CKA Lab Part 5 - Security

Lab 1 - RBAC within a namespace

Implement the following:

- Create the namespace “rbac-test”
- Create the service account “rbac-test-sa”
- Create a role “rbac-test-role” that grants the following pod level resources:
    - Get
    - Watch
    - List
- Bind the “rbac-test-a” service account to the “rbac-test-role” role

```
apiVersion: v1
kind: Namespace
metadata:
 name: rbac-test
---
apiVersion: v1
kind: ServiceAccount
metadata:
 name: rbac-test-sa
 namespace: rbac-test
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 name: rbac-test-role
 namespace: rbac-test
rules:
 - apiGroups: [""]
   resources: ["pods"]
   verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: rbac-test-rolebinding
 namespace: rbac-test
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: Role
 name: rbac-test-role
subjects:
- kind: ServiceAccount
 name: rbac-test-sa
 namespace: rbac-test
```

- Test RBAC is working by trying to do something the service account is not authorised to do

Getting pods is permitted:

```
kubectl -n rbac-test --as=system:serviceaccount:rbac-test:rbac-test-sa auth can-i get pods
yes
```

Getting secrets is not:

```
kubectl -n rbac-test --as=system:serviceaccount:rbac-test:rbac-test-sa auth can-i get secrets
no
```

Lab 2 - RBAC within a cluster

Implement the following:

- Create the user “cluster-user-secretadmin” authenticating with a password
- Create a role “cluster-role-secretadmin” that grants the following cluster level secret resources:

    - Get
    - Watch
    - List
- Bind “cluster-user-secretadmin” user to the “cluster-role-secretadmin”

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
 name: cluster-user-secretadmin
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: cluster-role-secretadmin
rules:
 - apiGroups: [""]
   resources: ["secret"]
   verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: rbac-test-clusterRolebinding
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: cluster-role-secretadmin
subjects:
- kind: User
 name: cluster-user-secretadmin
```

<u>Lab 3 - Network security policy</u>

- Create a nginx pod that listens on port 80, note the IP assigned to it.

```
kubectl run nginx-web1 --image=nginx --labels="tier=web,env=test"
```

- Create two pods that can use “curl” named busybox1 and busybox2. Note the IP addresses assigned to them. Label them with tier:jumppod

```
kubectl run busybox1 --image=pstauffer/curl --labels="tier=jumppod,env=test"
-- "sleep" "30000
kubectl run busybox2 --image=pstauffer/curl --labels="tier=jumppod,env=test"
-- "sleep" "30000
```

- Take a interactive shell to busybox1 and run:
    - Curl [IP Address of nginx pod]. You should get a HTML response.

kubectl exec -it busybox1-67c6755c8-sjgsr sh

/ # curl 10.10.57.4:80

<!DOCTYPE html>

****

****

<title>Welcome to nginx!

<style>

- Create a NetworkPolicy rule that blocks all ingress traffic to the nginx pod

```
apiVersion: networking.k8s.io/v1
```

```
kind: NetworkPolicy
metadata:
 name: deny-to-nginx
spec:
 podSelector:
   matchLabels:
     tier: web
 policyTypes:
 - Ingress
```

- Rerun the curl command from busybox1, it should fail.

```
kubectl exec -it busybox1-76b464d884-gf2cp sh
/ # curl 10.10.57.4:80
^C
```

- Create a NetworkPolicy that blocks all ingress traffic to the nginx pod with the exception of all pods labelled with tier:jumppod

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: deny-to-nginx
spec:
 podSelector:
   matchLabels:
     tier: web
 policyTypes:
 - Ingress
 ingress:
   - from:
     - podSelector:
         matchLabels:
          tier: jumppod
```

<u>Lab 4 - Enable Pod Security Policy</u>

Configure the admission controller in your cluster to use PodSecurityPolicy

```
sudo nano /etc/kubernetes/manifests/kube-apiserver.yaml
```

Change the line

```
- --enable-admission-plugins=NodeRestriction
```

To

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

<u>Lab 5 - Create policies</u>

Create two pod security policies

- One named “Privileged” with no restrictions

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: privileged
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - '*'
  volumes:
  - '*'
  hostNetwork: true
  hostPorts:
  - min: 0
    max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
```

- One named “Restricted” with the following restrictions
  - Cannot run privilaged containers
  - Can only be exposed on port 433

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: privileged
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
spec:
  privileged: false
  allowPrivilegeEscalation: false
  allowedCapabilities:
  - '*'
  volumes:
  - '*'
  hostNetwork: true
  hostPorts:
  - min: 443
    max: 443
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
```

Lab 6 - Security Context

Create a pod that defines subsequent containers to run as a user id of 600

```
apiVersion: v1
kind: Pod
metadata:
 name: security-context-demo
spec:
 securityContext:
   runAsUser: 600
 containers:
   - name : security-context
     image: busybox
     command: [ "sh", "-c", "sleep 1h" ]
```

Lab 7 - Secure persistent key value store

- Generate a key that will be used to encrypt information located in etcd and create the respective configuration file

```
head -c 32 /dev/urandom | base64
yriXiiDjtmUdAR/E8qIMWd0xR4YMaqZAqZAj3KJiTSM=

kind: EncryptionConfiguration
apiVersion: apiserver.config.k8s.io/v1
resources:
  - resources:
 - secrets
 providers:
 - aescbc:
     keys:
     - name: key1
       secret: yriXiiDjtmUdAR/E8qIMWd0xR4YMaqZAqZAj3KJiTSM=
 - identity: {}
```

- Modify the API server to leverage a encryption configuration leveraging the key generated in step 1

```
sudo cat kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
 component: kube-apiserver
 tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
 - kube-apiserver
 - --encryption-provider-config=/etc/kubernetes/config/securityconfig.conf

Note: Ensure the location is somewhere that the pod has access to, as defined in the volume and volumemounts section of the config file
```

- Create a secret called “testsecret” via any applicable means. Verify the contents are encrypted

```
sudo ETCDCTL_API=3 etcdctl get /registry/secrets/default/test-secret  --cacert /etc/kubernetes/pki/etcd/server.crt --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key
[sudo] password for david:
/registry/secrets/default/test-secret
0□=□□□□□□H=9□□□1$_□□6□□os□Uj+6ə[□□Z1□ □o□se<J□□_□□d□□;□~□N\□*□□□h□!□gez□PD
```

For a secret that’s not encrypted

```
sudo ETCDCTL_API=3 etcdctl get /registry/secrets/default/my-secret  --cacert /etc/kubernetes/pki/etcd/server.crt --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key
/registry/secrets/default/my-secret
k8s

v1Secret□
N
     my-secret▒default"*$5fa1ccff-62aa-11e9-a64a-005056afc0bc2□□□z▒
password
     somepassword▒
username
     someusername
```