Linux Foundation CKS Daily Practice Exam New 2022 Updated 44 Questions [Q11-Q28

Linux Foundation CKS Daily Practice Exam New 2022 Updated 44 Questions
Use Valid CKS Exam - Actual Exam Question & Answer

NO.11 Cluster: dev

Master node: master1

Worker node: worker1

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

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

Task:

Retrieve the content of the existing secret named adam in the safe namespace.

Store the username field in a file names /home/cert-masters/username.txt, and the password field in a file named /home/cert-masters/password.txt.

1. You must create both files; they don’t exist yet.

2. Do not use/modify the created files in the following steps, create new temporary files if needed.

Create a new secret names newsecret in the safe namespace, with the following content:

Username: dbadmin

Password: moresecurepas

Finally, create a new Pod that has access to the secret newsecret via a volume:

Namespace: safe

Pod name: mysecret-pod

Container name: db-container

Image: redis

Volume name: secret-vol

Mount path: /etc/mysecret
1. Get the secret, decrypt it & save in files

k get secret adam -n safe -o yaml

2. Create new secret using –from-literal

[desk@cli] $k create secret generic newsecret -n safe –from-literal=username=dbadmin –from-literal=password=moresecurepass

3. Mount it as volume of db-container of mysecret-pod

Explanation

```
controlplane $ k get secret adam -n safe -o yaml
apiVersion: v1
data:
  password: c2VjcmV0cGFzcw==
  username: Y2VydC1tYXN0ZXJz
kind: Secret
metadata:
  creationTimestamp: "2021-06-13T11:56:32Z"
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:data:
        .: {}
        f:password: {}
        f:username: {}
      f:type: {}
    manager: kubectl-create
    operation: Update
    time: "2021-06-13T11:56:32Z"
  name: adam
  namespace: safe
  resourceVersion: "6405"
  selfLink: /api/v1/namespaces/safe/secrets/adam
  uid: da04fa1e-22a5-49d0-95d5-d73e40542033
type: Opaque
```



```
controlplane $ echo "c2VjcmV0cGFzcw==" | base64 -d | tee -a /home/cert-masters/password.txt
secretpasscontrolplane $
```

[desk@cli] $k create secret generic newsecret -n safe –from-literal=username=dbadmin –from-literal=password=moresecurepass secret/newsecret created

[desk@cli] $vim /home/certs_masters/secret-pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: mysecret-pod

namespace: safe

labels:

run: mysecret-pod

spec:

containers:

– name: db-container

image: redis

volumeMounts:

– name: secret-vol

mountPath: /etc/mysecret

readOnly: true

volumes:

– name: secret-vol

secret:

secretName: newsecret

[desk@cli] $ k apply -f /home/certs_masters/secret-pod.yaml

pod/mysecret-pod created

[desk@cli] $ k exec -it mysecret-pod -n safe – cat /etc/mysecret/username dbadmin

```
controlplane $ k exec -it mysecret-pod -n safe -- cat /etc/mysecret/username
dbadmincontrolplane $
```

[desk@cli] $ k exec -it mysecret-pod -n safe – cat /etc/mysecret/password moresecurepas

```
controlplane $ k exec -it mysecret-pod -n safe -- cat /etc/mysecret/password
moresecurepasscontrolplane $
```

NO.12 SIMULATION

a. Retrieve the content of the existing secret named default-token-xxxxx in the testing namespace.

Store the value of the token in the token.txt

b. Create a new secret named test-db-secret in the DB namespace with the following content:

username: mysql

password: password@123

Create the Pod name test-db-pod of image nginx in the namespace db that can access test-db-secret via a volume at path /etc/mysql-credentials
To add a Kubernetes cluster to your project, group, or instance:

Navigate to your:

Project’s Operations > Kubernetes page, for a project-level cluster.

Group’s Kubernetes page, for a group-level cluster.

Admin Area > Kubernetes page, for an instance-level cluster.

Click Add Kubernetes cluster.

Click the Add existing cluster tab and fill in the details:

Kubernetes cluster name (required) – The name you wish to give the cluster.

Environment scope (required) – The associated environment to this cluster.

API URL (required) – It’s the URL that GitLab uses to access the Kubernetes API. Kubernetes exposes several APIs, we want the “base” URL that is common to all of them. For example, https://kubernetes.example.com rather than https://kubernetes.example.com/api/v1.

Get the API URL by running this command:

kubectl cluster-info | grep -E ‘Kubernetes master|Kubernetes control plane’ | awk ‘/http/ {print $NF}’ CA certificate (required) – A valid Kubernetes certificate is needed to authenticate to the cluster. We use the certificate created by default.

List the secrets with kubectl get secrets, and one should be named similar to default-token-xxxxx. Copy that token name for use below.

Get the certificate by running this command:

kubectl get secret <secret name> -o jsonpath=”{[‘data’][‘ca.crt’]}”

NO.13 Given an existing Pod named test-web-pod running in the namespace test-system Edit the existing Role bound to the Pod’s Service Account named sa-backend to only allow performing get operations on endpoints.

Create a new Role named test-system-role-2 in the namespace test-system, which can perform patch operations, on resources of type

statefulsets.
* Create a new RoleBinding named test-system-role-2-binding binding the newly created Role to the Pod’s ServiceAccount sa-backend.

NO.14 Using the runtime detection tool Falco, Analyse the container behavior for at least 20 seconds, using filters that detect newly spawning and executing processes in a single container of Nginx.
* store the incident file art /opt/falco-incident.txt, containing the detected incidents. one per line, in the format [timestamp],[uid],[processName]

NO.15 Create a Pod name Nginx-pod inside the namespace testing, Create a service for the Nginx-pod named nginx-svc, using the ingress of your choice, run the ingress on tls, secure port.
* Send us your Feedback on this.

NO.16 SIMULATION

Create a Pod name Nginx-pod inside the namespace testing, Create a service for the Nginx-pod named nginx-svc, using the ingress of your choice, run the ingress on tls, secure port.
* Sendusyourfeedbackonit

NO.17 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
* Send us the Feedback on it.

NO.18 SIMULATION

Enable audit logs in the cluster, To Do so, enable the log backend, and ensure that

1. logs are stored at /var/log/kubernetes-logs.txt.

2. Log files are retained for 12 days.

3. at maximum, a number of 8 old audit logs files are retained.

4. set the maximum size before getting rotated to 200MB

Edit and extend the basic policy to log:

1. namespaces changes at RequestResponse

2. Log the request body of secrets changes in the namespace kube-system.

3. Log all other resources in core and extensions at the Request level.

4. Log “pods/portforward”, “services/proxy” at Metadata level.

5. Omit the Stage RequestReceived

All other requests at the Metadata level
Kubernetes auditing provides a security-relevant chronological set of records about a cluster. Kube-apiserver performs auditing. Each request on each stage of its execution generates an event, which is then pre-processed according to a certain policy and written to a backend. The policy determines what’s recorded and the backends persist the records.

You might want to configure the audit log as part of compliance with the CIS (Center for Internet Security) Kubernetes Benchmark controls.

The audit log can be enabled by default using the following configuration in cluster.yml:

services:

kube-api:

audit_log:

enabled: true

When the audit log is enabled, you should be able to see the default values at /etc/kubernetes/audit-policy.yaml The log backend writes audit events to a file in JSONlines format. You can configure the log audit backend using the following kube-apiserver flags:

–audit-log-path specifies the log file path that log backend uses to write audit events. Not specifying this flag disables log backend. – means standard out

–audit-log-maxage defined the maximum number of days to retain old audit log files

–audit-log-maxbackup defines the maximum number of audit log files to retain

–audit-log-maxsize defines the maximum size in megabytes of the audit log file before it gets rotated If your cluster’s control plane runs the kube-apiserver as a Pod, remember to mount the hostPath to the location of the policy file and log file, so that audit records are persisted. For example:

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

–audit-log-path=/var/log/audit.log

NO.19 Using the runtime detection tool Falco, Analyse the container behavior for at least 30 seconds, using filters that detect newly spawning and executing processes
* store the incident file art /opt/falco-incident.txt, containing the detected incidents. one per line, in the format [timestamp],[uid],[user-name],[processName]

NO.20 a. Retrieve the content of the existing secret named default-token-xxxxx in the testing namespace.

Store the value of the token in the token.txt

b. Create a new secret named test-db-secret in the DB namespace with the following content:

username: mysql

password: password@123

Create the Pod name test-db-pod of image nginx in the namespace db that can access test-db-secret via a volume at path /etc/mysql-credentials
To add a Kubernetes cluster to your project, group, or instance:

Navigate to your:

Project’s Operations > Kubernetes page, for a project-level cluster.

Group’s Kubernetes page, for a group-level cluster.

Admin Area > Kubernetes page, for an instance-level cluster.

Click Add Kubernetes cluster.

Click the Add existing cluster tab and fill in the details:

Kubernetes cluster name (required) – The name you wish to give the cluster.

Environment scope (required) – The associated environment to this cluster.

API URL (required) – It’s the URL that GitLab uses to access the Kubernetes API. Kubernetes exposes several APIs, we want the “base” URL that is common to all of them. For example, https://kubernetes.example.com rather than https://kubernetes.example.com/api/v1.

Get the API URL by running this command:

kubectl cluster-info | grep -E ‘Kubernetes master|Kubernetes control plane’ | awk ‘/http/ {print $NF}’ CA certificate (required) – A valid Kubernetes certificate is needed to authenticate to the cluster. We use the certificate created by default.

List the secrets with kubectl get secrets, and one should be named similar to default-token-xxxxx. Copy that token name for use below.

Get the certificate by running this command:

kubectl get secret <secret name> -o jsonpath=”{[‘data’][‘ca.crt’]}”

NO.21 Context:

Cluster: prod

Master node: master1

Worker node: worker1

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

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

Task:

Analyse and edit the given Dockerfile (based on the ubuntu:18:04 image)

/home/cert_masters/Dockerfile fixing two instructions present in the file being prominent security/best-practice issues.

Analyse and edit the given manifest file

/home/cert_masters/mydeployment.yaml fixing two fields present in the file being prominent security/best-practice issues.

Note: Don’t add or remove configuration settings; only modify the existing configuration settings, so that two configuration settings each are no longer security/best-practice concerns.

Should you need an unprivileged user for any of the tasks, use user nobody with user id 65535
1. For Dockerfile: Fix the image version & user name in Dockerfile

2. For mydeployment.yaml : Fix security contexts

Explanation

[desk@cli] $ vim /home/cert_masters/Dockerfile

FROM ubuntu:latest # Remove this

FROM ubuntu:18.04 # Add this

USER root # Remove this

USER nobody # Add this

RUN apt get install -y lsof=4.72 wget=1.17.1 nginx=4.2

ENV ENVIRONMENT=testing

USER root # Remove this

USER nobody # Add this

CMD [“nginx -d”]

```
FROM ubuntu:latest  # Remove this
FROM ubuntu:18.04   # Add this
USER root           # Remove this
USER nobody         # Add this
RUN apt get install -y lsof=4.72 wget=1.17.1 nginx=4.2
ENV  ENVIRONMENT=testing
USER root           # Remove this
USER nobody         # Add this
CMD ["nginx -d"]
```


[desk@cli] $ vim /home/cert_masters/mydeployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

creationTimestamp: null

labels:

app: kafka

name: kafka

spec:

replicas: 1

selector:

matchLabels:

app: kafka

strategy: {}

template:

metadata:

creationTimestamp: null

labels:

app: kafka

spec:

containers:

– image: bitnami/kafka

name: kafka

volumeMounts:

– name: kafka-vol

mountPath: /var/lib/kafka

securityContext:

{“capabilities”:{“add”:[“NET_ADMIN”],”drop”:[“all”]},”privileged”: True,”readOnlyRootFilesystem”: False, “runAsUser”: 65535} # Delete This

{“capabilities”:{“add”:[“NET_ADMIN”],”drop”:[“all”]},”privileged”: False,”readOnlyRootFilesystem”: True, “runAsUser”: 65535} # Add This resources: {} volumes:

– name: kafka-vol

emptyDir: {}

status: {}

Pictorial View:

[desk@cli] $ vim /home/cert_masters/mydeployment.yaml

NO.22 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
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 Reference: https://kubernetes.io/docs/concepts/policy/pod-security-policy/ master1 $ k apply -f cr1.yaml master1 $ k apply -f cb1.yaml master1 $ k apply -f psp.yaml master1 $ k apply -f cr1.yaml master1 $ k apply -f cb1.yaml Reference: https://kubernetes.io/docs/concepts/policy/pod-security-policy/

NO.23 SIMULATION

Given an existing Pod named nginx-pod running in the namespace test-system, fetch the service-account-name used and put the content in /candidate/KSC00124.txt Create a new Role named dev-test-role in the namespace test-system, which can perform update operations, on resources of type namespaces.

Create a new RoleBinding named dev-test-role-binding, which binds the newly created Role to the Pod’s ServiceAccount (found in the Nginx pod running in namespace test-system).
* Sendusyourfeedbackonit

NO.24 Before Making any changes build the Dockerfile with tag base:v1

Now Analyze and edit the given Dockerfile(based on ubuntu 16:04)

Fixing two instructions present in the file, Check from Security Aspect and Reduce Size point of view.

Dockerfile:

FROM ubuntu:latest

RUN apt-get update -y

RUN apt install nginx -y

COPY entrypoint.sh /

RUN useradd ubuntu

ENTRYPOINT [“/entrypoint.sh”]

USER ubuntu

entrypoint.sh

#!/bin/bash

echo “Hello from CKS”

After fixing the Dockerfile, build the docker-image with the tag base:v2
* To Verify: Check the size of the image before and after the build.

NO.25 Cluster: admission-cluster

Master node: master

Worker node: worker1

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

[desk@cli] $ kubectl config use-context admission-cluster

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:

You have to complete the entire task on the cluster’s master node, where all services and files have been prepared and placed.

Given an incomplete configuration in directory /etc/Kubernetes/config and a functional container image scanner with HTTPS endpoint https://imagescanner.local:8181/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 /home/cert_masters/test-pod.yml Note: You can find the container image scanner’s log file at /var/log/policy/scanner.log
[master@cli] $ cd /etc/Kubernetes/config

1. Edit kubeconfig to explicity deny

[master@cli] $ vim kubeconfig.json

“defaultAllow”: false # Change to false

2. fix server parameter by taking its value from ~/.kube/config

[master@cli] $cat /etc/kubernetes/config/kubeconfig.yaml | grep server

server:

3. Enable ImagePolicyWebhook

[master@cli] $ vim /etc/kubernetes/manifests/kube-apiserver.yaml

– –enable-admission-plugins=NodeRestriction,ImagePolicyWebhook # Add this

– –admission-control-config-file=/etc/kubernetes/config/kubeconfig.json # Add this Explanation

[desk@cli] $ ssh master

[master@cli] $ cd /etc/Kubernetes/config

[master@cli] $ vim kubeconfig.json

{

“imagePolicy”: {

“kubeConfigFile”: “/etc/kubernetes/config/kubeconfig.yaml”,

“allowTTL”: 50,

“denyTTL”: 50,

“retryBackoff”: 500,

“defaultAllow”: true # Delete this

“defaultAllow”: false # Add this

}

}

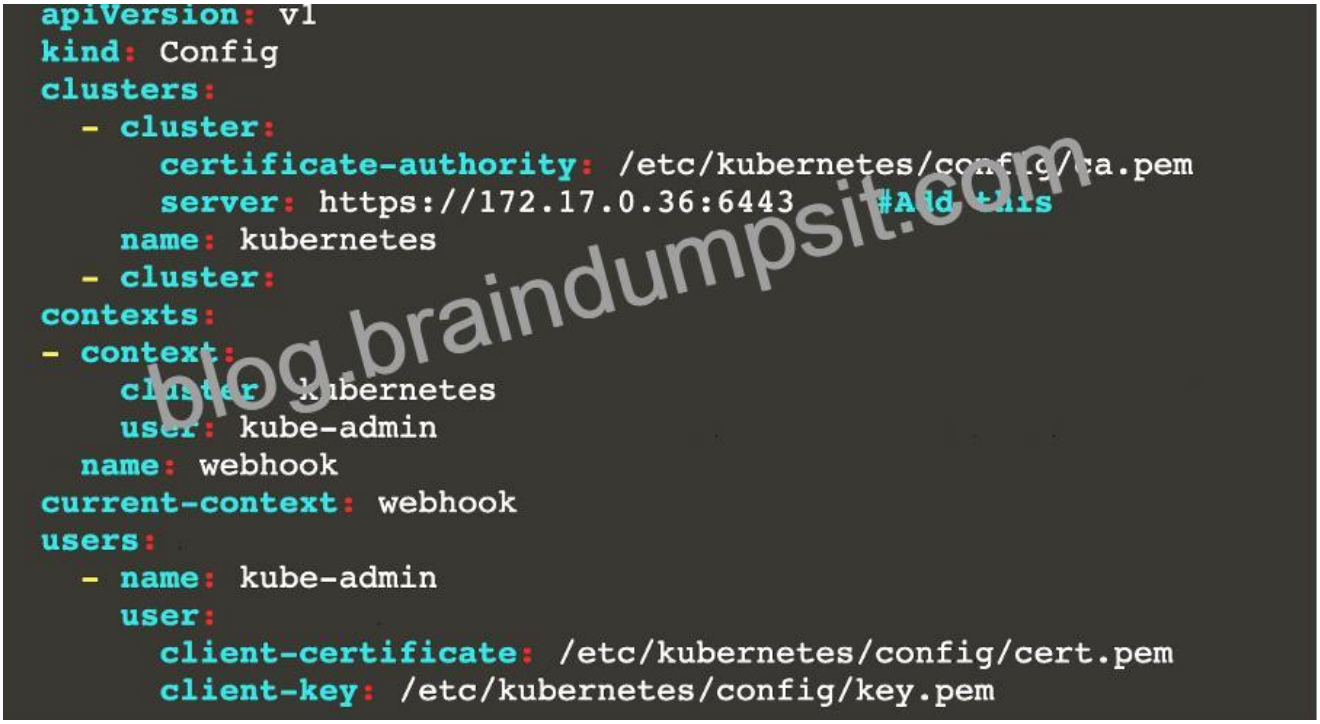
Note: We can see a missing value here, so how from where i can get this value

[master@cli] $cat ~/.kube/config | grep server

or

[master@cli] $cat /etc/kubernetes/manifests/kube-apiserver.yaml

[master@cli] $vim /etc/kubernetes/config/kubeconfig.yaml



[master@cli] $ vim /etc/kubernetes/manifests/kube-apiserver.yaml – –enable-admission-plugins=NodeRestriction # Delete This – –enable-admission-plugins=NodeRestriction,ImagePolicyWebhook # Add this – –admission-control-config-file=/etc/kubernetes/config/kubeconfig.json # Add this Reference: https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/

– –enable-admission-plugins=NodeRestriction # Delete This

– –enable-admission-plugins=NodeRestriction,ImagePolicyWebhook # Add this

– –admission-control-config-file=/etc/kubernetes/config/kubeconfig.json # Add this

[master@cli] $ vim /etc/kubernetes/manifests/kube-apiserver.yaml – –enable-admission-plugins=NodeRestriction # Delete This – –enable-admission-plugins=NodeRestriction,ImagePolicyWebhook # Add this – –admission-control-config-file=/etc/kubernetes/config/kubeconfig.json # Add this Reference: https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/

NO.26 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.
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”]

NO.27 You must complete this task on the following cluster/nodes:

Cluster: trace

Master node: master

Worker node: worker1

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

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

Given: You may use Sysdig or Falco documentation.

Task:

Use detection tools to detect anomalies like processes spawning and executing something weird frequently in the single container belonging to Pod tomcat.

Two tools are available to use:

1. falco

2. sysdig

Tools are pre-installed on the worker1 node only.

Analyse the container’s behaviour for at least 40 seconds, using filters that detect newly spawning and executing processes.

Store an incident file at /home/cert_masters/report, in the following format:

[timestamp],[uid],[processName]

Note: Make sure to store incident file on the cluster’s worker node, don’t move it to master node.
$vim /etc/falco/falco_rules.local.yaml

– rule: Container Drift Detected (open+create)

desc: New executable created in a container due to open+create

condition: >

evt.type in (open,openat,creat) and

evt.is_open_exec=true and

container and

not runc_writing_exec_fifo and

not runc_writing_var_lib_docker and

not user_known_container_drift_activities and

evt.rawres>=0

output: >

%evt.time,%user.uid,%proc.name # Add this/Refer falco documentation

priority: ERROR

$kill -1 <PID of falco>

Explanation

[desk@cli] $ ssh node01

[node01@cli] $ vim /etc/falco/falco_rules.yaml

search for Container Drift Detected & paste in falco_rules.local.yaml

[node01@cli] $ vim /etc/falco/falco_rules.local.yaml

– rule: Container Drift Detected (open+create)

desc: New executable created in a container due to open+create

condition: >

evt.type in (open,openat,creat) and

evt.is_open_exec=true and

container and

not runc_writing_exec_fifo and

not runc_writing_var_lib_docker and

not user_known_container_drift_activities and

evt.rawres>=0

output: >

%evt.time,%user.uid,%proc.name # Add this/Refer falco documentation

priority: ERROR

[node01@cli] $ vim /etc/falco/falco.yaml

NO.28 SIMULATION

Create a new ServiceAccount named backend-sa in the existing namespace default, which has the capability to list the pods inside the namespace default.

Create a new Pod named backend-pod in the namespace default, mount the newly created sa backend-sa to the pod, and Verify that the pod is able to list pods.

Ensure that the Pod is running.
A service account provides an identity for processes that run in a Pod.

When you (a human) access the cluster (for example, using kubectl), you are authenticated by the apiserver as a particular User Account (currently this is usually admin, unless your cluster administrator has customized your cluster). Processes in containers inside pods can also contact the apiserver. When they do, they are authenticated as a particular Service Account (for example, default).

When you create a pod, if you do not specify a service account, it is automatically assigned the default service account in the same namespace. If you get the raw json or yaml for a pod you have created (for example, kubectl get pods/<podname> -o yaml), you can

see the spec.serviceAccountName field has been automatically set.

You can access the API from inside a pod using automatically mounted service account credentials, as described in Accessing the Cluster. The API permissions of the service account depend on the authorization plugin and policy in use.

In version 1.6+, you can opt out of automounting API credentials for a service account by setting automountServiceAccountToken: false on the service account:

apiVersion: v1

kind: ServiceAccount

metadata:

name: build-robot

automountServiceAccountToken: false

…

In version 1.6+, you can also opt out of automounting API credentials for a particular pod:

apiVersion: v1

kind: Pod

metadata:

name: my-pod

spec:

serviceAccountName: build-robot

automountServiceAccountToken: false

…

The pod spec takes precedence over the service account if both specify a automountServiceAccountToken value.