Linux Foundation CKS Practice Questions

Shared by Merritt on 18-08-2026

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

SIMULATION

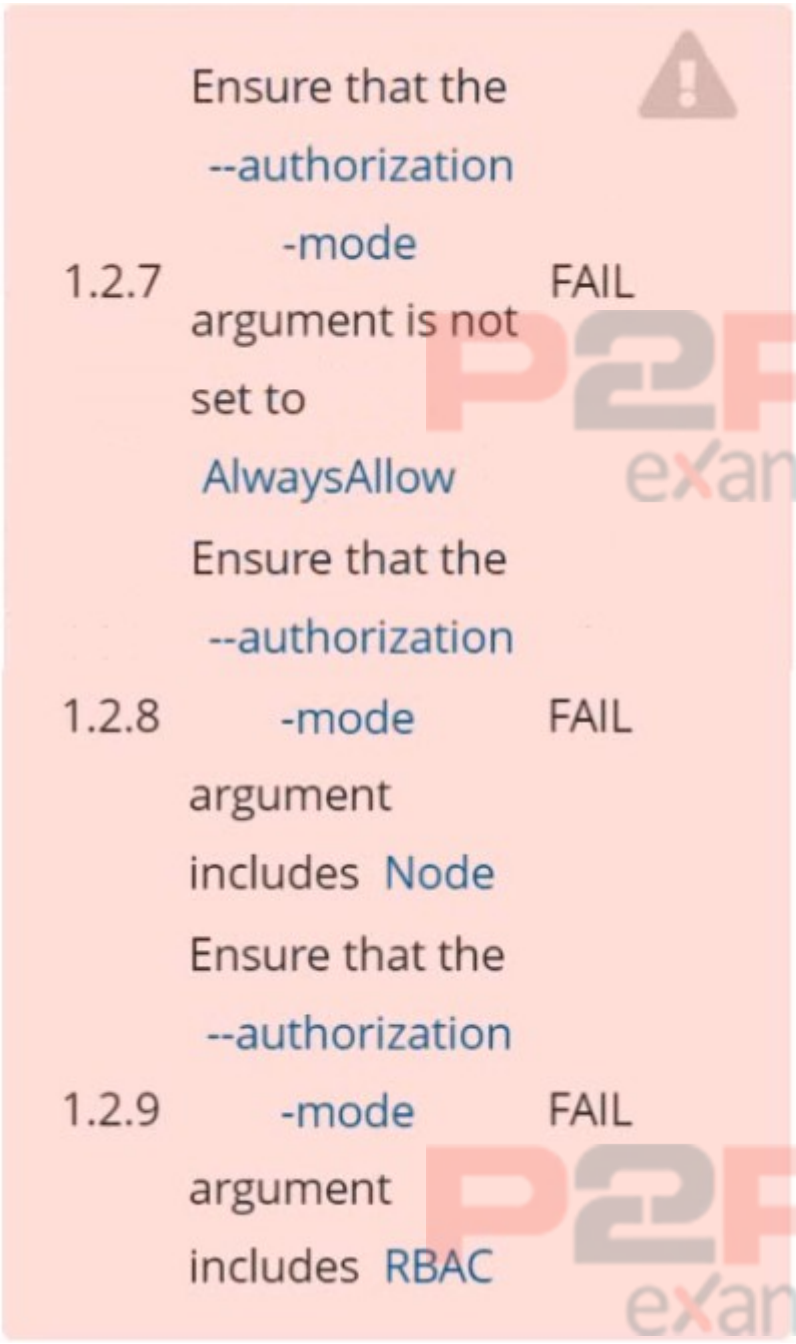
Context

A CIS Benchmark tool was run against the kubeadm-created cluster and found multiple issues that must be addressed immediately.

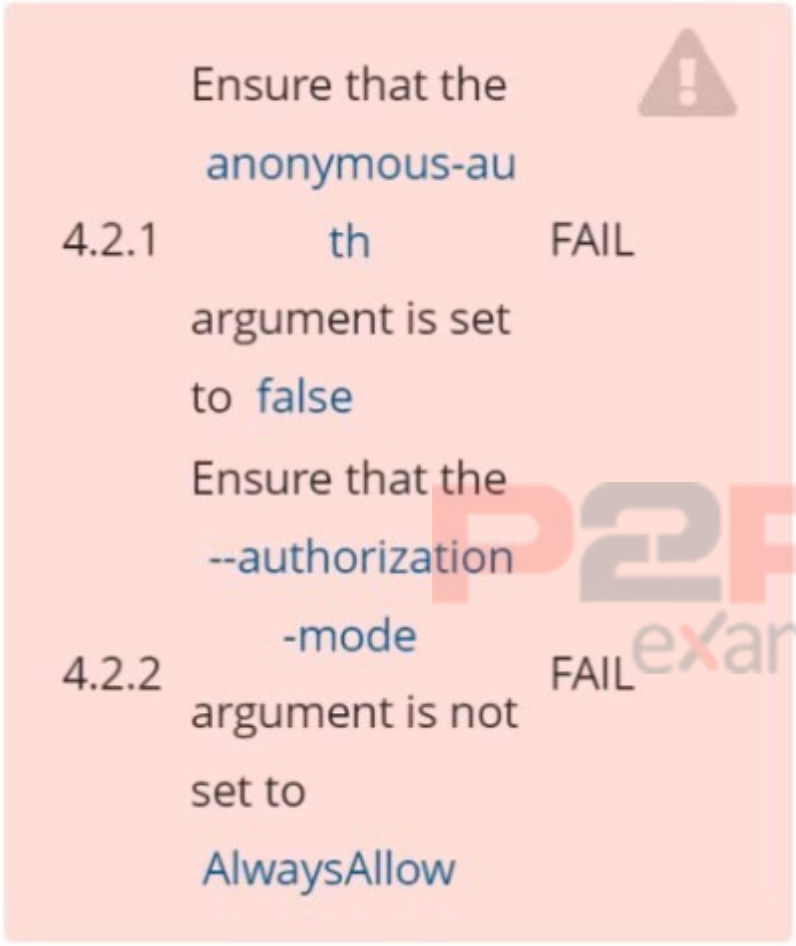
Task

Fix all issues via configuration and restart the affected components to ensure the new settings take effect.

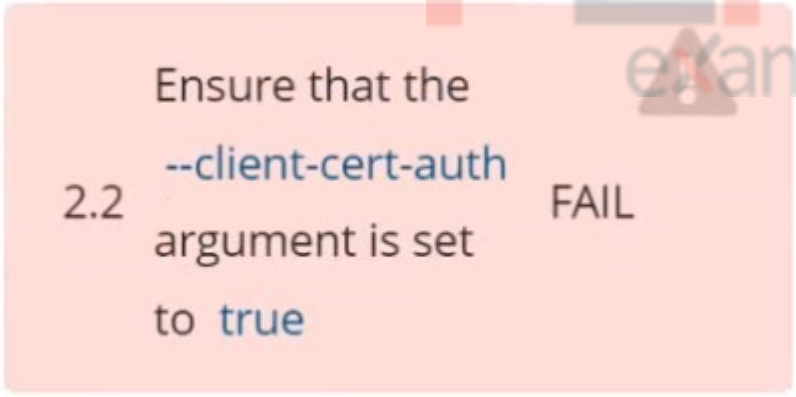
Fix all of the following violations that were found against the API server:



Fix all of the following violations that were found against the Kubelet:



Fix all of the following violations that were found against etcd:



Options:

A- See the Explanation below

Answer:

A

Question 2

Question Type: MultipleChoice

SIMULATION

You can switch the cluster/configuration context using the following command:

[desk@cli] $kubectl config use-context stage

Context:

A PodSecurityPolicy shall prevent the creation of privileged Pods in a specific namespace.

Task:

1. Create a new PodSecurityPolcy named deny-policy, which prevents the creation of privileged Pods.

2. Create a new ClusterRole name deny-access-role, which uses the newly created PodSecurityPolicy deny-policy.

3. Create a new ServiceAccount named psd-denial-sa in the existing namespace development.

Finally, create a new ClusterRoleBindind named restrict-access-bind, which binds the newly created ClusterRole deny-access-role to the newly created ServiceAccount psp-denial-sa

Options:

A- See the Explanation below

Answer:

A

Explanation:

Create psp to disallow privileged container

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: deny-access-role

rules:

- apiGroups: ['policy']

resources: ['podsecuritypolicies']

verbs: ['use']

resourceNames:

- ''deny-policy''

k create sa psp-denial-sa -n development

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: restrict-access-bing

roleRef:

kind: ClusterRole

name: deny-access-role

apiGroup: rbac.authorization.k8s.io

subjects:

- kind: ServiceAccount

name: psp-denial-sa

namespace: development

Explanation

master1 $ vim psp.yaml

apiVersion: policy/v1beta1

kind: PodSecurityPolicy

metadata:

name: deny-policy

spec:

privileged: false # Don't allow privileged pods!

seLinux:

rule: RunAsAny

supplementalGroups:

rule: RunAsAny

runAsUser:

rule: RunAsAny

fsGroup:

rule: RunAsAny

volumes:

- '*'

master1 $ vim cr1.yaml

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: deny-access-role

rules:

- apiGroups: ['policy']

resources: ['podsecuritypolicies']

verbs: ['use']

resourceNames:

- ''deny-policy''

master1 $k create sa psp-denial-sa -n development

master1 $ vim cb1.yaml

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: restrict-access-bing

roleRef:

kind: ClusterRole

name: deny-access-role

apiGroup: rbac.authorization.k8s.io

subjects:

Authorize specific service accounts:

- kind: ServiceAccount

name: psp-denial-sa

namespace: development

master1 $k apply -f psp.yaml

master1 $k apply -f cr1.yaml

master1 $k apply -f cb1.yaml

Question 3

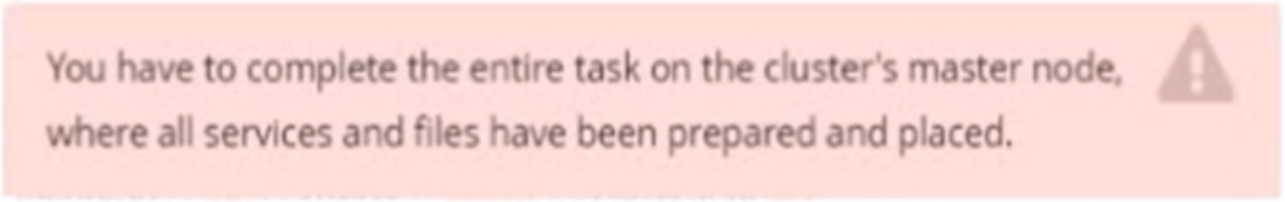
Question Type: MultipleChoice

SIMULATION

Context

A container image scanner is set up on the cluster, but it's not yet fully integrated into the cluster s configuration. When complete, the container image scanner shall scan for and reject the use of vulnerable images.

Task



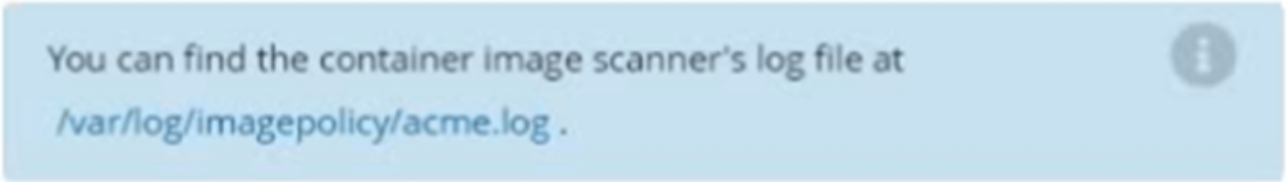
Given an incomplete configuration in directory /etc/kubernetes/epconfig and a functional container image scanner with HTTPS endpoint https://wakanda.local:8081 /image_policy :

1. Enable the necessary plugins to create an image policy

2. Validate the control configuration and change it to an implicit deny

3. Edit the configuration to point to the provided HTTPS endpoint correctly

Finally, test if the configuration is working by trying to deploy the vulnerable resource /root/KSSC00202/vulnerable-resource.yml.



Options:

A- See the Explanation below

Answer:

A

Question 4

Question Type: MultipleChoice

SIMULATION

Create a User named john, create the CSR Request, fetch the certificate of the user after approving it.

Create a Role name john-role to list secrets, pods in namespace john

Finally, Create a RoleBinding named john-role-binding to attach the newly created role john-role to the user john in the namespace john.

To Verify:Use the kubectl auth CLI command to verify the permissions.

Options:

A- See explanation below

Answer:

A

Explanation:

se kubectl to create a CSR and approve it.

Get the list of CSRs:

kubectl get csr

Approve the CSR:

kubectl certificate approve myuser

Get the certificate

Retrieve the certificate from the CSR:

kubectl get csr/myuser -o yaml

here are the role and role-binding to give john permission to create NEW_CRD resource:

kubectl apply -f roleBindingJohn.yaml --as=john

rolebinding.rbac.authorization.k8s.io/john_external-rosource-rb created

kind: RoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: john_crd

namespace: development-john

subjects:

- kind: User

name: john

apiGroup: rbac.authorization.k8s.io

roleRef:

kind: ClusterRole

name: crd-creation

kind: ClusterRole

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: crd-creation

rules:

- apiGroups: ['kubernetes-client.io/v1']

resources: ['NEW_CRD']

verbs: ['create, list, get']

Question 5

Question Type: MultipleChoice

SIMULATION

Analyze and edit the given Dockerfile

FROM ubuntu:latest

RUN apt-get update -y

RUN apt-install nginx -y

COPY entrypoint.sh /

ENTRYPOINT ["/entrypoint.sh"]

USER ROOT

Fixing two instructions present in the file being prominent security best practice issues

Analyze and edit the deployment manifest file

apiVersion: v1

kind: Pod

metadata:

name: security-context-demo-2

spec:

securityContext:

runAsUser: 1000

containers:

- name: sec-ctx-demo-2

image: gcr.io/google-samples/node-hello:1.0

securityContext:

runAsUser: 0

privileged: True

allowPrivilegeEscalation: false

Fixing two fields present in the file being prominent security best practice issues

Don't add or remove configuration settings; only modify the existing configuration settings

Whenever you need an unprivileged user for any of the tasks, use user test-user with the user id 5487

Options:

A- SeetheExplanationbelow

Answer:

A

Explanation:

FROM debian:latest

MAINTAINER k@bogotobogo.com

1 - RUN

RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -yq apt-utils

RUN DEBIAN_FRONTEND=noninteractive apt-get install -yq htop

RUN apt-get clean

2 - CMD

#CMD ['htop']

#CMD ['ls', '-l']

3 - WORKDIR and ENV

WORKDIR /root

ENV DZ version1

$ docker image build -t bogodevops/demo .

Sending build context to Docker daemon 3.072kB

Step 1/7 : FROM debian:latest

---> be2868bebaba

Step 2/7 : MAINTAINER k@bogotobogo.com

---> Using cache

---> e2eef476b3fd

Step 3/7 : RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -yq apt-utils

---> Using cache

---> 32fd044c1356

Step 4/7 : RUN DEBIAN_FRONTEND=noninteractive apt-get install -yq htop

---> Using cache

---> 0a5b514a209e

Step 5/7 : RUN apt-get clean

---> Using cache

---> 5d1578a47c17

Step 6/7 : WORKDIR /root

---> Using cache

---> 6b1c70e87675

Step 7/7 : ENV DZ version1

---> Using cache

---> cd195168c5c7

Successfully built cd195168c5c7

Successfully tagged bogodevops/demo:latest

Question 6

Question Type: MultipleChoice

SIMULATION

On the Cluster worker node, enforce the prepared AppArmor profile

#include

profile nginx-deny flags=(attach_disconnected) {

#include

file,

Deny all file writes.

deny /** w,

}

EOF'

Edit the prepared manifest file to include the AppArmor profile.

apiVersion: v1

kind: Pod

metadata:

name: apparmor-pod

spec:

containers:

- name: apparmor-pod

image: nginx

Finally, apply the manifests files and create the Pod specified on it.

Verify: Try to make a file inside the directory which is restricted.

Options:

A- SeetheExplanationbelow

Answer:

A

Question 7

Question Type: MultipleChoice

SIMULATION

Context

You must implement auditing for the kubeadm provisioned cluster.

Task

First, reconfigure the cluster 's API server, so that:

. the basic audit policy located at

/etc/kubernetes/logpolicy/audit-policy.yaml is used,

. logs are stored at /var/log/kubernetes/audit-logs.txt,

. and a maximum of 2 logs are retained for 10 days.

The cluster uses the Docker Engine as its container runtime . If needed, use the docker command to troubleshoot running containers.

The basic policy only specifies what not to log.

Next, edit and extend the basic policy to log:

. namespaces interactions at RequestResponse level

. the request body of deployments interactions in the namespace webapps

. ConfigMap and Secret interactions in all namespaces at the Metadata level

. all other requests at the Metadata level

Make sure the API server uses the extended policy.

Failure to do so may result in a reduced score.

Options:

A- See the Explanation below for complete solution

Answer:

A

Explanation:

1) Connect to the correct host

ssh cks000028

sudo -i

(If hostname differs in your exam, use the one shown in the question banner.)

2) Edit the API server static pod manifest

API server is a static pod in kubeadm.

vi /etc/kubernetes/manifests/kube-apiserver.yaml

3) Configure API server to enable auditing

Inside the command: section, ensure ALL of the following flags exist

(add them if missing, modify if present).

3.1 Use the given audit policy file

- --audit-policy-file=/etc/kubernetes/logpolicy/audit-policy.yaml

3.2 Store audit logs at the required location

- --audit-log-path=/var/log/kubernetes/audit-logs.txt

3.3 Retain a maximum of 2 log files

- --audit-log-maxbackup=2

3.4 Retain logs for 10 days

- --audit-log-maxage=10

Example (your file may have more flags --- that's fine):

- command:

- kube-apiserver

- --audit-policy-file=/etc/kubernetes/logpolicy/audit-policy.yaml

- --audit-log-path=/var/log/kubernetes/audit-logs.txt

- --audit-log-maxbackup=2

- --audit-log-maxage=10

Save and exit:

:wq

The API server will auto-restart (static pod).

Optional quick check:

docker ps | grep kube-apiserver

4) Edit and EXTEND the audit policy

Open the given basic policy:

vi /etc/kubernetes/logpolicy/audit-policy.yaml

The file already contains rules for what NOT to log.

You must ADD rules BELOW them (do not delete existing ones).

5) Add the required audit rules (EXACT ORDER)

Append the following rules in this order

(order matters in audit policies).

5.1 Log namespaces interactions at RequestResponse

- level: RequestResponse

resources:

- group: ''

resources: ['namespaces']

5.2 Log deployment request bodies in namespace webapps

- level: RequestResponse

namespaces: ['webapps']

resources:

- group: 'apps'

resources: ['deployments']

5.3 Log ConfigMap and Secret interactions (all namespaces) at Metadata

- level: Metadata

resources:

- group: ''

resources: ['configmaps', 'secrets']

5.4 Log all other requests at Metadata

This must be LAST

- level: Metadata

5.5 Final audit-policy.yaml should END like this

(existing 'do not log' rules above)

- level: RequestResponse

resources:

- group: ''

resources: ['namespaces']

- level: RequestResponse

namespaces: ['webapps']

resources:

- group: 'apps'

resources: ['deployments']

- level: Metadata

resources:

- group: ''

resources: ['configmaps', 'secrets']

- level: Metadata

Save and exit:

:wq

6) Make sure API server uses the EXTENDED policy

Touch the manifest to guarantee reload:

touch /etc/kubernetes/manifests/kube-apiserver.yaml

Wait a few seconds.

7) Verify auditing is working

7.1 Check audit log file exists

ls -l /var/log/kubernetes/audit-logs.txt

7.2 Generate test activity

kubectl get namespaces

kubectl get configmaps -A

7.3 Confirm logs are written

tail -n 20 /var/log/kubernetes/audit-logs.txt

You should see audit entries.

Question 8

Question Type: MultipleChoice

SIMULATION

Documentation Upgrading kubeadm clusters

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

[candidate@base] $ ssh cks000034

Context

The kubeadm provisioned cluster was recently upgraded, leaving one node on a slightly older version due to workload compatibility concerns.

Task

Upgrade the cluster node compute-0 to match the version of the control plane node.

Use a command like the following to connect to the compute node:

[candidate@cks000034] $ ssh compute-0

Do not modify any running workloads in the cluster.

Do not forget to exit from the compute node once you have completed your tasks:

[candidate@icompute-e] $ exit

Options:

A- See the Explanation below for complete solution

Answer:

A

Explanation:

Below is the CKS / CKA exam-style, exact step-by-step solution for Upgrading a kubeadm worker node.

Follow in order, type exact commands, no extra actions.

QUESTION --- Upgrade node compute-0 (EXAM MODE)

1) Connect to the correct host (control plane)

ssh cks000034

sudo -i

export KUBECONFIG=/etc/kubernetes/admin.conf

2) Identify the control plane Kubernetes version

This is the target version for compute-0.

kubectl get nodes

Example output:

NAME STATUS ROLES VERSION

control-plane Ready control-plane v1.27.4

compute-0 Ready <none> v1.26.6

Note the control-plane version

Example: v1.27.4

3) Drain the compute node (do NOT modify workloads manually)

kubectl drain compute-0 --ignore-daemonsets --delete-emptydir-data

Wait until drain completes successfully.

4) SSH into the compute node

ssh compute-0

sudo -i

5) Check current kubeadm version on compute node

kubeadm version

6) Upgrade kubeadm to match control plane version

Replace 1.27.4 with the exact control-plane version you observed.

apt-get update

apt-get install -y kubeadm=1.27.4-00

Verify:

kubeadm version

7) Run kubeadm upgrade for the node

kubeadm upgrade node

This updates node-specific configs (NO workloads touched).

8) Upgrade kubelet and kubectl to the same version

apt-get install -y kubelet=1.27.4-00 kubectl=1.27.4-00

9) Restart kubelet

systemctl daemon-reload

systemctl restart kubelet

systemctl status kubelet --no-pager

10) Exit the compute node (IMPORTANT)

exit

11) Uncordon the compute node (back on control plane)

kubectl uncordon compute-0

12) Final verification

kubectl get nodes

Expected:

NAME STATUS VERSION

compute-0 Ready v1.27.4

Question 9

Question Type: MultipleChoice

SIMULATION

Documentation Namespace, NetworkPolicy, Pod

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

[candidate@base] $ ssh cks000031

Context

You must implement NetworkPolicies controlling the traffic flow of existing Deployments across namespaces.

Task

First, create a NetworkPolicy named deny-policy in the prod namespace to block all ingress traffic.

The prod namespace is labeled env:prod

Next, create a NetworkPolicy named allow-from-prod in the data namespace to allow ingress traffic only from Pods in the prod namespace.

Use the label of the prod names & Click to copy traffic.

The data namespace is labeled env:data

Do not modify or delete any namespaces or Pods . Only create the required NetworkPolicies.

Options:

A- See the Explanation below for complete solution

Answer:

A

Explanation:

1) Connect to the correct host

ssh cks000031

sudo -i

2) Use admin kubeconfig (safe default)

export KUBECONFIG=/etc/kubernetes/admin.conf

PART A --- Deny ALL ingress traffic in prod namespace

Requirement:

NetworkPolicy name: deny-policy

Namespace: prod (namespace is labeled env=prod)

Effect: block all ingress

3) Create deny-policy in prod

Create the policy directly with kubectl (fastest & safest):

```
cat <<EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: deny-policy
namespace: prod
spec:
podSelector: {}
policyTypes:
- Ingress
EOF
```

What this does:

podSelector: {} selects all Pods in prod

No ingress: rules deny all ingress traffic

4) Verify

```
kubectl -n prod get networkpolicy deny-policy
```

PART B --- Allow ingress to data ONLY from Pods in prod

Requirement:

NetworkPolicy name: allow-from-prod

Namespace: data (namespace is labeled env=data)

Allow ingress only from Pods in prod namespace

Use namespace label (env=prod)

5) Create allow-from-prod policy in data

```
cat <<EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: allow-from-prod
```

namespace: data

spec:

podSelector: {}

policyTypes:

- Ingress

ingress:

- from:

- namespaceSelector:

matchLabels:

env: prod

EOF

What this does:

Applies to all Pods in data

Allows ingress only from namespaces labeled env=prod

All other ingress traffic is denied by default

6) Verify

kubectl -n data get networkpolicy allow-from-prod

FINAL CHECK (What the examiner expects)

kubectl get networkpolicy -n prod

kubectl get networkpolicy -n data

You should see:

deny-policy in prod

allow-from-prod in data

Question 10

Question Type: MultipleChoice

SIMULATION

Use the kubesec docker images to scan the given YAML manifest, edit and apply the advised changes, and passed with a score of 4 points.

kubesec-test.yaml

apiVersion: v1

kind: Pod

metadata:

name: kubesec-demo

spec:

containers:

- name: kubesec-demo

image: gcr.io/google-samples/node-hello:1.0

securityContext:

readOnlyRootFilesystem: true

Hint:docker run -i kubesec/kubesec:512c5e0 scan /dev/stdin < kubesec-test.yaml

Options:

A- See the Explanation below

Answer:

A

Explanation:

kubesec scan k8s-deployment.yaml

cat <<EOF > kubesec-test.yaml

apiVersion: v1

kind: Pod

metadata:

name: kubesec-demo

spec:

containers:

- name: kubesec-demo

image: gcr.io/google-samples/node-hello:1.0

securityContext:

readOnlyRootFilesystem: true

EOF

kubesec scan kubesec-test.yaml

docker run -i kubesec/kubesec:512c5e0 scan /dev/stdin < kubesec-test.yaml

kubesec http 8080 &

[1] 12345

{'severity':'info','timestamp':'2019-05-12T11:58:34.662+0100','caller':'server/server.go:69','message':'Starting HTTP server on port 8080'}

curl -sSX POST --data-binary @test/asset/score-0-cap-sys-admin.yml http://localhost:8080/scan

[

{

'object': 'Pod/security-context-demo.default',

'valid': true,

'message': 'Failed with a score of -30 points',

'score': -30,

'scoring': {

'critical': [

{

'selector': 'containers[] .securityContext .capabilities .add == SYS_ADMIN',

'reason': 'CAP_SYS_ADMIN is the most privileged capability and should always be avoided'

},

{

'selector': 'containers[] .securityContext .runAsNonRoot == true',

'reason': 'Force the running image to run as a non-root user to ensure least privilege'

},

// ...

Question 11

Question Type: MultipleChoice

SIMULATION

Create a PSP that will prevent the creation of privileged pods in the namespace.

Create a new PodSecurityPolicy named prevent-privileged-policy which prevents the creation of privileged pods.

Create a new ServiceAccount named psp-sa in the namespace default.

Create a new ClusterRole named prevent-role, which uses the newly created Pod Security Policy prevent-privileged-policy.

Create a new ClusterRoleBinding named prevent-role-binding, which binds the created ClusterRole prevent-role to the created SA psp-sa.

Also, Check the Configuration is working or not by trying to Create a Privileged pod, it should get failed.

Options:

A- SeetheExplanationbelow

Answer:

A

Explanation:

Create a PSP that will prevent the creation of privileged pods in the namespace.

$ cat clusterrole-use-privileged.yaml

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: use-privileged-psp

rules:

- apiGroups: ['policy']

resources: ['podsecuritypolicies']

verbs: ['use']

resourceNames:

- default-psp

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: privileged-role-bind

namespace: psp-test

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: use-privileged-psp

subjects:

- kind: ServiceAccount

name: privileged-sa

$ kubectl -n psp-test apply -f clusterrole-use-privileged.yaml

After a few moments, the privileged Pod should be created.

Create a new PodSecurityPolicy named prevent-privileged-policy which prevents the creation of privileged pods.

apiVersion: policy/v1beta1

kind: PodSecurityPolicy

metadata:

name: example

spec:

privileged: false # Don't allow privileged pods!

The rest fills in some required fields.

seLinux:

rule: RunAsAny

supplementalGroups:

rule: RunAsAny

runAsUser:

rule: RunAsAny

fsGroup:

rule: RunAsAny

volumes:

- '*'

And create it with kubectl:

kubectl-admin create -f example-psp.yaml

Now, as the unprivileged user, try to create a simple pod:

kubectl-user create -f- <<EOF

apiVersion: v1

kind: Pod

metadata:

name: pause

spec:

containers:

- name: pause

image: k8s.gcr.io/pause

EOF

The output is similar to this:

Error from server (Forbidden): error when creating 'STDIN': pods 'pause' is forbidden: unable to validate against any pod security policy: []

Create a new ServiceAccount named psp-sa in the namespace default.

$ cat clusterrole-use-privileged.yaml

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: use-privileged-psp

rules:

- apiGroups: ['policy']

resources: ['podsecuritypolicies']

verbs: ['use']

resourceNames:

- default-psp

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: privileged-role-bind

namespace: psp-test

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: use-privileged-psp

subjects:

- kind: ServiceAccount

name: privileged-sa

$ kubectl -n psp-test apply -f clusterrole-use-privileged.yaml

After a few moments, the privileged Pod should be created.

Create a new ClusterRole named prevent-role, which uses the newly created Pod Security Policy prevent-privileged-policy.

apiVersion: policy/v1beta1

kind: PodSecurityPolicy

metadata:

name: example

spec:

privileged: false # Don't allow privileged pods!

The rest fills in some required fields.

seLinux:

rule: RunAsAny

supplementalGroups:

rule: RunAsAny

runAsUser:

rule: RunAsAny

fsGroup:

rule: RunAsAny

volumes:

- '*'

And create it with kubectl:

kubectl-admin create -f example-psp.yaml

Now, as the unprivileged user, try to create a simple pod:

kubectl-user create -f- <<EOF

apiVersion: v1

kind: Pod

metadata:

name: pause

spec:

containers:

- name: pause

image: k8s.gcr.io/pause

EOF

The output is similar to this:

Error from server (Forbidden): error when creating 'STDIN': pods 'pause' is forbidden: unable to validate against any pod security policy: []

Create a new ClusterRoleBinding named prevent-role-binding, which binds the created ClusterRole prevent-role to the created SA psp-sa.

apiVersion: rbac.authorization.k8s.io/v1

This role binding allows 'jane' to read pods in the 'default' namespace.

You need to already have a Role named 'pod-reader' in that namespace.

kind: RoleBinding

metadata:

name: read-pods

namespace: default

subjects:

You can specify more than one 'subject'

- kind: User

name: jane # 'name' is case sensitive

apiGroup: rbac.authorization.k8s.io

roleRef:

'roleRef' specifies the binding to a Role / ClusterRole

kind: Role #this must be Role or ClusterRole

name: pod-reader # this must match the name of the Role or ClusterRole you wish to bind to

apiGroup: rbac.authorization.k8s.io

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

namespace: default

name: pod-reader

rules:

- apiGroups: ['] # '' indicates the core API group

resources: ['pods']

verbs: ['get', 'watch', 'list']

Question 12

Question Type: MultipleChoice

SIMULATION

A container image scanner is set up on the cluster.

Given an incomplete configuration in the directory

/etc/kubernetes/confcontrol and a functional container image scanner with HTTPS endpoint https://test-server.local.8081/image_policy

1. Enable the admission plugin.

2. Validate the control configuration and change it to implicit deny.

Finally, test the configuration by deploying the pod having the image tag as latest.

Options:

A- SeetheExplanationbelow

Answer:

A

Explanation:

ssh-add ~/.ssh/tempprivate

eval '$(ssh-agent -s)'

cd contrib/terraform/aws

vi terraform.tfvars

terraform init

terraform apply -var-file=credentials.tfvars

ansible-playbook -i ./inventory/hosts ./cluster.yml -e ansible_ssh_user=core -e bootstrap_os=coreos -b --become-user=root --flush-cache -e ansible_user=core