Certified Kubernetes Security Specialist (CKS) Study Guide

This is the study guide for the CKS course. Feel free to use this document to study for the CKS exam!

Cluster Setup

NetworkPolicies

Relevant Documentation

- Network Policies

Exam Tips

- Use a default deny network policy to block all network traffic to and/or from Pods in a namespace.
- Use an empty podSelector `{}` to make the policy apply to all Pods in the namespace.
- Use the policyTypes field to block incoming traffic (ingress), outgoing traffic (egress), or both.
- If any policy allows a particular type of traffic, it will be allowed. This means you can add targeted policies alongside a default deny policy to allow necessary traffic.
- Use a policy's podSelector to target the policy to specific Pods based upon their labels.
- Use Ingress/Egress rules to specify Pods, namespaces, or IP addresses to allow traffic to and/or from.
- Pay attention to the difference between one rule with multiple selectors, and multiple rules.

A default deny NetworkPolicy blocks traffic by default within a namespace. For example:

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

Combine a default deny policy with a targeted policy to allow specific traffic:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: nginx-ingress
  namespace: nptest
spec:
  podSelector:
    matchLabels:
      app: nginx
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          project: test
      podSelector:
        matchLabels:
          app: client
    ports:
    - port: 80
      protocol: TCP
```

CIS Benchmark

Relevant Documentation

- kube-bench
- CIS Kubernetes Benchmark

Exam Tips

- The CIS Kubernetes Benchmark is a set of standards and best practices for a secure Kubernetes cluster.
- Tools like kube-bench automatically check your cluster to see if it conforms to the CIS Kubernetes Benchmark standards and generate a report.
- The CIS Benchmark output includes remediation steps which you can use to address issues.
- kubeadm clusters use a kubelet config file located at `/var/lib/kubelet/config.yaml` on each node.
- In a kubeadm cluster, manifest files for control plane components like the API server can be found in `/etc/kubernetes/manifests` on the control plane server.

CIS Benchmark - A set of standards and best practices for secure cluster configuration.

kube-bench - A tool that runs automated tests to see how well a cluster conforms to CIS Benchmark standards.

TLS With Ingress

Relevant Documentation

- Ingress - TLS

Exam Tips

- You can implement TLS termination using an Ingress.
- Store TLS certificates using a Secret. Pass the Secret to the Ingress using `spec.tls[].secretName`.
- Bookmark the Ingress - TLS documentation.

Verifying Kubernetes Platform Binaries

Relevant Documentation

- Install kubectl binary with curl on Linux

Exam Tips

- When installing Kubernetes binaries manually, you can validate the binary files using the checksum to ensure the binaries have not been altered.

Download the checksum for kubectl. Provide the client version number from the pervious command (including the `v`):

```
curl -LO "https://dl.k8s.io/<kubectl client version>/bin/linux/amd64/kubectl.sha256"
```

Verify the kubectl binary using the checksum:

```
echo "$(<kubectl.sha256) /usr/bin/kubectl" | sha256sum --check
```

If the binary is valid, you should see output that says `OK`.

Cluster Hardening

Service Accounts and RBAC

Relevant Documentation

- Using RBAC Authorization
- Configure Service Accounts for Pods
- Managing Service Accounts

Exam Tips

- If a container is compromised, an attacker could use the ServiceAccount to access the Kubernetes API.
- Use Kubernetes RBAC to control what the ServiceAccount can do and limit permissions to only what is necessary.
- Examine existing RoleBindings and ClusterRoleBindings to determine what permissions a ServiceAccount has.
- Design your RBAC setup in such a way that service accounts don't have unnecessary permissions.
- You can bind multiple roles to an account. Use this to keep the Role separate, rather than overloading it with a lot of permissions.
- You can bind a ClusterRole with a RoleBinding to provide the necessary permissions only within the RoleBinding's namespace.

Sample ServiceAccount manifest:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: deployment-viewer-sa
  namespace: sa-permissions-test
automountServiceAccountToken: true
```

Sample Role manifest:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: deployment-viewer-role
  namespace: sa-permissions-test
rules:
- apiGroups: [""]
  resources: ["deployments"]
  verbs: ["get", "list"]
```

Sample RoleBinding manifest:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: deployment-viewer-rb
  namespace: sa-permissions-test
subjects:
- kind: ServiceAccount
  name: deployment-viewer-sa
  namespace: auth
roleRef:
  kind: Role
  name: deployment-viewer-role
  apiGroup: rbac.authorization.k8s.io
```

Set a ServiceAccount for a Pod with `spec.serviceAccountName`:

```
apiVersion: v1
kind: Pod
metadata:
  name: sa-pod
spec:
  serviceAccountName: my-sa
  containers:
  - name: nginx
    image: nginx
```

Restricting Access to the Kubernetes API

Relevant Documentation

- Controlling Access to the Kubernetes API

Exam Tips

- Use RBAC to control user permissions within the API. Make sure accounts do not have permissions they do not need.
- Limit network access to the API to prevent attackers from being able to communicatewith it.

Keeping k8s Updated

Relevant Documentation

- Kubernetes version and version skew support policy

Exam Tips

- Keep Kubernetes up to date to take advantage of the latest security patches.
- Since version 1.19, major releases receive approximately one year of patch support.

System Hardening

Host OS Security Concerns

Relevant Documentation

- Kubernetes API Reference - PodSpec
- Kubernetes API Reference - SecurityContext
- Security Context

Exam Tips

- Protect your hosts from attacks that might come from within containers.
- Beware of Pod settings like `hostPID`, `hostIPC`, and `hostNetwork`. Use them only when absolutely necessary!
- Beware of using privileged mode for containers with `securityContext.privileged`. Use this only when absolutely necessary!

IAM Roles

Relevant Documentation

- AWs Docs - Security Best Practices in IAM
- IAM Roles for Amazon EC2

Exam Tips

- If you are running Kubernetes in AWS, your container applications may be able to access IAM credentials.
- Avoid providing unnecessary permissions to IAM roles (principle of least privilege).
- If your app does not need IAM access, consider blocking access to IAM credentials via firewall, NetworkPolicy, etc.

Network-Level Security

Relevant Documentation

- Cluster Networking

Exam Tips

- By default, anyone who can access the cluster network can communicate with all Pods and Services in the cluster.
- When possible, limit access to the cluster network from outside.

AppArmor

Relevant Documentation

- AppArmor Documentation
- AppArmor in k8s

Exam Tips

- AppArmor is a Linux kernel security module that allows granular control over what individual programs can and cannot do.
- Load a profile in enforce mode (sometimes called "enforcing the profile") to actively prevent programs from doing anything the profile does not allow.
- Load a profile in complain mode to simply report on what the program is doing.
- Use the `apparmor_parser` command to load an AppArmor profile from a file. It will load the profile in enforcing mode by default.
- Use Pod annotations to apply an AppArmor profile to a container. For example: `container.apparmor.security.beta.kubernetes.io/nginx: localhost/k8s-deny-write`

Enforce an AppArmor profile:

```
sudo apparmor_parser /path/to/file
```

Apply an AppArmor profile to a container (supply the container name and profile name where indicated):

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    container.apparmor.security.beta.kubernetes.io/$containername: localhost/$profilename
```

Minimizing Microservice Vulnerabilities

SecurityContexts

Relevant Documentation

- Configure a Security Context for a Pod or Container
- API Reference - Pod Security Context
- API Reference - Container Security Context

Exam Tips

- securityContext offers a variety of security and access control-related settings.
- `spec.securityContext` sets securityContext settings at the Pod level. These settings apply to all containers in the Pod.
- `spec.containers[].securityContext` sets securityContext settings at the container level. These settings apply to individual containers within the Pod.

Sample pod with securityContext configuration:

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-pod
spec:
  securityContext:
    runAsUser: 1000
  containers:
  - name: busybox
    image: busybox
    command: [ "sh", "-c", "sleep 3600" ]
    securityContext:
      allowPrivilegeEscalation: false
```

PodSecurityPolicies

Relevant Documentation

- Pod Security Policies
- PodSecurityPolicy Admission Controller
- PodSecurityPolicy Deprecation: Past, Present, and Future

Exam Tips

- Use Pod security policies to enforce desired security configurations for new Pods.
- Pod security policies can reject Pods that don't meet the desired standard, or modify Pods by applying default settings.
- To use Pod security policies, you must first enable the PodSecurityPolicy admission controller. Use the `--enable-admission-plugins` flag on kube-apiserver to do this.
- In order to create a Pod, a user (or the Pod's ServiceAccount) must be authorized to use a PodSecurityPolicy via the use verb in RBAC.
- To apply a PodSecurityPolicy within the context of a specific namespace, authorize a ServiceAccount in that namespace to use the policy.

Sample PodSecurityPolicy:

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: psp-nonpriv
spec:
  privileged: false
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - secret
  - projected
```

Using OPA Gatekeeper

Relevant Documentation

- OPA Gatekeeper Docs

Exam Tips

- Open Policy Agent (OPA) Gatekeeper allows you to enforce custom policies on any k8s object at creation time.
- Constraint templates define reusable constraint logic and any parameters that can be passed in.
- Constraint objects apply a constraint template to a specific group of potential incoming objects, alongside specific parameters.

Sample ConstraintTemplate:

```
apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        openAPIV3Schema:
          properties:
            labels:
              type: array
              items: string
  targets:
  - target: admission.k8s.gatekeeper.sh
    rego: >-
      package k8srequiredlabels
      violation[{"msg": msg, "details": {"missing_labels": missing}}] {
        provided := {label | input.review.object.metadata.labels[label]}
        required := {label | label := input.parameters.labels[_]}
        missing := required - provided
        count(missing) > 0
        msg := sprintf("you must provide labels: %v", [missing])
      }
```

Sample constraint:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
  name: dep-must-have-contact
spec:
  match:
    kinds:
    - apiGroups: ["apps"]
      kinds: ["Deployment"]
  parameters:
    labels: ["contact"]
```

Secrets

Relevant Documentation

- Secrets

Exam Tips

- Secrets store sensitive data, and can pass it to containers.
- You can pass Secret data to a container using either environment variables or mounted volumes.
- To retrieve Secret data from the command line, you can use `kubectl get -o yaml` to get the Base64-encoded data, then decode it with `base64 --decode`.

Base64-encode a string when creating a Secret:

```
echo mypassword | base64
```

Get raw data from an existing Secret:

```
kubectl get secret my-secret -o yaml
```

Base64-decode existing Secret data:

```
echo Y2FudGZpbmRtZQo=  base64 --decode
```

Pass Secret data to a Pod in an environment variable:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-env-secret
spec:
  restartPolicy: Never
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'echo "The credentials are $USERNAME:$PASSWORD"']
    env:
    - name: USERNAME
      valueFrom:
        secretKeyRef:
          name: my-secret
          key: username
    - name: PASSWORD
      valueFrom:
        secretKeyRef:
          name: my-secret
          key: password
```

Pass Secret data to a Pod in a volume:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-vol-secret
spec:
  restartPolicy: Never
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'cat /etc/credentials/username; echo " "; cat /etc/credentials/password; sleep
3600']
    volumeMounts:
    - name: credentials
      mountPath: "/etc/credentials"
      readOnly: true
  volumes:
  - name: credentials
    secret:
      secretName: my-secret
```

Container Runtime Sandbox

Relevant Documentation

- gVisor Docs
- Kata Containers Docs
- Runtime Class

Exam Tips

- Container runtime sandboxes provide a specialized runtime with additional layers of isolation, allowing you to run untrusted workloads more securely.
- gVisor creates a runtime sandbox by running a Linux application kernel within the host OS. `runsc` is the OCI-compliant container runtime that allows Kubernetes to interface with gVisor.
- Kata Containers create a sandbox by transparently running containers of lightweight virtual machines.
- Use a RuntimeClass to define a specialized container runtime configuration, such as one that will use gVisor/runsc.
- Set the `runtimeClassName` property in a Pod specification to make the Pod use the container runtime sandbox.

Runtime Sandbox - A specialized container runtime with additional layers of security and isolation. Often used to run untrusted workloads.

Sample runtimeClass:

```
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: runsc-sandbox
handler: runsc
```

Pod in a runtime sandbox:

```
apiVersion: v1
kind: Pod
metadata:
  name: sandbox-pod
spec:
  runtimeClassName: runsc-sandbox
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'while true; do echo "Running..."; sleep 5; done']
```

Pod-to-Pod mTLS

Relevant Documentation

- Mutual Authentication (Wikipedia)
- Manage TLS Certificates in a Cluster

Exam Tips

- Mutual Transport Layer Security (mTLS) means clients and servers mutually authenticate with each other and encrypt their communications.
- You can obtain certificates using the Kubernetes API.

Signing Certificates

Relevant Documentation

- Manage TLS Certificates in a Cluster

Exam Tips

- Create a CertificateSigningRequest object to request a new certificate.
- Manage, approve, or deny requests via the command line with `kubectl certificate`.
- Once approved, the signed certificate can be retrieved from the `status.certificate` field of the CertificateSigningRequest.

Sample CertificateSigningRequest:

> **Note:** The request field must contain an actual base64-encoded certificate signing request.

```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: tls-svc-csr
spec:
  request: base64encodedcsr
  signerName: kubernetes.io/kubelet-serving
  usages:
  - digital signature
  - key encipherment
  - server auth
```

Approve a certificate signing request:

```
kubectl certificate approve my-csr
```

Once approved, the signed certificate can be found in the the `status.certificate` field of the CertificateSigningRequest.

Supply Chain Security

Minimizing Base Image Attack Surface

Relevant Documentation

- Overview of Cloud Native Security
- Images

Exam Tips

- Try to use images that run up-to-date software to minimize software vulnerabilities.
- Minimize the presence of unnecessary software in images that could increase security risks.
- Beware of the possibility of images that have been compromised by an attacker.

Whitelisting Allowed Image Registries

Relevant Documentation

- Blog: OPA Gatekeeper
- OPA Gatekeeper - GitHub

Exam Tips

- Limit users to only trusted image registries to prevent them from running images from untrusted sources in the cluster.
- You can limit registries using OPA Gatekeeper.

Validating Signed Images

Relevant Documentation

- Busybox 1.33.1 - Docker Hub

Exam Tips

- Container images can be signed with a hash generated from the image contents.
- To validate the image, you can append the hash to the image reference in your container spec with `image: imageName:tag@sha256:hash` .

Sample Pod configured so that a hash will be used to validate the image:

```
apiVersion: v1
kind: Pod
metadata:
  name: signed-image-pod
spec:
  restartPolicy: Never
  containers:
  - name: busybox
    image: busybox:1.33.1@sha256:9687821b96b24fa15fac11d936c3a633ce1506d5471ebef02c349d85bebb11b5
    command: ['sh', '-c', 'echo "Hello, world!"']
```

Analyzing a Dockerfile

Relevant Documentation

- Dockerfile Reference
- Sample Dockerfile

Exam Tips

- To avoid running the container as the root user, make sure that the final `USER` directive in the Dockerfile is not set to `root` or `0`.
- Avoid using the `:latest` tag in the `FROM` directive.
- Try to avoid including unnecessary software in the final image.
- Avoid storing sensitive data such as passwords in the Dockerfile (for example, using the `ENV` directive). Use Kubernetes Secrets instead.

Analyzing Resource YAML Files

Relevant Documentation

- Overview of Cloud Native Security
- Sample YAML File

Exam Tips

- When possible, avoid the use of host namespaces in your Pod configurations (i.e., with `hostNetwork: true`, `hostIPC: true`, or `hostPID: true`).
- When possible, avoid using privileged containers with `privileged: true`.
- Avoid running as user `root` or `0` in `securityContext.runAsUser`.
- Don't use the `:latest` tag, but instead use a specific, fixed tag to avoid downloading a new and potentially unvetted image.

Scanning Images for Known Vulnerabilities

Relevant Documentation

- Cloud Native Security - Container
- Trivy Documentation

Exam Tips

- Vulnerability scanning allows you to scan images to detect security vulnerabilities that have already been discovered and documented by security researchers.
- Trivy is a command-line tool that allows you to scan images by name and tag.
- Scan an image name and tag with Trivy like so:

```
trivy image busybox:1.33.1
```

- In some cases, you may need to omit `image` like so:

```
trivy nginx:1.14.1
```

Scan images with newer versions of Trivy:

```
trivy image nginx:1.14.2
```

Scan with an older version of Trivy:

```
trivy nginx:1.14.2
```

ImagePolicyWebhook Admission Controller

Relevant Documentation

- Cloud Native Security - Container
- ImagePolicyWebhook Admission Controller

Exam Tips

- Admission controllers intercept requests to the Kubernetes API before objects are created. They can allow objects to be created, prevent their creation, or make changes to objects before creating them.
- The ImagePolicyWebhook admission controller allows you to use customizable logic to approve or deny the creation of workloads based upon the container image being used.
- You can use the ImagePolicyWebhook admission controller to have an external application scan images for vulnerabilities automatically as workloads are created.
- The ImagePolicyWebhook admission controller sends a JSON request to an external service to determine if images are allowed.
- The external service provides a JSON response indicating whether the images are allowed or disallowed.
- Use `--enable-admission-plugins` in the kube-apiserver manifest to enable the ImagePolicyWebhook admission controller.
- Use `--admission-control-config-file` to specify the location of the admission control configuration file.
- If the config files are on the host file system, you may need to mount them to the kube-apiserver container.
- In the admission control config, `kubeConfigFile` specifies the location of a kubeconfig file. This file tells ImagePolicyWebhook how to reach the webhook backend.
- In the admission control config, `defaultAllow` controls whether or not workloads will be allowed if the backend webhook in unreachable.

Sample admission control config file:

```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
- name: ImagePolicyWebhook
  configuration:
    imagePolicy:
      kubeConfigFile: /etc/kubernetes/admission-control/imagepolicywebhook_backend.kubeconfig
      allowTTL: 50
      denyTTL: 50
      retryBackoff: 500
      defaultAllow: true
```

Sample kubeconfig:

Note: `clusters[0].cluster.server` stores the url of the backend webhook service.

```
apiVersion: v1
kind: Config
clusters:
- name: trivy-k8s-webhook
  cluster:
    certificate-authority: /etc/kubernetes/admission-control/imagepolicywebhook-ca.crt
    server: https://acg.trivy.k8s.webhook:8090/scan
contexts:
- name: trivy-k8s-webhook
  context:
    cluster: trivy-k8s-webhook
    user: api-server
current-context: trivy-k8s-webhook
preferences: {}
users:
- name: api-server
  user:
    client-certificate: /etc/kubernetes/admission-control/api-server-client.crt
    client-key: /etc/kubernetes/admission-control/api-server-client.key
```

How to enable ImagePolicyWebhook in the kube-apiserver manifest:

```
- --enable-admission-plugins=NodeRestriction,ImagePolicyWebhook
```

Monitoring, Logging, and Runtime Security

Behavioral Analytics (Falco)

Relevant Documentation

- Sysdig
- Falco
- Falco - formatting

Exam Tips

- Behavioral analytics is the process of monitoring what is happening within a system to detect malicious activity.
- One way to perform behavioral analytics in Kubernetes is to use tools like Falco.
- You can run Falco from the command line with the `falco` command. View options with `falco --help`.

- Use `-p` to customize output format (e.g., `-p %evt.time,%user.uid,%proc.name`). Use `falco --list` to see all available output fields.
- Use `-M` to set the number of seconds Falco should collect data for (e.g., `-M 45` to run for 45 seconds).

A sample Falco rule:

```
- rule: spawned_process_in_test_container
  desc: A process was spawned in the test container.
  condition: container.name = "falco-test" and evt.type = execve
  output: "%evt.time,%user.uid,%proc.name,%container.id,%container.name"
  priority: WARNING
```

Run Falco with a custom rules file for 45 seconds:

```
sudo falco -r falco-rules.yml -M 45
```

List available condition and output fields:

```
falco --list
```

Ensuring Containers Are Immutable

Relevant Documentation

- Container Security Context

Exam Tips

- Immutability means that containers do not change at runtime (e.g., by downloading and running new code).
- Containers that use privileged mode (i.e., `securityContext.privileged: true`) may also be considered mutable.
- Use `container.securityContext.readOnlyRootFilesystem` to prevent a container from writing to its file system.
- If an application needs to write to files, such as for caching or logging, you can use an emptyDir volume alongside `readOnlyRootFilesystem`.

Audit Logging

Relevant Documentation

- Auditing
- Policy Configuration Reference

Exam Tips

- Kubernetes auditing allows you to capture logs of all changes made through the Kubernetes API.
- In audit log policy rules, `level` identifies how detailed the log data should be for the rule.
  - `None` - Don't log anything for the rule.
  - `RequestResponse` - Log both the request and the API server response.
  - `Request` - Log only the request.
  - `Metadata` - Log only basic metadata about the request.
- In audit log policy rules, `resources` identifies the Kubernetes resource types the rule applies to.

- In audit log policy rules, `namespaces` limits the rule to only specific namespaces.
- Define audit rules in the audit policy configuration file.
- kube-apiserver flags for audit logging:
  - `--audit-policy-file` - Points to the audit policy config file.
  - `--audit-log-path` - The location of log output files.
  - `--audit-log-maxage` - The number of days to keep old log files.
  - `--audit-log-maxbackup` - The number of old log files to keep.

Sample audit policy file:

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
# Log changes to Namespaces at the RequestResponse level.
- level: RequestResponse
  resources:
  - group: ""
    resources: ["namespaces"]

# Log pod changes in the audit-test Namespace at Request level
- level: Request
  resources:
  - group: ""
    resources: ["pods"]
  namespaces: ["audit-test"]

# Log all ConfigMap and Secret changes at the Metadata level.
- level: Metadata
  resources:
  - group: ""
    resources: ["secrets", "configmaps"]

# Catch-all - Log all requests at the metadata level.
- level: Metadata
```

kube-apiserver flags:

```
  - command:
    - kube-apiserver
    - --audit-policy-file=/etc/kubernetes/audit-policy.yaml
    - --audit-log-path=/var/log/k8s-audit/k8s-audit.log
    - --audit-log-maxage=30
    - --audit-log-maxbackup=10
```