**Q1 – 6%**

**Context**

A pod fails to run because of an incorrectly specified ServiceAccount.

Task

Create a new ServiceAccount named *append-sa* in the existing namespace *test*, which must not have access to **any** secretes.

Inspect the Pod named *append* running in the namespace test.

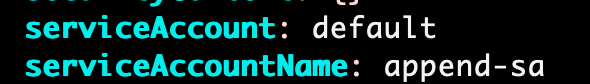
Edit the Pod to use the newly created ServiceAccount.

**Actual Answer**

Step 1: k create sa <service-account name> -n <namespace name>

Step2:k get sa -n <namespace name>

Step 8: Edit the pod and change the serviceAccountName as below

****

automountServiceAccountToken: false ensure this in the service account yaml file

automountServiceAccountToken: false ensure this in the pod yaml file

Step 9: k apply -f append.yaml

**Verification**

Step 10: k get po append -n test

**Q2 – 4%**

**Context**

A default-deny NetworkPolicy avoids to accidentally expose a Pod in a namespace that doesn’t have any other NetworkPolicy defined.

**Task**

Create a new default-deny NetworkPolicy named *notallowedpolicy* in the namespace *preprod* for all traffic of type Ingress + Egress.

The new NetworkPolicy must deny all Ingress + Egress traffic in the namespace preprod.

Apply the newly created default-deny NetworkPolicy to all Pods running in namespace preprod.

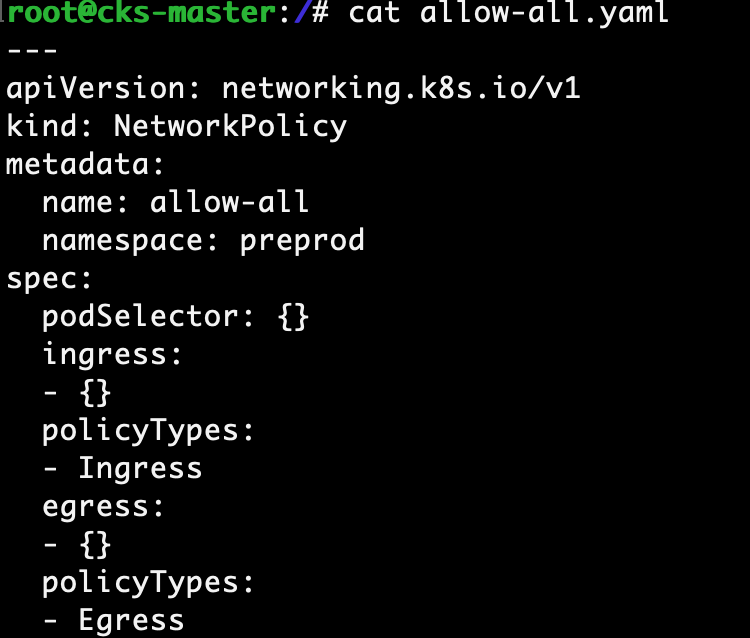
*You can find a skeleton manifest file at /home/candidate/KSCS00101/network-policy.yaml*

**Simulation of Question**

**Step 1: vim deny-all.yaml**

**Step 2: create -f deny-all.yaml**

**Step 3: cat deny-all.yaml**

****

**Actual Answer**

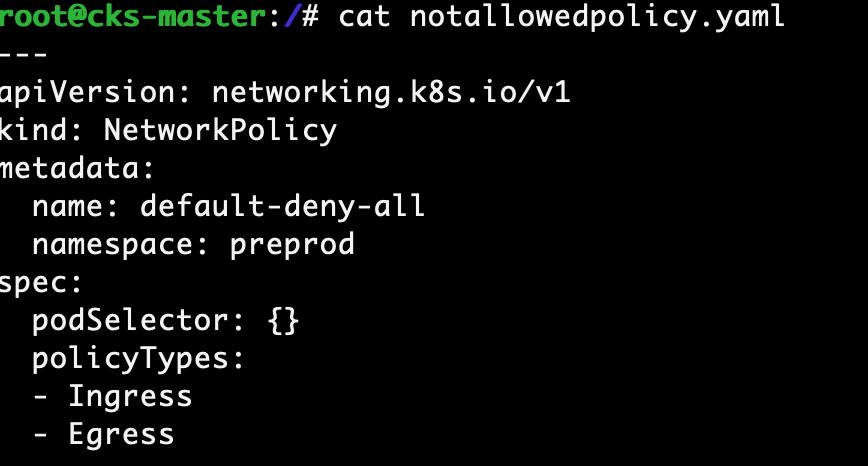
Step 3:create a network policy by copying the code from the default-deny-all policy document give the name to the file as notallowedpolicy.yaml

Step 4: Under the policyTypes of the yaml file make sure to include

-Ingress

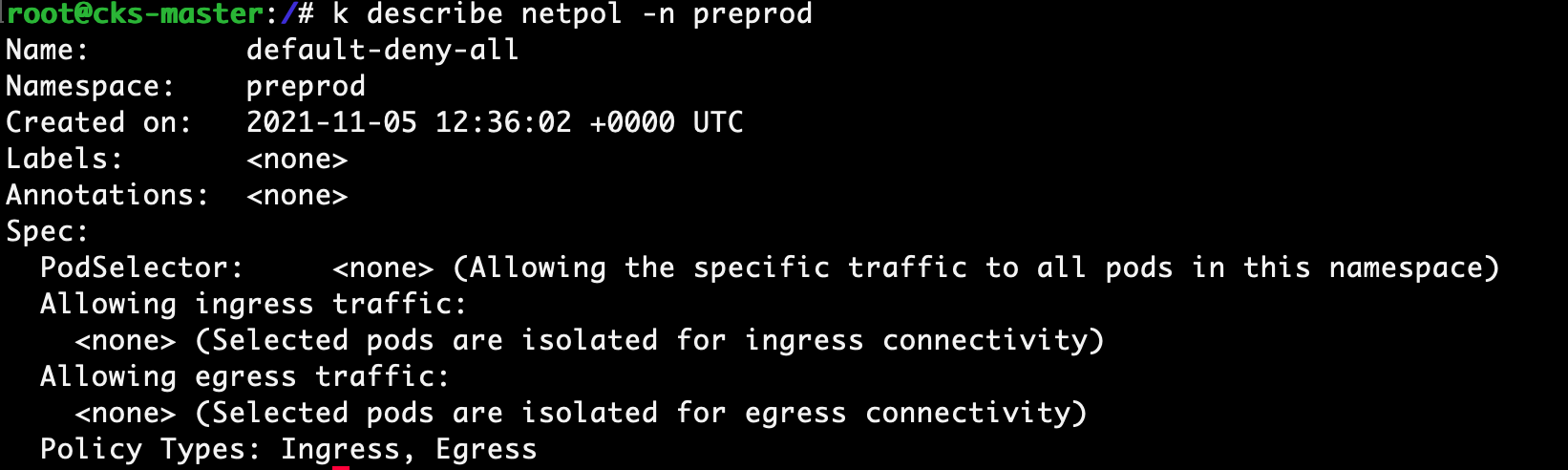
-Egress

See below yaml file as reference



Step 5:k apply -f deny-all.yaml

**Validation**

****

**Q3 – 7%**

**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:

Ensure that the

--authorization-mode argument is not set to AlwaysAllow FAIL

--authorization-mode argument includes Node FAIL

--authorization-mode argument includes RBAC FAIL

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

Ensure that the anonymous-auth argument is set to false FAIL

Ensure that the –authorization-mode argument is not set to AlwaysAllow FAIL

Use Webhook authn/authz where possible.

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

Ensure that the –client-cert-auth argument is set to true FAIL

**Actual Answer**

Step 1: SSH to the master node

Step 2: vim /etc/kubernetes/mainfests/kube-apiserver.yaml

Step 3: Under the spec: containers: section

Remove AlwaysAllow line

Add Node in the --authorization-mode=Node

Add RBAC in the –authorization-mode=RBAC

Step 4: vim /var/lib/kubelet/config.yaml

Look for anonymous tag under authentication and anonymous: enabled: false

Look for AlwaysAllow under authorization and set mode: Webhook

systemctl daemon-reload

systemctl restart kubelet

systemctl enable kubelet

switch back to the exam terminal

ssh to the worker node

Repeat Step 4 in the worker node

Switch back to the exam terminal

Step 5: ssh to the master node

cd /etc/kubernetes/manifest

vim etcd.yaml

set –client-cert-auth=true under the spec:containers: section

**Q4 6%**

**Context**

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

**Task**

Create a new PodSecurity named *restrict-policy*, which prevents the creation of privileged Pods.

Create a new ClusterRole named *restrict-access-role*, which uses the newly created PodSecurityPolicy *restrict-policy*.

Create a new ServiceAccount named *psp-denial-sa* in the existing namespace *staging*.

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

**Skeleton manifest files are located in**

/home/candidate/ksmv00102/pod-security-policy.yaml

/home/candidate/ksmv00102/cluster-role.yaml

/home/candidate/ksmv00102/service-account.yaml

/home/candidate/ksmv00102/cluster-role-binding.yaml

**Q5 4%**

**Context**

A role bound to a Pod’s ServiceAccount grants overly permissive permissions. Complete the following tasks to reduce the set of permissions.

**Task**

Given an existing Pod named *web-pod* running in the namespace *db*.

Edit the existing Role bound to the Pod’s ServiceAccount *service-account-web* to **only** allow performing get or append operations, **only** on resources of type Endpoints.

Create a new Role named role-2 in the namespace db, which **only** allows performing delete operations, **only** on resources of type *namespaces*.

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

Don’t delete the existing RoleBinding.

**Q6 12%**

**Task**

Enable audit logs in the cluster.

To do so, enable the log backend, and ensure that:

* logs are stored at /var/log/kubernetes/audit-logs.txt
* log files are retained for 10 days
* at maximum, a number of 2 old audit log files are retained

A basic policy is provided at /etc/kubernetes/logpolicy/sample-policy.yaml. It only specifies what not to log. The base policy is located on the cluster’s master node.

Edit and extend the basic policy to log:

* namespaces changes at RequestResponse level
* the request body of *persistentvolumes* changes in the namespace *front-apps*
* ConfigMap and Secret changes in all namespaces at the *Metadata* level

Also, add a catch-all rule to log all other requests at the *Metadata* level.

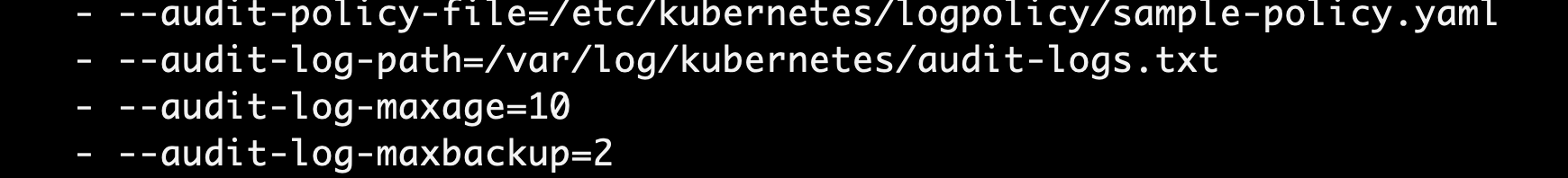
Don’t forget to apply the modified policy.

Simulation

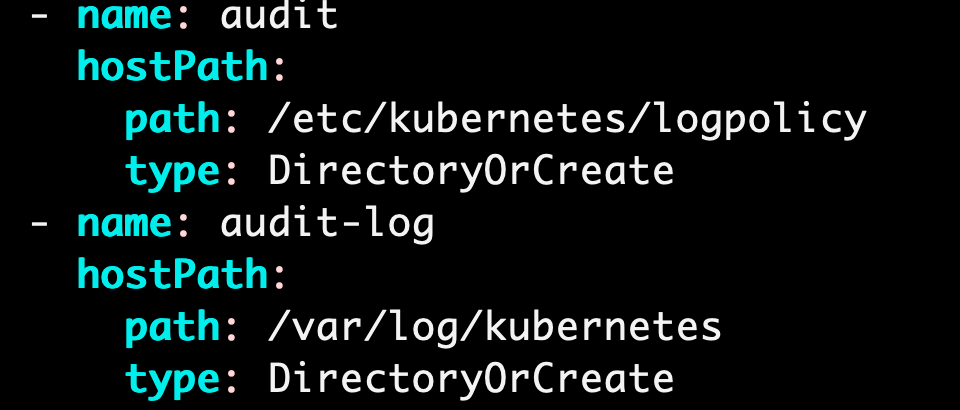
Take any sample policy.yaml file from kubernetes documentation.

Actual Answer

Add these under spesc:containers:volumeMounts of kuber-apiserver manifest.yaml file

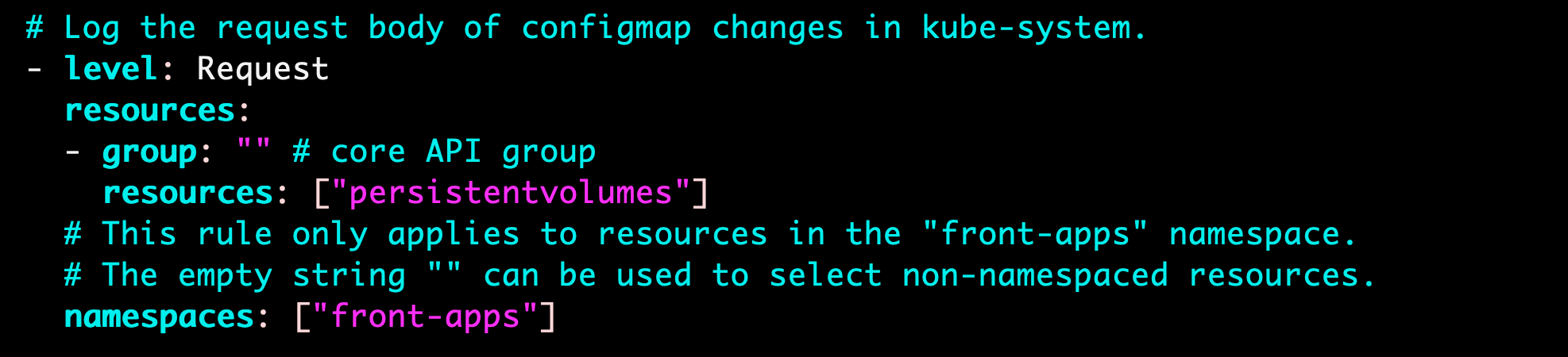


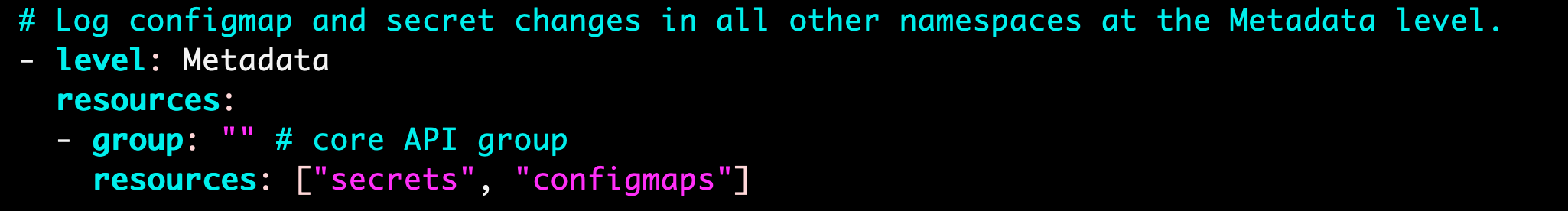
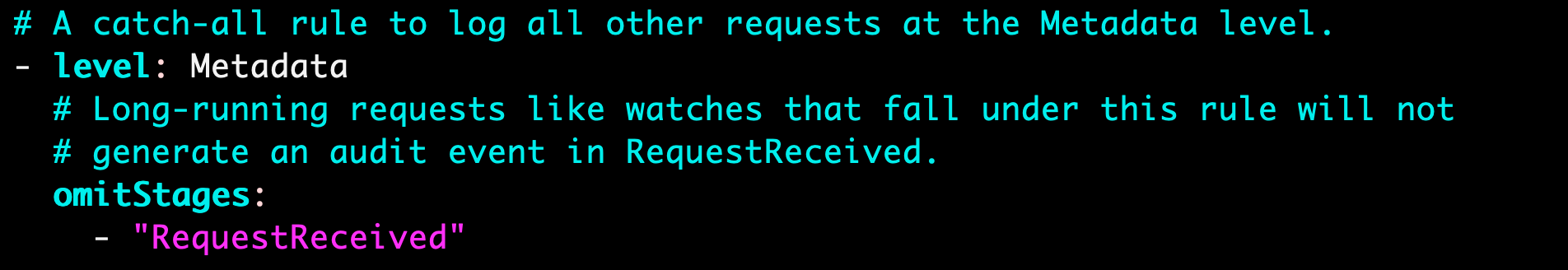
Add these under volumes



Edit the sample policy.yaml file and follow below steps





Make sure the api-server is up and running by checking

ps aux | grep -i apiserver

check the contents of this file cat /var/log/kubernetes/audit-logs.txt to ensure logs are captured.

**Q7 4%**

**Task**

Retrieve the content of the existing secret named db1-test in the istio-system namespace.

Store the username field in a file named /home/profile/user.txt and the password field in a file named /home/profile/pass.txt.

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

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

Create a new secret named db2-test in the isito-system namespace, with the following content:

username: production-instance

password: KvLfKgs4aH

Finally, create a new Pod that has access to the secret db2-test via a volume:

Pod name secret-pod

Namespace istio-system

Container name dev-container

Image nginx

Volume name secret-volume

Mount path /etc/secret

**Simulation of Question**

**Actual Answer**

**Q8 6%**

**Task**

Analyze and edit the given Dockerfile (based on the ubuntu:16.04 image) /home/profile/KSSC00301/Dockerfile fixing two instructions present in the file being prominent security/best-practice issues.

Analyze and edit the given manifest file /home/profile/KSSC00301/deployment.yaml 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, 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.

**Q09 11%**

**Context**

This cluster users containerd as CRI runtime. Containerd’s default runtime handler is runc. Containerd has been prepared to support an additional runtime handler, runsc (gVisor).

**Task**

Create a RuntimeClass named untrusted using the prepared runtime handler named runsc.

Update all Pods in the namespace server to run on gVisor.

*You can find a skeleton manifest file at /home/candidate/ksmv00301/runtime-class.yaml*

**Simulation of Question**

**Actual Answer**

Step 1: vim runtime.yaml while copying the content from /home/candidate/ksmv00301/runtime-class.yaml

Step 2: change the name to untrusted and handler to runsc under metadata

Step 3: k apply -f runtime.yaml

Step 4: k get deployment -n server

Step 5: k edit deployment <deployment name> -n server

Step 6: add runtimeClassName: untrusted under spec column above container line

Step 7: k rollout restart deployment <deployment-name>

Step 8: k get deployment <deployment name> -o yaml | grep -i untrusted

Step 9: k get pods -n <namespace name> -o yaml | grep -i runtimeClassName

**Q10 4%**

**Context**

It is best-practice to design containers to be stateless and immutable.

**Task**

Inspect Pods running in namespace production and delete any Pod that is either **not stateless** and **not immutable**.

Use the following strict interpretation of stateless and immutable:

* Pods being able to store data inside containers must be treated as **not stateless**.

*You don’t have to worry whether data is actually stored inside containers or not already.*

* Pods being configured to be privileged in any way must be treated as potentially **not stateless** and **not immutable**.

**Actual Answer**

Step 1: k get pods -n production -o yaml | grep -i privileged

Step 2: k get pods <pod name> -n production -o yaml | grep -i securityContext -A 5

Step 3: k delete pod <pod-name> -n production --grace-period=0 --force

**Q11 6%**

**Task**

Create a NetworkPolicy named pod-restricted to restrict access to Pod catelog-service running in namespace agile-team.

Only allow the following Pods to connect to Pod catelog-service:

* Pods in the namespace testing
* Pods with label environment preprod, in any namespace

Make sure to apply the NetworkPolicy.

*Refer to any skeleton manifest file of network-policy.yaml from K8s document.*

**Actual Answer**

**k get pod <pod-name> -n agile-team --show-labels**

**k get ns <namespace name> --show-labels**

**k get netpol -n <namespace name>**

**Q12 11%**

**Context**

A container 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 are files have been prepared and placed.*

Given an incomplete configuration in directory /etc/kubernetes/epconfig and a functional container image scanner with HTTPS endpoint <https://acme.local:8082/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 configurations to point to the provided

Finally, test if the configuration is working by trying to deploy the vulnerable resource /root/worker-node-1/vunerable-mainfest.yaml.

You can find the container image scanner’s log file at /var/log/imagepolicy/roadrunner.log.

**Actual Answer**

**Q13 6%**

**Task**

Use runtime detection tools to detect anomalous processes spawning and executing frequently in the single container belonging to Pod tomcat.

Two tools are available to use:

* sysdig
* falco

*The tools are pre-installed on the cluster’s worker node only; they are not available on the base system or the master node.*

Using the tool of your choice (including any non pre-installed tool), analyse the container’s behavior for at least 30 seconds, using filters that detect newly spawning and executing processes.

Store an incident file at /opt/KSR00101/incidents/summary, containing the detected incidents, one per line, in the following format:

[timestamp],[uid],[processName]

*Keep the tool’s original timestamp-format as-is.*

*Make sure to store the incident file on the cluster’s worker node.*

**Q14 11%**

**Context**

AppArmor is enabled on the cluster’s worker node. An Apparmor profile is prepared, but not enforced yet.

**Task**

On the cluster’s worker node, enforce the prepared AppArmor profile located at /etc/apparmor.d/nginx\_apparmor.

Edit the prepared manifest file located at /home/student/KSSH04401/nginx-deploy.yaml to apply the AppArmor profile.

Finally, apply the mainfest file and create the Pod specified in it.

**Actual Answer**

Step 1: ssh <worker-node-name>

Step 2: cat /sys/module/apparmor/parameters/enabled – output should be Y

Step 3: aa-status | grep <profile-name> - output should be nil

Step 4: cd /etc/apparmor.d/nginx\_apparmor – This location need to

Step 5: sudo apparmor\_parser <profile-name>

Step 6: aa-status | grep <profile-name>

Step 7: vim /home/student/KSSH04401/nginx-deploy.yaml

Step 8: add annotations under metadata section after labels and name

annotations:

container.apparmor.security.beta.kubernetes.io/<container-name>: localhost/<profile-name>

Step 10: k apply -f /home/student/KSSH04401/nginx-deploy.yaml

Step 11: k get po 🡪 pods should be in running state

Step 12: exit

**Q15 3%**

You may use your browser to open one additional tab to access Trivy’s documentation.

**Task**

Use the Trivy open-source container scanner to detect images with severe vulnerabilities used by pods in the namespace kamino.

Look for images with High and Critical severity vulnerabilities, and delete the pods that use those images.

*Trivy is pre-installed on the cluster’s master node only; it is not available on the base system or the worker nodes. You’ll have to connect to the cluster’s master node to use Trivy.*

**Actual Answer**

Step 1: trivy image <pod-image name>

Step 2: k delete po -n kamino for High and Crtical serverity vulnerabilities