CKS Valid Exam Testking & CKS Reliable Exam Topics

Linux Foundation Certified Kubernetes Security Specialist (CKS) Sample Questions (Q30-Q35):

BONUS!!! Download part of TestkingPDF CKS dumps for free: https://drive.google.com/open?id=1rixlihY8NWuLhIabN35MKE4uW8rYNVmh

Three versions for CKS test materials are available, and you can choose the most suitable one according to your own needs. CKS PDF version is printable, and if you prefer to practice on paper, this version must be your taste. CKS Soft test engine can stimulate the real exam environment, and you can know the procedures for the exam, and your confidence will be strengthened. CKS Online Test engine supports all web browsers and it also supports Android and iOS etc. This version can give you a general review of what you have leant last time.

The CKS Exam covers a wide range of topics related to Kubernetes security, including authentication and authorization, network policies, cluster hardening, and vulnerability management. CKS exam is challenging and requires a solid understanding of Kubernetes architecture, network security, and Linux administration. Certified Kubernetes Security Specialist (CKS) certification is highly valued in the industry and is recognized as a standard for Kubernetes security professionals.

>> CKS Valid Exam Testking <<

CKS Reliable Exam Topics, Valid CKS Torrent

Our Certified Kubernetes Security Specialist (CKS) (CKS) exam dumps comes in three formats: Linux Foundation CKS PDF dumps file, desktop-based practice test software, and a web-based practice exam. These versions are specially designed to make Certified Kubernetes Security Specialist (CKS) (CKS) preparation for users easier. CKS Questions in these formats of

TestkingPDF's material are enough grasp every test topic in the shortest time possible.

The CKS Certification Exam is designed for individuals who have a deep understanding of Kubernetes security and are experienced in implementing security best practices in a Kubernetes environment. CKS exam covers a wide range of topics, including cluster setup, securing the Kubernetes API, network policies, securing Kubernetes workloads, and monitoring and logging. Candidates must be familiar with various Kubernetes security tools and be able to troubleshoot common security issues.

Linux Foundation Certified Kubernetes Security Specialist (CKS) Sample Questions (Q10-Q15):

NEW QUESTION # 10
Create a PSP that will only allow the persistentvolumeclaim as the volume type in the namespace restricted.
Create a new PodSecurityPolicy named prevent-volume-policy which prevents the pods which is having different volumes mount apart from persistentvolumeclaim.
Create a new ServiceAccount named psp-sa in the namespace restricted.
Create a new ClusterRole named psp-role, which uses the newly created Pod Security Policy prevent-volume-policy
Create a new ClusterRoleBinding named psp-role-binding, which binds the created ClusterRole psp-role to the created SA psp-sa.
Hint:
Also, Check the Configuration is working or not by trying to Mount a Secret in the pod maifest, it should get failed.
POD Manifest:
apiVersion: v1
kind: Pod
metadata:
name:
spec:
containers:
- name:
image:
volumeMounts:
- name:
mountPath:
volumes:
- name:
secret:
secretName:

Answer:

Explanation:
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
name: restricted
annotations:
seccomp.security.alpha.kubernetes.io/allowedProfileNames: 'docker/default,runtime/default'
apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default'
seccomp.security.alpha.kubernetes.io/defaultProfileName: 'runtime/default'
apparmor.security.beta.kubernetes.io/defaultProfileName: 'runtime/default' spec:
privileged: false
Required to prevent escalations to root.
allowPrivilegeEscalation: false
This is redundant with non-root + disallow privilege escalation,
but we can provide it for defense in depth.
requiredDropCapabilities:
- ALL
Allow core volume types.
volumes:
- 'configMap'
- 'emptyDir'
- 'projected'
- 'secret'

- 'downwardAPI'
Assume that persistentVolumes set up by the cluster admin are safe to use.
- 'persistentVolumeClaim'
hostNetwork: false
hostIPC: false
hostPID: false
runAsUser:
Require the container to run without root privileges.
rule: 'MustRunAsNonRoot'
seLinux:
This policy assumes the nodes are using AppArmor rather than SELinux.
rule: 'RunAsAny'
supplementalGroups:
rule: 'MustRunAs'
ranges:
Forbid adding the root group.
- min: 1
max: 65535
fsGroup:
rule: 'MustRunAs'
ranges:
Forbid adding the root group.
- min: 1
max: 65535
readOnlyRootFilesystem: false

NEW QUESTION # 11
You are tasked with hardening a Kubernetes cluster running on a public cloud provider. The cluster currently runs Kubernetes version 1.18 and has been exposed to the internet for several months. A security audit has identified several vulnerabilities in the current Kubernetes version, including CVE-2021-25743, which affects all versions prior to 1.22.
How do you upgrade your cluster to Kubernetes 1.22 and patch the vulnerabilities without disrupting the applications running on the cluster?

Answer:

Explanation:
Solution (Step by Step) :
1. Plan the upgrade:
- Identify the workloads running in the cluster.
- Understand the dependencies and configurations of each workload.
- Check compatibility of workloads with the new Kubernetes version.
- Research the recommended upgrade path for your cloud provider.
2. Prepare the environment:
- Create a backup of the cluster configuration. This includes the cluster manifest, service account configurations, and any custom resources.
- Test the upgrade process on a staging environment. This helps to identify potential issues and avoid downtime in the production cluster.
- Identify and fix any issues discovered in the staging environment. This could involve updating application configurations or deploying new versions of workloads.
3. Perform the upgrade:
- Use the recommended upgrade process for your cloud provider. Most cloud providers provide automated tools for Kubernetes upgrades.
- Monitor the upgrade process closely. Keep an eye on logs and metrics for any issues or errors.
- Rollback to the previous version if necessary. Have a plan to revert the upgrade if any critical issues arise.
4. Validate the upgrade:
- Verify,/ that all applications are running as expected. Check application logs, metrics, and functionality to ensure that there are no regressions.
- Confirm that the vulnerabilities have been patched. Use tools like 'kubectl audit or 'kubeadm upgrade' to verify the patched version.
Example using Google Kubernetes Engine:

- Create a new cluster with the desired Kubernetes version (1.22) in the Google Cloud Console.
- Use 'kubectl get nodes --all-namespaces to list the nodes in the existing cluster.
- Use 'kubectl drain' to drain the nodes in the existing cluster-
- Use 'kubectl cordon' to cordon the nodes in the existing cluster.
- Once the nodes are drained and cordoned, use 'kubectl delete node to delete the nodes in the existing cluster
- Join the nodes to the new cluster using 'kuoectl join
- Migrate the applications and configurations from the old cluster to the new cluster
- Delete the old cluster

This process ensures a minimal disruption to the applications during the upgrade, and that the vulnerabilities are patched effectively.

NEW QUESTION # 12

SIMULATION

Documentation Deployment, Pod, Namespace

You must connect to the correct host . Failure to do so may result in a zero score.

[candidate@base] $ ssh cks000028

Context

You must update an existing Pod to ensure the immutability of its containers.

Task

Modify the existing Deployment named lamp-deployment, running in namespace lamp, so that its containers:

. run with user ID 20000

. use a read-only root filesystem

. forbid privilege escalation

The Deployment's manifest file con be found at /home/candidate/finer-sunbeam/lamp-deployment.yaml.

Answer:

Explanation:

See the Explanation below for complete solution

Explanation:

1) Connect to the correct host

ssh cks000028

sudo -i

2) Use the right kubeconfig (safe in exam)

export KUBECONFIG=/etc/kubernetes/admin.conf

3) Open the provided Deployment manifest

vi /home/candidate/finer-sunbeam/lamp-deployment.yaml

4) Edit ONLY the Pod template security settings (add/modify these fields) Inside:

spec: -> template: -> spec:

4.1 Set container to run as user 20000

Add (or change) under the container securityContext::

securityContext:

runAsUser: 20000

4.2 Make root filesystem read-only

In the SAME container securityContext: ensure:

readOnlyRootFilesystem: true

4.3 Forbid privilege escalation

In the SAME container securityContext: ensure:

allowPrivilegeEscalation: false

 The container section should look like this (example - keep your existing image/ports/etc):

spec:

template:

spec:

containers:

- name: <your-container-name>

image: <unchanged>

securityContext:

runAsUser: 20000

readOnlyRootFilesystem: true

allowPrivilegeEscalation: false

If there are multiple containers, apply the same securityContext to each container.
Save and exit:
:wq
5) Apply the manifest (updates Deployment -> recreates Pods)
kubectl -n lamp apply -f /home/candidate/finer-sunbeam/lamp-deployment.yaml
6) Wait for rollout
kubectl -n lamp rollout status deployment/lamp-deployment
7) Verify the security settings are live
7.1 Check the Pod is running
kubectl -n lamp get pods -l app=lamp -o wide
(if label differs, just kubectl -n lamp get pods)
7.2 Verify the three fields on a running Pod
Pick the Pod name and run:
POD=$(kubectl -n lamp get pods -o jsonpath='{.items[0].metadata.name}') kubectl -n lamp get pod $POD -o jsonpath='{.spec.containers[0].securityContext.runAsUser} {"\n"} {.spec.containers[0].securityContext.readOnlyRootFilesystem} {"\n"} {.spec.containers[0].securityContext.allowPrivilegeEscalation} {"\n"}' Expected output:
20000
true
false
If the pod fails after readOnlyRootFilesystem=true
Don't change the requirement (task demands it). Usually the app needs writable dirs via volumes, but the task doesn't ask for that-so only adjust if the manifest already has volumes and just needs these securityContext fields.

NEW QUESTION # 13
You need to implement a secure way to handle sensitive configuration data for your applications deployed within a Kubernetes cluster. This data, including database credentials and API keys, must be protected from unauthorized access. Describe a secure solution, including specific configuration and tools to address this challenge.

Answer:

Explanation:
Solution (Step by Step) :
1. Utilize a Secret Management Solution:
- Choose a secure secret management solution designed for Kubernetes.
- Popular options include:
- Vault: A comprehensive secret management tool offering encryption, access control, and auditing.
- Hashicorp Vault: A popular open-source solution that provides a secure and centralized way to store, manage, and access secrets.
- AWS Secrets Manager: A managed service from AWS for securely storing and retrieving secrets.
2. Configure Secret Management:
- Integrate the chosen secret management solution with your Kubernetes cluster.
- This typically involves deploying the secret management tool as a containerized application within the cluster.
- Configure access control policies to restrict access to secrets based on roles or identities.
3. Store Secrets Securely:
- Store sensitive configuration data as secrets within the chosen solution.
- Utilize strong encryption mechanisms to protect the secrets at rest and in transit.
4. Retrieve Secrets within Pods:
- Provide mechanisms for your applications to access secrets securely.
- This can be achieved through:
- Kubernetes Secrets: Mount secrets as files within pod containers.
- Environment Variables: Inject secrets as environment variables.
- Secret Management APIs Use APIs provided by the secret management solution to fetch secrets within the application code.
5. Securely Rotate Secrets:
- Implement a process for regularly rotating secrets to minimize exposure in case ot compromise.
- Automate this process to ensure timely rotation.

NEW QUESTION # 14
You have a critical web application running in your Kubernetes cluster. This application relies on a database service that should only be accessible by the web application pods. You need to implement network security policies to enforce this restriction.

Answer:

Explanation:
Solution (Step by Step) :
1. Identify Application and Database Namespaces: Determine the namespaces where your web application and database are deployed. Let's assume
the web application is in namespace 'web-app' and the database is in namespace 'db-service'
2. Create Database Network Policy: Create a NetworkPolicy YAML file named 'do-access-policy.yamr to define the allowed access to the database:
- This policy allows ingress connections to the database pods only from pods in the 'web-apps namespace that have the label 'app: web-app' 3. Apply Database Network Policy: Use 'kubectr to apply the NetworkPolicy: bash kubectl apply -f db-access-policy.yaml 4. Verify Database Network Policy: Verify that the NetworkPolicy is applied: bash kubectl get networkpolicies -n db-service 5. Test Access: Try connecting to the database from a pod outside of the 'web-app' namespace. The connection should be blocked. Connections from pods within the 'web-app' namespace should be allowed. This NetworkPolicy restricts ingress traffic to the 'db-service' namespace. It only allows connections from pods in the 'web-app' namespace with the specified label. This effectively isolates the database and prevents unauthorized access.

NEW QUESTION # 15
.....