **Anonymous Auth** - Disable on API server and kubelet
5. **Admission Controllers** - NodeRestriction, PodSecurity

1.30 Practice Questions

1. How do you check if a user can perform an action?
2. What is the difference between Role and ClusterRole?
3. How do you disable ServiceAccount token auto-mount?
4. What admission controller restricts kubelet permissions?
5. How do you upgrade a Kubernetes cluster securely?

1.31 System Hardening



Download PDF Version

This domain covers hardening the underlying host systems and using kernel security features.

1.32 AppArmor

1.32.1 AppArmor Basics

```
# Check AppArmor status
aa-status
systemctl status apparmor

# List loaded profiles
cat /sys/kernel/security/apparmor/profiles
```

```
# Load a profile
apparmor_parser -q /etc/apparmor.d/my-profile

# Reload a profile
apparmor_parser -r /etc/apparmor.d/my-profile
```

1.32.2 AppArmor Profile Example

```
# /etc/apparmor.d/k8s-deny-write
#include <tunables/global>

profile k8s-deny-write flags=(attach_disconnected) {
  #include <abstractions/base>

  file,

  # Deny all file writes
  deny /** w,
}
```

1.32.3 Use AppArmor in Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: apparmor-pod
  annotations:
    container.apparmor.security.beta.kubernetes.io/nginx:
        localhost/k8s-deny-write
spec:
  containers:
  - name: nginx
    image: nginx
```

1.32.4 AppArmor Profile Modes

Mode	Description
`enforce`	Enforces the policy
`complain`	Logs violations but allows
`unconfined`	No restrictions

1.33 seccomp (Secure Computing Mode)

1.33.1 seccomp Profile Location

```
# Default kubelet seccomp profile path
/var/lib/kubelet/seccomp/
```

```
# Create profile directory
mkdir -p /var/lib/kubelet/seccomp/profiles
```

1.33.2 seccomp Profile Example

```
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "architectures": [
    "SCMP_ARCH_X86_64"
  ],
  "syscalls": [
    {
      "names": [
        "accept4",
        "bind",
        "clone",
        "close",
        "connect",
        "epoll_create1",
        "epoll_ctl",
        "epoll_pwait",
        "execve",
        "exit_group",
        "fcntl",
        "fstat",
        "futex",
        "getdents64",
        "getpid",
        "getrandom",
        "getsockname",
        "getsockopt",
        "listen",
        "mmap",
        "mprotect",
        "nanosleep",
        "newfstatat",
        "openat",
        "read",
        "recvfrom",
        "rt_sigaction",
        "rt_sigprocmask",
        "sendto",
        "set_tid_address",
        "setsockopt",
        "socket",
        "write"
      ],
      "action": "SCMP_ACT_ALLOW"
    }
```

```
  ]
}
```

1.33.3 Use seccomp in Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: seccomp-pod
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: profiles/audit.json
  containers:
  - name: nginx
    image: nginx
```

1.33.4 seccomp Profile Types

Type	Description
`RuntimeDefault`	Container runtime default profile
`Unconfined`	No seccomp filtering
`Localhost`	Custom profile from node

1.33.5 RuntimeDefault seccomp

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: runtime-default-pod
spec:
  securityContext:
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: nginx
    image: nginx
```

1.34 Minimize Host OS Footprint

1.34.1 Reduce Attack Surface

```
# Remove unnecessary packages
apt-get remove --purge <package>

# Disable unnecessary services
systemctl disable <service>
```

```
systemctl stop <service>

# Check listening ports
netstat -tlnp
ss -tlnp

# Check running processes
ps aux
```

1.34.2 Secure SSH

```
# /etc/ssh/sshd_config
PermitRootLogin no
PasswordAuthentication no
PubkeyAuthentication yes
PermitEmptyPasswords no
X11Forwarding no
MaxAuthTries 3
```

1.34.3 File System Security

```
# Set proper permissions
chmod 600 /etc/kubernetes/admin.conf
chmod 600 /etc/kubernetes/pki/*.key

# Check file permissions
ls -la /etc/kubernetes/
ls -la /etc/kubernetes/pki/
```

1.35 Limit Node Access

1.35.1 Restrict SSH Access

```
# Allow only specific users
# /etc/ssh/sshd_config
AllowUsers admin

# Use SSH keys only
PasswordAuthentication no
```

1.35.2 Use Network Policies for Node Access

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: restrict-node-access
  namespace: kube-system
spec:
  podSelector: {}
```

```
  policyTypes:
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 10.0.0.0/8
```

1.36 Linux Capabilities

1.36.1 Drop All Capabilities

```
apiVersion: v1
kind: Pod
metadata:
  name: drop-caps-pod
spec:
  containers:
  - name: nginx
    image: nginx
    securityContext:
      capabilities:
        drop:
        - ALL
```

1.36.2 Add Specific Capabilities

```
apiVersion: v1
kind: Pod
metadata:
  name: net-admin-pod
spec:
  containers:
  - name: nginx
    image: nginx
    securityContext:
      capabilities:
        drop:
        - ALL
        add:
        - NET_BIND_SERVICE
```

1.36.3 Common Capabilities

Capability	Description
`NET_BIND_SERVICE`	Bind to ports < 1024
`NET_ADMIN`	Network administration
`SYS_ADMIN`	System administration (dangerous)
`SYS_PTRACE`	Trace processes

Capability	Description
CHOWN	Change file ownership

1.37 Kernel Hardening

1.37.1 sysctl Settings

```
# /etc/sysctl.d/99-kubernetes-cis.conf

# Disable IP forwarding (if not needed)
net.ipv4.ip_forward = 0

# Disable source routing
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0

# Enable SYN cookies
net.ipv4.tcp_syncookies = 1

# Disable ICMP redirects
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0

# Apply settings
sysctl --system
```

1.38 IAM Roles (Cloud)

1.38.1 AWS IAM for Nodes

```
# Restrict node IAM role permissions
# Only allow necessary EC2 and ECR actions
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage"
      ],
      "Resource": "*"
    }
  ]
}
```

1.38.2 Pod Identity (IRSA)

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-sa
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::123456789:role/my-
        role
```

1.39 Key Concepts to Remember

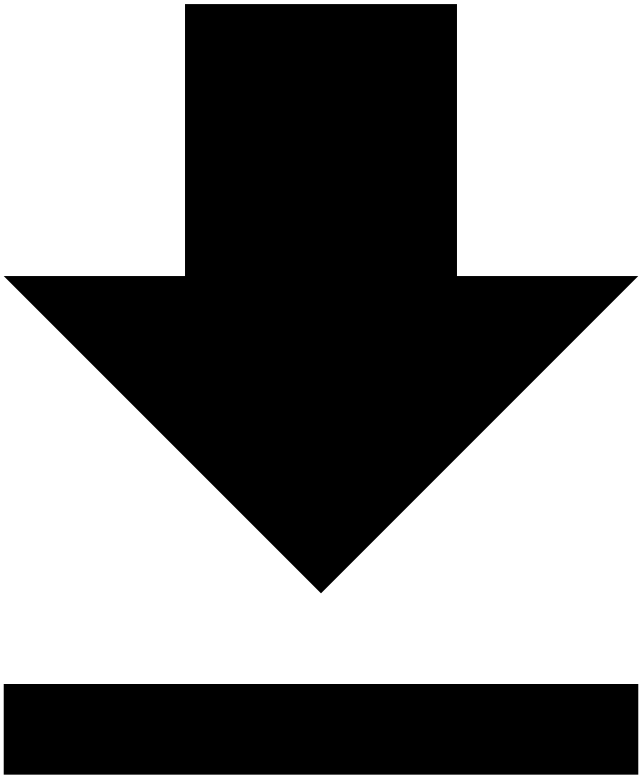
1. **AppArmor** - Linux security module for access control
2. **seccomp** - System call filtering
3. **Capabilities** - Drop ALL, add only needed
4. **Minimize footprint** - Remove unnecessary packages/services
5. **Kernel hardening** - sysctl security settings

1.40 Practice Questions

1. How do you apply an AppArmor profile to a container?
2. What is the difference between seccomp RuntimeDefault and Localhost?
3. How do you drop all capabilities from a container?
4. Where are seccomp profiles stored on the node?
5. How do you check AppArmor status on a node?

← Previous: Cluster Hardening | Back to CKS Overview | Next: Minimize Microservice Vulnerabilities →

1.41 Minimize Microservice Vulnerabilities



Download PDF Version

This domain covers securing containerized applications and implementing Pod security.

1.42 Pod Security Standards

1.42.1 Pod Security Admission

```yaml
# Apply to namespace
apiVersion: v1
kind: Namespace
metadata:
  name: secure-ns
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
```

1.42.2 Security Levels

Level	Description
`privileged`	Unrestricted, allows all
`baseline`	Minimally restrictive, prevents known privilege escalations
`restricted`	Heavily restricted, follows security best practices

1.42.3 Restricted Pod Example

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: restricted-pod
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: app
    image: nginx
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      capabilities:
        drop:
        - ALL
    volumeMounts:
    - name: tmp
      mountPath: /tmp
    - name: cache
      mountPath: /var/cache/nginx
    - name: run
      mountPath: /var/run
  volumes:
  - name: tmp
    emptyDir: {}
  - name: cache
    emptyDir: {}
  - name: run
    emptyDir: {}
```

1.43 Security Context

1.43.1 Pod-level Security Context

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: app
    image: nginx
```

1.43.2 Container-level Security Context

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: container-secure
spec:
  containers:
  - name: app
    image: nginx
    securityContext:
      runAsUser: 1000
      runAsNonRoot: true
      readOnlyRootFilesystem: true
      allowPrivilegeEscalation: false
      capabilities:
        drop:
        - ALL
        add:
        - NET_BIND_SERVICE
```

1.43.3 Security Context Fields

Field	Level	Description
`runAsUser`	Pod/Container	UID to run as
`runAsGroup`	Pod/Container	GID to run as
`runAsNonRoot`	Pod/Container	Must run as non-root
`fsGroup`	Pod	Group for volumes

Field	Level	Description
`readOnlyRootFilesystem`	Container	Read-only root FS
`allowPrivilegeEscalation`	Container	Prevent privilege escalation
`capabilities`	Container	Linux capabilities
`seccompProfile`	Pod/Container	seccomp profile

1.44 OPA Gatekeeper

1.44.1 Install Gatekeeper

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-
        agent/gatekeeper/master/deploy/gatekeeper.yaml
```

1.44.2 Constraint Template

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        openAPIV3Schema:
          type: object
          properties:
            labels:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels

        violation[{"msg": msg}] {
          provided := {label |
         input.review.object.metadata.labels[label]}
          required := {label | label :=
         input.parameters.labels[_]}
          missing := required - provided
          count(missing) > 0
          msg := sprintf("Missing required labels: %v", [missing])
        }
```

1.44.3 Constraint

```yaml
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
  name: require-team-label
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  parameters:
    labels: ["team"]
```

1.44.4 Deny Privileged Containers

```yaml
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sdenyprivileged
spec:
  crd:
    spec:
      names:
        kind: K8sDenyPrivileged
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8sdenyprivileged

        violation[{"msg": msg}] {
          container := input.review.object.spec.containers[_]
          container.securityContext.privileged == true
          msg := sprintf("Privileged containers are not allowed:
        %v", [container.name])
        }
```

1.45 Secrets Management

1.45.1 Create Secrets

```bash
# From literal
kubectl create secret generic db-secret \
  --from-literal=username=admin \
  --from-literal=password=secret123

# From file
kubectl create secret generic tls-secret \
```

```
  --from-file=tls.crt \
  --from-file=tls.key
```

1.45.2 Use Secrets as Environment Variables

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
  - name: app
    image: nginx
    env:
    - name: DB_PASSWORD
      valueFrom:
        secretKeyRef:
          name: db-secret
          key: password
```

1.45.3 Use Secrets as Volumes

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-vol-pod
spec:
  containers:
  - name: app
    image: nginx
    volumeMounts:
    - name: secret-volume
      mountPath: /etc/secrets
      readOnly: true
  volumes:
  - name: secret-volume
    secret:
      secretName: db-secret
      defaultMode: 0400
```

1.45.4 Encrypt Secrets at Rest

```yaml
# /etc/kubernetes/encryption-config.yaml
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
```

```
            - name: key1
              secret: <base64-encoded-32-byte-key>
      - identity: {}
```

1.46 Container Runtime Sandboxing

1.46.1 gVisor (runsc)

```
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: gvisor
handler: runsc
---
apiVersion: v1
kind: Pod
metadata:
  name: sandboxed-pod
spec:
  runtimeClassName: gvisor
  containers:
  - name: app
    image: nginx
```

1.46.2 Kata Containers

```
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: kata
handler: kata-qemu
---
apiVersion: v1
kind: Pod
metadata:
  name: kata-pod
spec:
  runtimeClassName: kata
  containers:
  - name: app
    image: nginx
```

1.47 mTLS with Service Mesh

1.47.1 Istio Strict mTLS

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
```

```yaml
metadata:
  name: default
  namespace: istio-system
spec:
  mtls:
    mode: STRICT
```

1.48 Key Concepts to Remember

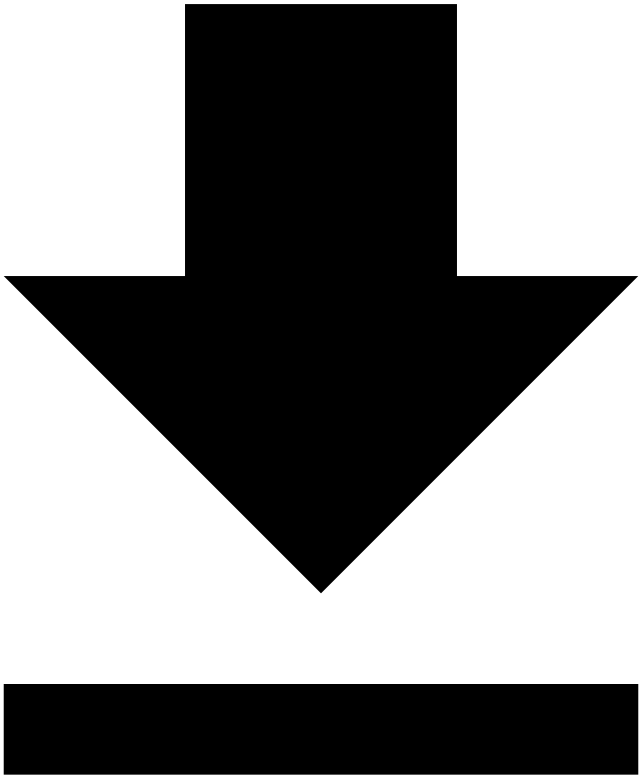
1. **Pod Security Standards** - privileged, baseline, restricted
2. **Security Context** - runAsNonRoot, readOnlyRootFilesystem, capabilities
3. **OPA Gatekeeper** - Policy enforcement
4. **Secrets** - Encrypt at rest, mount as volumes with restrictive permissions
5. **Runtime Sandboxing** - gVisor, Kata Containers

1.49 Practice Questions

1. How do you enforce the restricted Pod Security Standard on a namespace?
2. What security context fields prevent privilege escalation?
3. How do you create an OPA Gatekeeper constraint?
4. What is the recommended way to mount secrets in pods?
5. How do you use a sandboxed runtime for a pod?

← Previous: System Hardening | Back to CKS Overview | Next: Supply Chain Security →

1.50 Supply Chain Security



Download PDF Version

This domain covers securing the software supply chain including image security, static analysis, and image signing.

1.51 Image Vulnerability Scanning

1.51.1 Trivy

```
# Scan image
trivy image nginx:1.21

# Scan with severity filter
trivy image --severity HIGH,CRITICAL nginx:1.21

# Scan and fail on vulnerabilities
trivy image --exit-code 1 --severity CRITICAL nginx:1.21

# Scan local image
```

```
trivy image --input image.tar

# Output as JSON
trivy image -f json -o results.json nginx:1.21

# Scan filesystem
trivy fs /path/to/project

# Scan Kubernetes resources
trivy k8s --report summary cluster
```

1.51.2 Trivy Output Example

```
nginx:1.21 (debian 11.2)
========================
Total: 125 (UNKNOWN: 0, LOW: 85, MEDIUM: 26, HIGH: 12, CRITICAL:
2)

┌──────────────────┬────────────────┬──────────┬──────────────────
│     Library      │ Vulnerability  │ Severity │  Installed
Version│
├──────────────────┼────────────────┼──────────┼──────────────────
│ libssl1.1        │ CVE-2022-0778  │ HIGH     │ 1.1.1k-1+deb11u1
│
│ openssl          │ CVE-2022-0778  │ HIGH     │ 1.1.1k-1+deb11u1
│
└──────────────────┴────────────────┴──────────┴──────────────────
```

1.51.3 Integrate Trivy in CI/CD

```
# GitHub Actions example
- name: Scan image
  uses: aquasecurity/trivy-action@master
  with:
    image-ref: 'myimage:${{ github.sha }}'
    format: 'table'
    exit-code: '1'
    severity: 'CRITICAL,HIGH'
```

1.52 Image Signing and Verification

1.52.1 Cosign

```
# Generate key pair
cosign generate-key-pair

# Sign image
cosign sign --key cosign.key myregistry/myimage:v1

# Verify image
```

```
cosign verify --key cosign.pub myregistry/myimage:v1

# Sign with keyless (OIDC)
cosign sign myregistry/myimage:v1

# Verify keyless signature
cosign verify myregistry/myimage:v1
```

1.52.2 Image Policy Webhook

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: image-policy-webhook
webhooks:
- name: image-policy.example.com
  rules:
  - apiGroups: [""]
    apiVersions: ["v1"]
    operations: ["CREATE", "UPDATE"]
    resources: ["pods"]
  clientConfig:
    service:
      name: image-policy-webhook
      namespace: kube-system
      path: "/validate"
    caBundle: <base64-encoded-ca-cert>
  admissionReviewVersions: ["v1"]
  sideEffects: None
```

1.53 Minimize Base Image Footprint

1.53.1 Use Minimal Base Images

```
# Bad: Full OS image
FROM ubuntu:22.04

# Better: Slim image
FROM python:3.11-slim

# Best: Distroless
FROM gcr.io/distroless/python3

# Best: Scratch (for static binaries)
FROM scratch
```

1.53.2 Multi-stage Builds

```
# Build stage
FROM golang:1.21 AS builder
```

```
WORKDIR /app
COPY . .
RUN CGO_ENABLED=0 GOOS=linux go build -o myapp

# Runtime stage
FROM gcr.io/distroless/static:nonroot
COPY --from=builder /app/myapp /myapp
USER nonroot:nonroot
ENTRYPOINT ["/myapp"]
```

1.53.3 Image Best Practices

Practice	Description
Use specific tags	Avoid `latest`, use version tags
Use minimal base	distroless, alpine, scratch
Multi-stage builds	Reduce final image size
Run as non-root	Use USER instruction
No secrets in image	Use runtime secrets
Scan regularly	Integrate scanning in CI/CD

1.54 Secure Image Registries

1.54.1 Private Registry Authentication

```
# Create registry secret
kubectl create secret docker-registry regcred \
  --docker-server=myregistry.io \
  --docker-username=user \
  --docker-password=pass \
  --docker-email=user@example.com
```

1.54.2 Use ImagePullSecrets

```
apiVersion: v1
kind: Pod
metadata:
  name: private-image-pod
spec:
  containers:
  - name: app
    image: myregistry.io/myimage:v1
  imagePullSecrets:
  - name: regcred
```

1.54.3 ServiceAccount with ImagePullSecrets

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-sa
imagePullSecrets:
- name: regcred
```

1.55 Allowlist Registries

1.55.1 OPA Gatekeeper Policy

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sallowedrepos
spec:
  crd:
    spec:
      names:
        kind: K8sAllowedRepos
      validation:
        openAPIV3Schema:
          type: object
          properties:
            repos:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8sallowedrepos

        violation[{"msg": msg}] {
          container := input.review.object.spec.containers[_]
          not startswith(container.image,
        input.parameters.repos[_])
          msg := sprintf("Container image %v is not from an
        allowed registry", [container.image])
        }
---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: allowed-repos
spec:
  match:
    kinds:
      - apiGroups: [""]
```

```
        kinds: ["Pod"]
  parameters:
    repos:
      - "gcr.io/my-project/"
      - "myregistry.io/"
```

1.56 Static Analysis

1.56.1 Kubesec

```
# Scan Kubernetes manifest
kubesec scan pod.yaml

# Scan with HTTP API
curl -sSX POST --data-binary @pod.yaml https://v2.kubesec.io/scan
```

1.56.2 Kubesec Output

```
{
  "score": 0,
  "scoring": {
    "critical": [
      {
        "id": "CapSysAdmin",
        "selector":
        "containers[] .securityContext .capabilities .add ==
        SYS_ADMIN",
        "reason": "CAP_SYS_ADMIN is the most privileged
        capability"
      }
    ],
    "advise": [
      {
        "id": "ApparmorAny",
        "selector": ".metadata .annotations .
        \"container.apparmor.security.beta.kubernetes.io/nginx\"",
        "reason": "Well defined AppArmor policies may provide
        greater protection"
      }
    ]
  }
}
```

1.56.3 Conftest (OPA for Config Files)

```
# Test Kubernetes manifests
conftest test deployment.yaml
```

```
# With custom policy
conftest test --policy ./policy deployment.yaml
```

1.56.4 Example Conftest Policy

```
# policy/deployment.rego
package main

deny[msg] {
  input.kind == "Deployment"
  not input.spec.template.spec.securityContext.runAsNonRoot
  msg := "Containers must run as non-root"
}

deny[msg] {
  input.kind == "Deployment"
  container := input.spec.template.spec.containers[_]
  not container.resources.limits
  msg := sprintf("Container %v must have resource limits",
[container.name])
}
```

1.57 Dockerfile Security

1.57.1 Dockerfile Best Practices

```
# Use specific version
FROM python:3.11-slim-bookworm

# Create non-root user
RUN groupadd -r appgroup && useradd -r -g appgroup appuser

# Set working directory
WORKDIR /app

# Copy only necessary files
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY --chown=appuser:appgroup . .

# Switch to non-root user
USER appuser

# Use ENTRYPOINT for main command
ENTRYPOINT ["python", "app.py"]
```

1.57.2 Hadolint (Dockerfile Linter)

```
# Lint Dockerfile
hadolint Dockerfile

# Ignore specific rules
hadolint --ignore DL3008 Dockerfile
```

1.58 Key Concepts to Remember

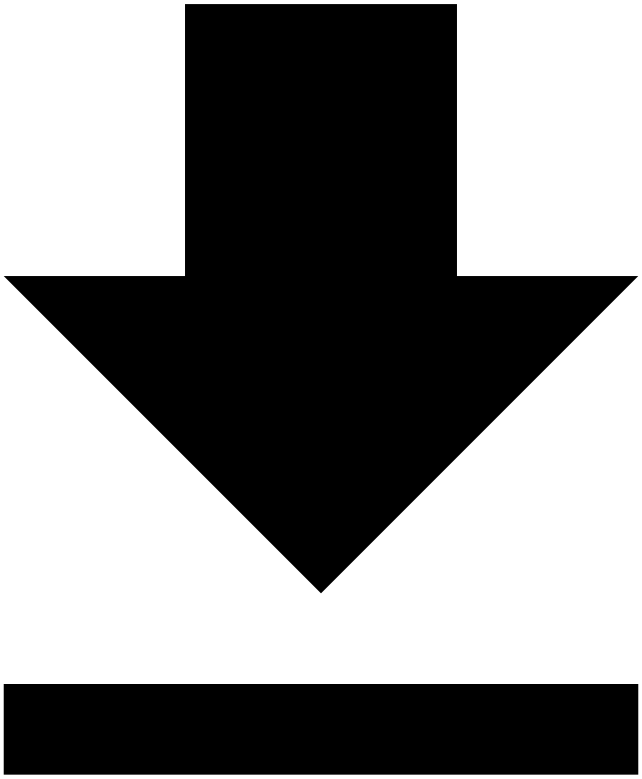
1. **Trivy** - Image vulnerability scanning
2. **Cosign** - Image signing and verification
3. **Minimal images** - distroless, alpine, scratch
4. **Multi-stage builds** - Reduce image size
5. **Registry allowlisting** - OPA Gatekeeper policies

1.59 Practice Questions

1. How do you scan an image for HIGH and CRITICAL vulnerabilities with Trivy?
2. How do you sign an image with Cosign?
3. What is the benefit of using distroless base images?
4. How do you create an OPA policy to allowlist registries?
5. What tool lints Dockerfiles for security issues?

← Previous: Minimize Microservice Vulnerabilities | Back to CKS Overview | Next: Monitoring, Logging and Runtime Security →

1.60 Monitoring, Logging and Runtime Security



Download PDF Version

This domain covers runtime security monitoring, audit logging, behavioral analysis, and threat detection in Kubernetes environments.

1.61 Overview

Runtime security focuses on detecting and preventing threats while workloads are running. Key areas include: - Behavioral analysis and anomaly detection - Audit logging for compliance and forensics - Container immutability enforcement - System call monitoring

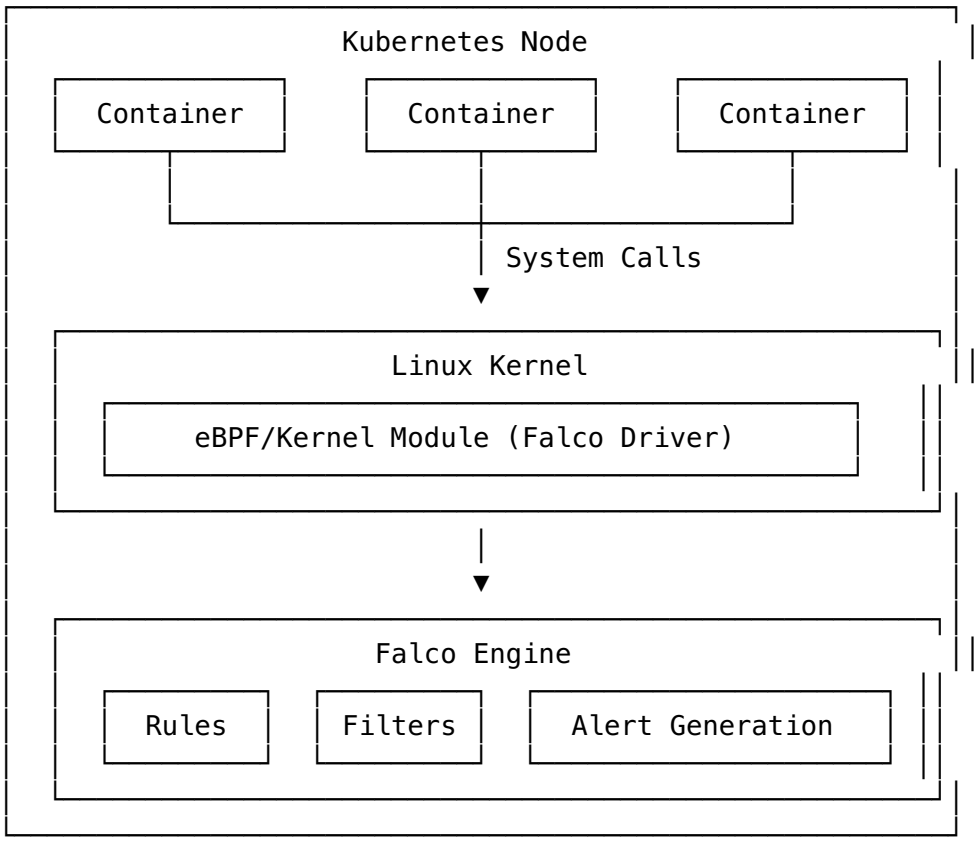
1.62 Falco - Runtime Threat Detection

Falco is a CNCF project that provides runtime security by monitoring system calls and detecting anomalous behavior.

1.62.1 Install Falco

```
# Using Helm
helm repo add falcosecurity https://falcosecurity.github.io/charts
helm repo update
helm install falco falcosecurity/falco \
  --namespace falco \
  --create-namespace \
  --set falcosidekick.enabled=true

# Verify installation
kubectl get pods -n falco
kubectl logs -n falco -l app.kubernetes.io/name=falco
```

1.62.2 Falco Architecture



1.62.3 Falco Rules Structure

```
# Rule components
- rule: <rule_name>           # Unique rule identifier
  desc: <description>         # Human-readable description
  condition: <filter>         # Sysdig filter expression
  output: <message>           # Alert message with fields
  priority: <level>           # EMERGENCY, ALERT, CRITICAL,
        ERROR, WARNING, NOTICE, INFO, DEBUG
```

```
  tags: [<tag1>, <tag2>]      # Optional categorization
  enabled: true/false         # Enable/disable rule
```

1.62.4 Common Falco Rules

```
# Detect shell spawning in containers
- rule: Terminal shell in container
  desc: A shell was used as the entrypoint/exec point into a
        container
  condition: >
    spawned_process and container
    and shell_procs
    and proc.tty != 0
    and container_entrypoint
  output: >
    Shell spawned in container
    (user=%user.name user_loginuid=%user.loginuid %container.info
    shell=%proc.name parent=%proc.pname cmdline=%proc.cmdline
    terminal=%proc.tty container_id=%container.id
        image=%container.image.repository)
  priority: WARNING
  tags: [container, shell, mitre_execution]

# Detect writes to sensitive directories
- rule: Write below etc
  desc: An attempt to write to /etc directory
  condition: >
    write and container
    and fd.directory = /etc
  output: >
    File written to /etc in container
    (user=%user.name command=%proc.cmdline
    file=%fd.name container=%container.name
        image=%container.image.repository)
  priority: ERROR
  tags: [filesystem, mitre_persistence]

# Detect package management in containers
- rule: Package management in container
  desc: Package management tool executed in container
  condition: >
    spawned_process and container
    and package_mgmt_procs
  output: >
    Package management process launched in container
    (user=%user.name command=%proc.cmdline
        container=%container.name)
  priority: ERROR
  tags: [process, software_mgmt]

# Detect sensitive file access
- rule: Read sensitive file
  desc: Sensitive file was read
```

```
  condition: >
    open_read and container
    and (fd.name startswith /etc/shadow or
         fd.name startswith /etc/passwd or
         fd.name startswith /etc/sudoers)
  output: >
    Sensitive file opened for reading
    (user=%user.name file=%fd.name container=%container.name)
  priority: WARNING
  tags: [filesystem, mitre_credential_access]

# Detect outbound connections to unusual ports
- rule: Unexpected outbound connection
  desc: Outbound connection to non-standard port
  condition: >
    outbound and container
    and not (fd.sport in (80, 443, 53, 8080, 8443))
  output: >
    Unexpected outbound connection
    (command=%proc.cmdline connection=%fd.name
         container=%container.name)
  priority: NOTICE
  tags: [network, mitre_exfiltration]
```

1.62.5 Custom Falco Rules

```
# /etc/falco/rules.d/custom-rules.yaml
- rule: Crypto mining detected
  desc: Detect crypto mining processes
  condition: >
    spawned_process and container
    and (proc.name in (xmrig, minerd, cpuminer, cgminer))
  output: >
    Crypto mining process detected
    (user=%user.name command=%proc.cmdline
         container=%container.name)
  priority: CRITICAL
  tags: [cryptomining, mitre_resource_hijacking]

- rule: Kubectl exec detected
  desc: Detect kubectl exec into containers
  condition: >
    spawned_process and container
    and proc.name = "runc"
    and proc.cmdline contains "exec"
  output: >
    Kubectl exec detected (container=%container.name
         cmdline=%proc.cmdline)
  priority: WARNING
  tags: [exec, mitre_execution]
```

1.62.6 Falco Commands

```
# Check Falco status
systemctl status falco

# View Falco logs
kubectl logs -n falco -l app.kubernetes.io/name=falco -f

# Test Falco rules
kubectl exec -it <pod> -- /bin/sh  # Should trigger shell alert

# Reload rules
kill -1 $(pidof falco)

# Check rule syntax
falco -V /etc/falco/rules.d/custom-rules.yaml
```

1.63 Kubernetes Audit Logging

Audit logging records all requests to the Kubernetes API server for security analysis and compliance.

1.63.1 Audit Stages

Stage	Description
RequestReceived	Event generated when request is received
ResponseStarted	Response headers sent, body not yet sent
ResponseComplete	Response body completed
Panic	Events generated when panic occurs

1.63.2 Audit Levels

Level	Description
None	Don’t log events
Metadata	Log request metadata only
Request	Log metadata and request body
RequestResponse	Log metadata, request, and response bodies

1.63.3 Comprehensive Audit Policy

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  # Don't log read-only endpoints
  - level: None
    nonResourceURLs:
```

```
    - /healthz*
    - /version
    - /swagger*
    - /readyz*
    - /livez*

# Don't log watch requests (too verbose)
- level: None
  verbs: ["watch"]
  resources:
    - group: ""
      resources: ["events"]

# Log secret access at Metadata level (don't log secret data)
- level: Metadata
  resources:
    - group: ""
      resources: ["secrets", "configmaps"]

# Log pod exec/attach at RequestResponse level
- level: RequestResponse
  resources:
    - group: ""
      resources: ["pods/exec", "pods/attach", "pods/
      portforward"]

# Log authentication events
- level: RequestResponse
  resources:
    - group: "authentication.k8s.io"
      resources: ["tokenreviews"]

# Log authorization events
- level: RequestResponse
  resources:
    - group: "authorization.k8s.io"
      resources: ["subjectaccessreviews",
      "selfsubjectaccessreviews"]

# Log RBAC changes
- level: RequestResponse
  resources:
    - group: "rbac.authorization.k8s.io"
      resources: ["roles", "rolebindings", "clusterroles",
      "clusterrolebindings"]

# Log node and namespace changes
- level: RequestResponse
  resources:
    - group: ""
      resources: ["nodes", "namespaces"]

# Log service account token creation
```

```
  - level: RequestResponse
    resources:
      - group: ""
        resources: ["serviceaccounts/token"]

  # Default: log at Metadata level
  - level: Metadata
    omitStages:
      - RequestReceived
```

1.63.4 Enable Audit Logging on API Server

```
# /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    # Audit configuration
    - --audit-policy-file=/etc/kubernetes/audit/audit-policy.yaml
    - --audit-log-path=/var/log/kubernetes/audit/audit.log
    - --audit-log-maxage=30
    - --audit-log-maxbackup=10
    - --audit-log-maxsize=100
    # Optional: webhook backend
    # - --audit-webhook-config-file=/etc/kubernetes/audit/webhook-
        config.yaml
    volumeMounts:
    - mountPath: /etc/kubernetes/audit
      name: audit-config
      readOnly: true
    - mountPath: /var/log/kubernetes/audit
      name: audit-log
  volumes:
  - hostPath:
      path: /etc/kubernetes/audit
      type: DirectoryOrCreate
    name: audit-config
  - hostPath:
      path: /var/log/kubernetes/audit
      type: DirectoryOrCreate
    name: audit-log
```

1.63.5 Analyze Audit Logs

```
# View all audit events
cat /var/log/kubernetes/audit/audit.log | jq .
```

```
# Find delete operations
cat /var/log/kubernetes/audit/audit.log | jq
        'select(.verb=="delete")'

# Find secret access
cat /var/log/kubernetes/audit/audit.log | jq
        'select(.objectRef.resource=="secrets")'

# Find failed requests
cat /var/log/kubernetes/audit/audit.log | jq
        'select(.responseStatus.code >= 400)'

# Find requests by user
cat /var/log/kubernetes/audit/audit.log | jq
        'select(.user.username=="system:admin")'

# Find pod exec events
cat /var/log/kubernetes/audit/audit.log | jq
        'select(.objectRef.subresource=="exec")'

# Find RBAC changes
cat /var/log/kubernetes/audit/audit.log | jq
        'select(.objectRef.apiGroup=="rbac.authorization.k8s.io")'

# Count events by verb
cat /var/log/kubernetes/audit/audit.log | jq -r '.verb' | sort |
        uniq -c | sort -rn
```

1.63.6 Audit Log Entry Structure

```
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "level": "RequestResponse",
  "auditID": "unique-id",
  "stage": "ResponseComplete",
  "requestURI": "/api/v1/namespaces/default/pods",
  "verb": "create",
  "user": {
    "username": "admin",
    "groups": ["system:masters"]
  },
  "sourceIPs": ["192.168.1.100"],
  "objectRef": {
    "resource": "pods",
    "namespace": "default",
    "name": "nginx",
    "apiVersion": "v1"
  },
  "responseStatus": {
    "code": 201
  },
  "requestObject": { },
```

```
  "responseObject": { },
  "requestReceivedTimestamp": "2024-01-15T10:30:00.000000Z",
  "stageTimestamp": "2024-01-15T10:30:00.100000Z"
}
```

1.64 Container Immutability

Immutable containers prevent runtime modifications, reducing attack surface.

1.64.1 Read-Only Root Filesystem

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: immutable-pod
spec:
  containers:
  - name: app
    image: nginx:1.21
    securityContext:
      readOnlyRootFilesystem: true
      allowPrivilegeEscalation: false
      runAsNonRoot: true
      runAsUser: 1000
      capabilities:
        drop:
          - ALL
    volumeMounts:
    # Mount writable directories as needed
    - name: tmp
      mountPath: /tmp
    - name: var-run
      mountPath: /var/run
    - name: var-cache-nginx
      mountPath: /var/cache/nginx
  volumes:
  - name: tmp
    emptyDir: {}
  - name: var-run
    emptyDir: {}
  - name: var-cache-nginx
    emptyDir: {}
```

1.64.2 Enforce Immutability with Pod Security Standards

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: production
  labels:
```

```
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
```

1.65 Sysdig and Other Tools

1.65.1 Sysdig for Troubleshooting

```
# Capture system calls
sysdig -c topprocs_cpu

# Monitor container activity
sysdig -c topcontainers_cpu

# Watch file opens in container
sysdig -c echo_fds container.name=nginx

# Monitor network connections
sysdig -c topconns

# Capture to file for analysis
sysdig -w capture.scap
sysdig -r capture.scap
```

1.66 Key Concepts Summary

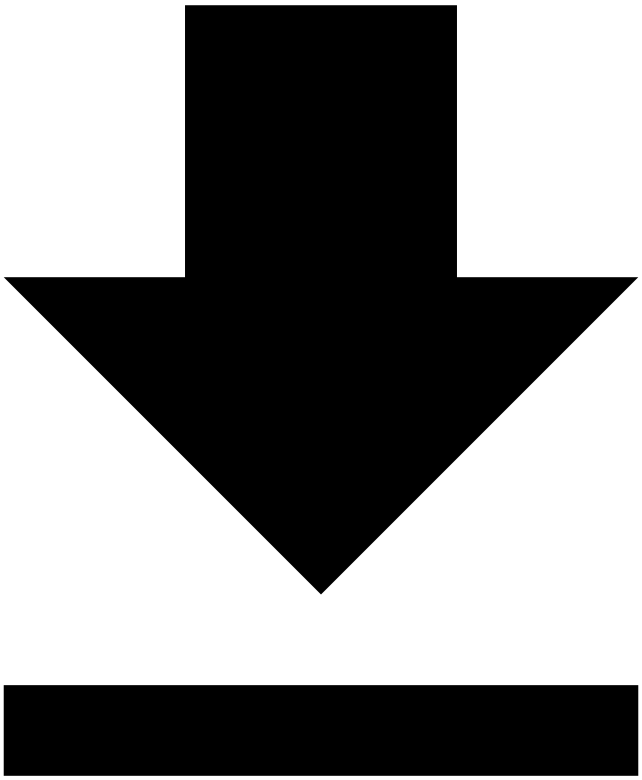
Concept	Description	Tool/Feature
Runtime Detection	Monitor running containers for threats	Falco
Audit Logging	Record API server activity	Kubernetes Audit
Immutability	Prevent runtime modifications	readOnlyRootFilesystem
Behavioral Analysis	Detect anomalous behavior	Falco rules
System Call Monitoring	Track kernel-level activity	eBPF, Sysdig

1.67 Practice Exercises

1. Install Falco and trigger a shell alert
2. Create a custom Falco rule to detect file writes to /tmp
3. Configure audit logging to capture secret access
4. Analyze audit logs to find failed authentication attempts
5. Create an immutable pod with read-only filesystem

← Previous: Supply Chain Security | Back to CKS Overview | Next: Sample Questions →

1.68 Sample Practice Questions



Download PDF Version

> **Disclaimer**: These are sample practice questions created for study purposes only. They are NOT actual exam questions and are designed to help you test your understanding of CKS concepts.

1.69 Practice Resources

- **Killercoda CKS Scenarios** ⭐ Free hands-on practice
- **killer.sh CKS Simulator** - Included with exam registration

---

1.70 Section 1: Cluster Setup (10%)

1.70.1 Question 1.1 - Network Policy

Create a NetworkPolicy named `deny-all` in namespace `secure` that denies all ingress and egress traffic.

Show Solution

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
  namespace: secure
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

1.70.2 Question 1.2 - CIS Benchmark

Run kube-bench to check the master node against CIS benchmarks.

Show Solution

```
kube-bench run --targets=master
```

1.71 Section 2: Cluster Hardening (15%)

1.71.1 Question 2.1 - RBAC

Create a Role named `pod-reader` that allows get, list, watch on pods. Bind it to user `jane`.

Show Solution

```
kubectl create role pod-reader --verb=get,list,watch --
        resource=pods
kubectl create rolebinding read-pods --role=pod-reader --user=jane
```

1.71.2 Question 2.2 - ServiceAccount

Create a Pod that doesn't auto-mount the ServiceAccount token.

Show Solution

```yaml
apiVersion: v1
kind: Pod
```

```yaml
metadata:
  name: no-token-pod
spec:
  automountServiceAccountToken: false
  containers:
  - name: nginx
    image: nginx
```

1.72 Section 3: System Hardening (15%)

1.72.1 Question 3.1 - AppArmor

Create a Pod with AppArmor profile `k8s-deny-write` applied.

Show Solution

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: apparmor-pod
  annotations:
    container.apparmor.security.beta.kubernetes.io/nginx:
        localhost/k8s-deny-write
spec:
  containers:
  - name: nginx
    image: nginx
```

1.72.2 Question 3.2 - seccomp

Create a Pod using RuntimeDefault seccomp profile.

Show Solution

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: seccomp-pod
spec:
  securityContext:
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: nginx
    image: nginx
```

1.73 Section 4: Minimize Microservice Vulnerabilities (20%)

1.73.1 Question 4.1 - Security Context

Create a Pod that runs as non-root with read-only filesystem and drops all capabilities.

Show Solution

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
  containers:
  - name: nginx
    image: nginx
    securityContext:
      readOnlyRootFilesystem: true
      allowPrivilegeEscalation: false
      capabilities:
        drop:
        - ALL
```

1.73.2 Question 4.2 - Pod Security Standards

Apply the restricted Pod Security Standard to namespace `production`.

Show Solution

```
kubectl label namespace production \
  pod-security.kubernetes.io/enforce=restricted \
  pod-security.kubernetes.io/audit=restricted \
  pod-security.kubernetes.io/warn=restricted
```

1.74 Section 5: Supply Chain Security (20%)

1.74.1 Question 5.1 - Image Scanning

Scan image `nginx:1.21` for HIGH and CRITICAL vulnerabilities using Trivy.

Show Solution

```
trivy image --severity HIGH,CRITICAL nginx:1.21
```

1.74.2 Question 5.2 - Allowed Registries

Create an OPA Gatekeeper constraint to only allow images from `gcr.io/myproject/`.

Show Solution

```yaml
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: allowed-repos
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  parameters:
    repos:
      - "gcr.io/myproject/"
```

1.75 Section 6: Monitoring, Logging and Runtime Security (20%)

1.75.1 Question 6.1 - Audit Logging

Configure API server to log all secret access at Metadata level.

Show Solution

```yaml
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  - level: Metadata
    resources:
      - group: ""
        resources: ["secrets"]
```

Add to kube-apiserver:

```
--audit-policy-file=/etc/kubernetes/audit-policy.yaml
--audit-log-path=/var/log/kubernetes/audit/audit.log
```

1.75.2 Question 6.2 - Falco

Write a Falco rule to detect shell spawning in containers.

Show Solution

```yaml
- rule: Shell in container
  desc: Detect shell spawned in container
```

```
  condition: spawned_process and container and shell_procs
  output: Shell spawned (user=%user.name
        container=%container.name)
  priority: WARNING
```

1.75.3 Question 6.3 - Container Immutability

Create a Pod with a read-only root filesystem that can still write to /tmp.

Show Solution

```
apiVersion: v1
kind: Pod
metadata:
  name: immutable-pod
spec:
  containers:
  - name: app
    image: nginx
    securityContext:
      readOnlyRootFilesystem: true
    volumeMounts:
    - name: tmp
      mountPath: /tmp
  volumes:
  - name: tmp
    emptyDir: {}
```

1.75.4 Question 6.4 - Analyze Audit Logs

Find all secret access events in the audit log.

Show Solution

```
cat /var/log/kubernetes/audit/audit.log | jq
        'select(.objectRef.resource=="secrets")'

# Or find specific operations on secrets
cat /var/log/kubernetes/audit/audit.log | jq
        'select(.objectRef.resource=="secrets" and .verb=="get")'
```

1.76 Section 7: Additional Practice Questions

1.76.1 Question 7.1 - Ingress TLS

Create an Ingress with TLS termination using a secret named `tls-secret`.

Show Solution

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: secure-ingress
spec:
  tls:
  - hosts:
    - secure.example.com
    secretName: tls-secret
  rules:
  - host: secure.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-service
            port:
              number: 80
```

1.76.2 Question 7.2 - RuntimeClass with gVisor

Create a RuntimeClass for gVisor and a Pod that uses it.

Show Solution

```yaml
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: gvisor
handler: runsc
---
apiVersion: v1
kind: Pod
metadata:
  name: sandboxed-pod
spec:
  runtimeClassName: gvisor
  containers:
  - name: app
    image: nginx
```

1.76.3 Question 7.3 - Encrypt etcd Data

Configure encryption at rest for secrets in etcd.

Show Solution

```yaml
# /etc/kubernetes/enc/enc.yaml
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
```

```yaml
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: <base64-encoded-32-byte-key>
      - identity: {}
```

Add to kube-apiserver:

```
--encryption-provider-config=/etc/kubernetes/enc/enc.yaml
```

1.76.4 Question 7.4 - Verify Image Signature

Use cosign to verify an image signature.

Show Solution

```
# Verify image signature
cosign verify --key cosign.pub gcr.io/myproject/myimage:v1

# Generate a key pair
cosign generate-key-pair

# Sign an image
cosign sign --key cosign.key gcr.io/myproject/myimage:v1
```

1.76.5 Question 7.5 - Restrict Syscalls with Seccomp

Create a Pod that blocks the `chmod` syscall using a custom seccomp profile.

Show Solution

```
// /var/lib/kubelet/seccomp/profiles/block-chmod.json
{
  "defaultAction": "SCMP_ACT_ALLOW",
  "syscalls": [
    {
      "names": ["chmod", "fchmod", "fchmodat"],
      "action": "SCMP_ACT_ERRNO"
    }
  ]
}
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: seccomp-pod
spec:
  securityContext:
    seccompProfile:
```

```
      type: Localhost
      localhostProfile: profiles/block-chmod.json
  containers:
  - name: app
    image: nginx
```

1.76.6 Question 7.6 - Network Policy for Database

Create a NetworkPolicy that only allows pods with label `app=backend` to access pods with label `app=database` on port 5432.

Show Solution

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: db-policy
spec:
  podSelector:
    matchLabels:
      app: database
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: backend
    ports:
    - protocol: TCP
      port: 5432
```

1.76.7 Question 7.7 - Admission Controller Webhook

Which admission controllers should be enabled for security?

Show Solution

Essential security admission controllers: - `NodeRestriction` - Limits kubelet permissions - `PodSecurity` - Enforces Pod Security Standards - `AlwaysPullImages` - Forces image pull on every pod start - `DenyServiceExternalIPs` - Prevents external IP assignment

Enable in kube-apiserver:

```
--enable-admission-
plugins=NodeRestriction,PodSecurity,AlwaysPullImages
```

1.76.8 Question 7.8 - Investigate Compromised Pod

A pod is suspected of being compromised. What steps would you take?

Show Solution

```
# 1. Isolate the pod with NetworkPolicy
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: isolate-pod
spec:
  podSelector:
    matchLabels:
      app: compromised
  policyTypes:
  - Ingress
  - Egress
EOF

# 2. Check pod events and logs
kubectl describe pod <pod-name>
kubectl logs <pod-name>

# 3. Check running processes
kubectl exec <pod-name> -- ps aux

# 4. Check network connections
kubectl exec <pod-name> -- netstat -tulpn

# 5. Check audit logs
cat /var/log/kubernetes/audit/audit.log | jq
        'select(.objectRef.name=="<pod-name>")'

# 6. Check Falco alerts
kubectl logs -n falco -l app.kubernetes.io/name=falco | grep <pod-
        name>
```

1.77 Exam Tips

1. **Practice on Killercoda** - Free hands-on scenarios
2. **Use killer.sh** - Included with exam registration
3. **Know Trivy** for image scanning - `trivy image --severity HIGH,CRITICAL`
4. **Master Network Policies** - Default deny patterns, ingress/egress rules
5. **Understand RBAC** deeply - Roles, ClusterRoles, bindings
6. **Practice with Falco** rules - Rule syntax, common detections
7. **Know audit logging** configuration - Policy levels, API server flags
8. **Understand Pod Security Standards** - privileged, baseline, restricted
9. **Practice seccomp and AppArmor** - Profile application
10. **Know encryption at rest** - etcd encryption configuration

1.78 Quick Reference Commands

```
# Image scanning
trivy image nginx:latest

# Check CIS benchmarks
kube-bench run --targets=master

# Create secret
kubectl create secret generic my-secret --from-
        literal=password=secret

# Apply NetworkPolicy
kubectl apply -f networkpolicy.yaml

# Check RBAC
kubectl auth can-i create pods --as=jane

# View audit logs
cat /var/log/kubernetes/audit/audit.log | jq .

# Check Falco logs
kubectl logs -n falco -l app.kubernetes.io/name=falco
```

← Back to CKS Overview
