You can access this page also inside the Remote Desktop by using the icons on the desktop

- Score
- Questions and Answers
- Preview Questions and Answers
- Exam Tips

CKS Simulator Kubernetes 1.31

https://killer.sh

Each question needs to be solved on a specific instance other than your main `candidate@terminal`. You'll need to connect to the correct instance via ssh, the command is provided before each question. To connect to a different instance you always need to return first to your main terminal by running the `exit` command, from there you can connect to a different one.

In the real exam each question will be solved on a different instance whereas in the simulator multiple questions will be solved on same instances.

Use `sudo -i` to become root on any node in case necessary.

Question 1 | Contexts

Solve this question on: `ssh cks3477`

You have access to multiple clusters from your main terminal through `kubectl` contexts. Write all context names into `/opt/course/1/contexts` on `cks3477`, one per line.

From the kubeconfig extract the certificate of user `restricted@infra-prod` and write it decoded to `/opt/course/1/cert`.

Answer:

Maybe the fastest way is just to run:

```
➜ ssh cks3477

➜ candidate@cks3477:~$ k config get-contexts # copy by hand

➜ candidate@cks3477:~$ k config get-contexts -o name > /opt/course/1/contexts
```

Or using jsonpath:

```
k config view -o jsonpath="{.contexts[*].name}"
k config view -o jsonpath="{.contexts[*].name}" | tr " " "\n" # new lines
k config view -o jsonpath="{.contexts[*].name}" | tr " " "\n" > /opt/course/1/contexts
```

The content could then look like:

```
# cks3477:/opt/course/1/contexts
gianna@infra-prod
infra-prod
restricted@infra-prod
```

For the certificate we could just run

```
k config view --raw
```

And copy it manually. Or we do:

```
k config view --raw -ojsonpath="{.users[2].user.client-certificate-data}" | base64 -d > /opt/course/1/cert
```

Or even:

```
k config view --raw -ojsonpath="{.users[?(.name == 'restricted@infra-prod')].user.client-certificate-data}" | base64 -d
> /opt/course/1/cert
```

```
# cks3477:/opt/course/1/cert
-----BEGIN CERTIFICATE-----
MIIDHzCCAgegAwIBAgIQN5Qe/Rj/PhaqckEI23LPnjANBgkqhkiG9w0BAQsFADAV
MRMwEQYDVQQDEwprdWJlcm5ldGVzMB4XDTIwMDkyNjIwNTUwNFoXDTIxMDkyNjIw
```

```
NTUwNFowKjETMBEGA1UEChMKcmVzdHJpY3RlZDETMBEGA1UEAxMKcmVzdHJpY3Rl
ZDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAL/Jaf/QQdijyJTWIDij
qa5p4oAh+xDBX3jR9R0G5DkmPU/FgXjxej3rTwHJbuxg7qjTuqQbf9Fb2AHcVtwH
gUjC12ODUDE+nVtap+hCe8OLHZwH7BGFWWscgInZOZW2IATK/YdqyQL5OKpQpFkx
iAknVZmPa2DTZ8FoyRESboFSTZj6y+JVA7ot0pM09jnxswstal9GZLeqioqfFGY6
YBO/Dg4DDsbKhqfUwJVT6Ur3ELsktZIMTRS5By4Xz18798eBiFAHvgJGq1TTwuPM
EhBfwYwgYbalL8DSHeFrelLBKgciwUKjr1lolnnuc1vhkX1peV1J3xrf6o2KkyMc
lY0CAwEAAaNWMFQwDgYDVR0PAQH/BAQDAgWgMBMGA1UdJQQMMAoGCCsGAQUFBwMC
MAwGA1UdEwEB/wQCMAAwHwYDVR0jBBgwFoAUPrspZIWR7YMN8vT5DF3s/LvpxPQw
DQYJKoZIhvcNAQELBQADggEBADq0Zt77gXI1s+uW46zBw4mIWgA1BLl2QqCuwmV
kd86eH5bD0FCtWlb6vGdcKPdFccHh8Z6z2LjjLu6UoiGUdIJaALhbYNJiXXi/7cf
M7sqNOxpxQ5X5hyvOBYD1W7d/EzPHV/lcbXPUDYFHNqBYs842LWSTlPQioDpupXp
FFUQPxsenNXDa4TbmaRvnK2jka0yXcqdiXuIteZZovp/IgNkfmx2Ld4/Q+Xlnscf
CFtWbjRa/0W/3EW/ghQ7xtC7bgcOHJesoiTZPCZ+dfKuUfH6d1qxgj6Jwt0HtyEf
QTQSc66BdMLnw5DMObs4lXDo2YE6LvMrySdXm/S7img5YzU=
-----END CERTIFICATE-----
```


Completed.

Question 2 | Runtime Security with Falco

Solve this question on: `ssh cks7262`

Falco is installed on worker node `cks7262-node1`. Connect using `ssh cks7262-node1` from `cks7262`. There is file `/etc/falco/rules.d/falco_custom.yaml` with rules that help you to:

1. Find a *Pod* running image `httpd` which modifies `/etc/passwd`.

   Scale the *Deployment* that controls that *Pod* down to 0.

2. Find a *Pod* running image `nginx` which triggers rule `Package management process launched`.

   Change the rule log text after `Package management process launched` to only include:

   ```
   time-with-nanosconds,container-id,container-name,user-name
   ```

   Collect the logs for at least 20 seconds and save them under `/opt/course/2/falco.log` on `cks7262`.

   Scale the *Deployment* that controls that *Pod* down to 0.

> ℹ️ Use `sudo -i` to become root which may be required for this question

Answer:

> ℹ️ Other tools you might have to be familar with are sysdig or tracee

Check out Falco files

First we can investigate Falco config a little:

```
➜ ssh cks7262

➜ candidate@cks7262:~$ ssh cks7262-node1

➜ candidate@cks7262-node1:~$ sudo -i

➜ root@cks7262-node1:~# cd /etc/falco

➜ root@cks7262-node1:/etc/falco# ls -lh
total 132K
drwxr-xr-x 2 root root 4.0K Aug 19 13:18 config.d
-rw-r--r-- 1 root root  53K Sep  7 10:04 falco.yaml
-rw-r--r-- 1 root root   21 Aug 19 12:57 falco_rules.local.yaml
-rw-r--r-- 1 root root  63K Jan  1  1970 falco_rules.yaml
drwxr-xr-x 2 root root 4.0K Aug 19 13:18 rules.d

➜ root@cks7262-node1:/etc/falco# ls -lh rules.d
total 4.0K
-rw-r--r-- 1 root root 1.2K Sep  7 12:24 falco_custom.yaml
```

Here we see the Falco rule file `falco_custom.yaml` mentioned in the question text. We can also see the Falco configuration in `falco.yaml`:

```yaml
# /etc/falco/falco.yaml

...
# With Falco 0.36 and beyond, it's now possible to apply multiple rules that match
# the same event type, eliminating concerns about rule prioritization based on the
# "first match wins" principle. However, enabling the `all` matching option may result
# in a performance penalty. We recommend carefully testing this alternative setting
# before deploying it in production. Read more under the `rule_matching` configuration.
rules_files:
  - /etc/falco/falco_rules.yaml
  - /etc/falco/falco_rules.local.yaml
  - /etc/falco/rules.d
...
```

This means that Falco is checking these directories for rules. There is also `falco_rules.local.yaml` in which we can override existing default rules. This is a much cleaner solution for production. Choose the faster way for you in the exam if nothing is specified in the task.

Step 1

We can run Falco and filter for certain output:

```
➜ root@cks7262-node1:~# falco -U | grep httpd
Sat Sep  7 12:39:04 2024: Falco version: 0.38.2 (x86_64)
Sat Sep  7 12:39:04 2024: Falco initialized with configuration files:
Sat Sep  7 12:39:04 2024:    /etc/falco/falco.yaml
Sat Sep  7 12:39:04 2024: System info: Linux version 6.8.0-41-generic (buildd@lcy02-amd64-100) (x86_64-linux-gnu-gcc-13
(Ubuntu 13.2.0-23ubuntu4) 13.2.0, GNU ld (GNU Binutils for Ubuntu) 2.42) #41-Ubuntu SMP PREEMPT_DYNAMIC Fri Aug  2
20:41:06 UTC 2024
Sat Sep  7 12:39:04 2024: Loading rules from file /etc/falco/falco_rules.yaml
Sat Sep  7 12:39:04 2024: Loading rules from file /etc/falco/falco_rules.local.yaml
Sat Sep  7 12:39:04 2024: Loading rules from file /etc/falco/rules.d/falco_custom.yaml
Sat Sep  7 12:39:04 2024: The chosen syscall buffer dimension is: 8388608 bytes (8 MBs)
Sat Sep  7 12:39:04 2024: you required a buffer every '2' CPUs but there are only '1' online CPUs. Falco changed the
config to: one buffer every '1' CPUs
Sat Sep  7 12:39:04 2024: Starting health webserver with threadiness 1, listening on 0.0.0.0:8765
Sat Sep  7 12:39:04 2024: Loaded event sources: syscall
Sat Sep  7 12:39:04 2024: Enabled event sources: syscall
Sat Sep  7 12:39:04 2024: Opening 'syscall' source with modern BPF probe.
Sat Sep  7 12:39:04 2024: One ring buffer every '1' CPUs.
12:58:32.430165207: Warning Sensitive file opened for reading by non-trusted program (file=/etc/passwd
gparent=containerd-shim ggparent=systemd gggparent=<NA> evt_type=open user=root user_uid=0 user_loginuid=-1 process=sed
proc_exepath=/bin/busybox parent=sh command=sed -i $d /etc/passwd terminal=0 container_id=f86cd629e71c
container_name=httpd)
...
```

It can take a bit till Falco displays output, use `falco -U/--unbuffered` to speed up

We can see a matching log. Next we can find the belonging *Pod* and scale down the *Deployment*:

```
➜ root@cks7262-node1:~# crictl ps -id f86cd629e71c
CONTAINER ID    IMAGE          NAME   ...   POD ID          POD
f86cd629e71c4   f6b40f9f8ad71  httpd  ...   cab6dafd045d5   rating-service-5c8f54bd77-bgkh6
```

Using the Pod ID we can find out more information like the *Namespace*:

```
➜ root@cks7262-node1:~# crictl pods -id cab6dafd045d5
POD ID          CREATED       ...   NAME                               NAMESPACE    ...
cab6dafd045d5   3 hours ago   ...   rating-service-5c8f54bd77-bgkh6    team-purple  ...
```

Now we can scale down:

```
➜ root@cks7262-node1:~# k get pod -A | grep rating-service
team-purple     rating-service-5c8f54bd77-bgkh6             1/1     Running     0    ...

➜ root@cks7262-node1:~# k -n team-purple scale deploy rating-service --replicas 0
deployment.apps/rating-service scaled
```

Step 1: Rule Investigation

If we have a look in file `/etc/falco/rules.d/falco_custom.yaml` then we see:

```yaml
# cks7262-node1:/etc/falco/rules.d/falco_custom.yaml
- list: sensitive_file_names
  items: [/etc/shadow, /etc/sudoers, /etc/pam.conf, /etc/security/pwquality.conf, /etc/passwd]
...
```

This is a list that overwrites the default list in `falco_rules.yaml`. It's used for example by `macro: sensitive_files`. To find the rule we could simply search for `Sensitive file opened for reading by non-trusted program` in `falco_rules.yaml`.

If we would like to trigger the rule with additional files/paths we could simply add these to `list: sensitive_file_names`.

Step 2

We run Falco and filter for certain output:

```
➜ root@cks7262-node1:~# falco -U | grep 'Package management process launched'
Sat Sep  7 13:10:43 2024: Falco version: 0.38.2 (x86_64)
Sat Sep  7 13:10:43 2024: Falco initialized with configuration files:
Sat Sep  7 13:10:43 2024:    /etc/falco/falco.yaml
Sat Sep  7 13:10:43 2024: System info: Linux version 6.8.0-41-generic (buildd@lcy02-amd64-100) (x86_64-linux-gnu-gcc-13 (Ubuntu 13.2.0-23ubuntu4) 13.2.0, GNU ld (GNU Binutils for Ubuntu) 2.42) #41-Ubuntu SMP PREEMPT_DYNAMIC Fri Aug  2 20:41:06 UTC 2024
Sat Sep  7 13:10:43 2024: Loading rules from file /etc/falco/falco_rules.yaml
Sat Sep  7 13:10:43 2024: Loading rules from file /etc/falco/falco_rules.local.yaml
Sat Sep  7 13:10:43 2024: Loading rules from file /etc/falco/rules.d/falco_custom.yaml
Sat Sep  7 13:10:43 2024: The chosen syscall buffer dimension is: 8388608 bytes (8 MBs)
Sat Sep  7 13:10:43 2024: you required a buffer every '2' CPUs but there are only '1' online CPUs. Falco changed the config to: one buffer every '1' CPUs
Sat Sep  7 13:10:43 2024: Starting health webserver with threadiness 1, listening on 0.0.0.0:8765
Sat Sep  7 13:10:43 2024: Loaded event sources: syscall
Sat Sep  7 13:10:43 2024: Enabled event sources: syscall
Sat Sep  7 13:10:43 2024: Opening 'syscall' source with modern BPF probe.
Sat Sep  7 13:10:43 2024: One ring buffer every '1' CPUs.
13:10:46.307338039: Error Package management process launched (user=root user_loginuid=-1 command=apk container_id=65338e61dc48 container_name=nginx image=docker.io/library/nginx:1.19.2-alpine)
...
```

> It can take a bit till Falco displays output, use `falco -U/--unbuffered` to speed up

We can see a matching log. Next we can find the belonging *Pod*:

```
➜ root@cks7262-node1:~# crictl ps -id 65338e61dc48
CONTAINER ID    IMAGE          NAME  ...   POD ID         POD
65338e61dc485   6f715d38cfe0e  nginx ...   1e3d3ea3e06ee  webapi-5499fdc5db-k4c7c
```

Using the Pod ID we can find out more information like the *Namespace*:

```
➜ root@cks7262-node1:~# crictl pods -id 1e3d3ea3e06ee
POD ID          CREATED       ...   NAME                         NAMESPACE    ...
1e3d3ea3e06ee   3 hours ago   ...   webapi-5499fdc5db-k4c7c      team-blue    ...
```

We wait before scaling down because this task requires some more steps before.

Step 2: Update Rule

The task requires us to store logs for rule `Package management process launched` with data `time,container-id,container-name,user-name`. So we edit the rule in `/etc/falco/rules.d/falco_custom.yaml`:

```
➜ root@cks7262-node1:/etc/falco# vim rules.d/falco_custom.yaml
```

```yaml
# cks7262-node1:/etc/falco/rules.d/falco_custom.yaml

...

# Container is supposed to be immutable. Package management should be done in building the image.
- rule: Launch Package Management Process in Container
  desc: Package management process ran inside container
  condition: >
    spawned_process
    and container
    and user.name != "_apt"
    and package_mgmt_procs
    and not package_mgmt_ancestor_procs
  output: >
    Package management process launched (user=%user.name user_loginuid=%user.loginuid
    command=%proc.cmdline container_id=%container.id container_name=%container.name
image=%container.image.repository:%container.image.tag)
  priority: ERROR
  tags: [process, mitre_persistence]
```

We change the above rule to:

```yaml
# cks7262-node1:/etc/falco/rules.d/falco_custom.yaml

...

# Container is supposed to be immutable. Package management should be done in building the image.
- rule: Launch Package Management Process in Container
```

```
  desc: Package management process ran inside container
  condition: >
    spawned_process
    and container
    and user.name != "_apt"
    and package_mgmt_procs
    and not package_mgmt_ancestor_procs
  output: >
    Package management process launched %evt.time,%container.id,%container.name,%user.name
  priority: ERROR
  tags: [process, mitre_persistence]
```

For all available fields we can check https://falco.org/docs/rules/supported-fields, which should be allowed to open during the exam. We can also run for example `falco --list | grep user` to find available fields.

Step 2: Collect logs

Next we check the logs in our adjusted format:

```
➜ root@cks7262-node1:~# falco -U | grep 'Package management process launched'
Sat Sep  7 13:31:20 2024: Falco version: 0.38.2 (x86_64)
...0.0.0.0:8765
Sat Sep  7 13:31:20 2024: Loaded event sources: syscall
Sat Sep  7 13:31:20 2024: Enabled event sources: syscall
Sat Sep  7 13:31:20 2024: Opening 'syscall' source with modern BPF probe.
Sat Sep  7 13:31:20 2024: One ring buffer every '1' CPUs.
13:31:26.364958758: Error Package management process launched 13:31:26.364958758,65338e61dc48,nginx,root
13:31:31.356117694: Error Package management process launched 13:31:31.356117694,65338e61dc48,nginx,root
13:31:36.329307852: Error Package management process launched 13:31:36.329307852,65338e61dc48,nginx,root
...
```

If there are syntax or other errors in the `falco_custom.yaml` then Falco will display these and we would need to adjust.

Now we can collect for at least 20 seconds. Copy&paste the output into file `/opt/course/2/falco.log` **on cks7262**:

```
➜ root@cks7262-node1:~# exit
logout

➜ candidate@cks7262-node1:~$ exit
logout
Connection to cks7262-node1 closed.

➜ candidate@cks7262:~$ vim /opt/course/2/falco.log
```

```
# cks7262:/opt/course/2/falco.log
13:31:26.364958758: Error Package management process launched 13:31:26.364958758,65338e61dc48,nginx,root
13:31:31.356117694: Error Package management process launched 13:31:31.356117694,65338e61dc48,nginx,root
13:31:36.329307852: Error Package management process launched 13:31:36.329307852,65338e61dc48,nginx,root
13:31:41.338988597: Error Package management process launched 13:31:41.338988597,65338e61dc48,nginx,root
13:31:46.329154755: Error Package management process launched 13:31:46.329154755,65338e61dc48,nginx,root
13:31:51.308124986: Error Package management process launched 13:31:51.308124986,65338e61dc48,nginx,root
13:31:56.358522188: Error Package management process launched 13:31:56.358522188,65338e61dc48,nginx,root
13:32:01.360834976: Error Package management process launched 13:32:01.360834976,65338e61dc48,nginx,root
13:32:06.327657274: Error Package management process launched 13:32:06.327657274,65338e61dc48,nginx,root
13:32:11.342534392: Error Package management process launched 13:32:11.342534392,65338e61dc48,nginx,root
13:32:16.343746448: Error Package management process launched 13:32:16.343746448,65338e61dc48,nginx,root
13:32:21.303524240: Error Package management process launched 13:32:21.303524240,65338e61dc48,nginx,root
13:32:26.330027622: Error Package management process launched 13:32:26.330027622,65338e61dc48,nginx,root
13:32:31.364716844: Error Package management process launched 13:32:31.364716844,65338e61dc48,nginx,root
```

Step 2: Scale down Deployment

Now we can scale down using the information we got at the beginning of step (2):

```
➜ candidate@cks7262:~# k get pod -A | grep webapi
team-blue       webapi-5499fdc5db-k4c7c                     1/1     Running      ...

➜ candidate@cks7262:~$ k -n team-blue scale deploy webapi --replicas 0
deployment.apps/webapi scaled
```

You should be comfortable finding, creating and editing Falco rules.

Question 3 | Apiserver Security

Solve this question on: `ssh cks7262`

You received a list from the DevSecOps team which performed a security investigation of the cluster. The list states the following about the apiserver setup:

- Accessible through a NodePort *Service*

Change the apiserver setup so that:

- Only accessible through a ClusterIP *Service*

> Use `sudo -i` to become root which may be required for this question

Answer:

In order to modify the parameters for the apiserver, we first ssh into the controlplane node and check which parameters the apiserver process is running with:

```
➜ ssh cks7262

➜ candidate@cks7262:~# sudo -i

➜ root@cks7262:~# ps aux | grep kube-apiserver
root       27622  7.4 15.3 1105924 311788 ?      Ssl  10:31  11:03 kube-apiserver --advertise-address=192.168.100.11 --
allow-privileged=true --authorization-mode=Node,RBAC --client-ca-file=/etc/kubernetes/pki/ca.crt --enable-admission-
plugins=NodeRestriction --enable-bootstrap-token-auth=true --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt --etcd-
certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key --
etcd-servers=https://127.0.0.1:2379 --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt --
kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key --kubelet-preferred-address-
types=InternalIP,ExternalIP,Hostname --kubernetes-service-node-port=31000 --proxy-client-cert-
...
```

We may notice the following argument:

```
--kubernetes-service-node-port=31000
```

We can also check the *Service* and see it's of type NodePort:

```
➜ root@cks7262:~# k get svc
NAME         TYPE       CLUSTER-IP   EXTERNAL-IP   PORT(S)         AGE
kubernetes   NodePort   10.96.0.1    <none>        443:31000/TCP   5d2h
```

The apiserver runs as a static *Pod*, so we can edit the manifest. But before we do this we also create a copy in case we mess things up:

```
➜ root@cks7262:~# cp /etc/kubernetes/manifests/kube-apiserver.yaml ~/3_kube-apiserver.yaml

➜ root@cks7262:~# vim /etc/kubernetes/manifests/kube-apiserver.yaml
```

We should remove the unsecure settings:

```
# /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 192.168.100.11:6443
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.100.11
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
#    - --kubernetes-service-node-port=31000   # delete or set to 0
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
```

```
...
```

Wait for the apiserver container to restart:

```
➜ root@cks7262:~# watch crictl ps
```

Give the apiserver some time to start up again. Check the apiserver's *Pod* status and the process parameters:

```
➜ root@cks7262:~# k -n kube-system get pod | grep apiserver
kube-apiserver-cks7262            1/1     Running        0          38s

➜ root@cks7262:~# ps aux | grep kube-apiserver | grep node-port
```

The apiserver got restarted without the unsecure settings. However, the *Service* `kubernetes` will still be of type NodePort:

```
➜ root@cks7262:~# k get svc
NAME         TYPE       CLUSTER-IP   EXTERNAL-IP   PORT(S)         AGE
kubernetes   NodePort   10.96.0.1    <none>        443:31000/TCP   5d3h
```

We need to delete the *Service* for the changes to take effect:

```
➜ root@cks7262:~# k delete svc kubernetes
service "kubernetes" deleted
```

After a few seconds:

```
➜ root@cks7262:~# k get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   6s
```

This should satisfy the DevSecOps team.

Question 4 | Pod Security Standard

Solve this question on: `ssh cks7262`

There is *Deployment* `container-host-hacker` in *Namespace* `team-red` which mounts `/run/containerd` as a hostPath volume on the *Node* where it's running. This means that the *Pod* can access various data about other containers running on the same *Node*.

To prevent this configure *Namespace* `team-red` to `enforce` the `baseline` Pod Security Standard. Once completed, delete the *Pod* of the *Deployment* mentioned above.

Check the *ReplicaSet* events and write the event/log lines containing the reason why the *Pod* isn't recreated into `/opt/course/4/logs` on `cks7262`.

Answer:

Making *Namespaces* use Pod Security Standards works via labels. We can simply edit it:

```
➜ ssh cks7262

➜ candidate@cks7262:~# k edit ns team-red
```

Now we configure the requested label:

```
# kubectl edit namespace team-red
apiVersion: v1
kind: Namespace
metadata:
  labels:
    kubernetes.io/metadata.name: team-red
    pod-security.kubernetes.io/enforce: baseline # add
  name: team-red
...
```

This should already be enough for the default Pod Security Admission Controller to pick up on that change. Let's test it and delete the *Pod* to see if it'll be recreated or fails, it should fail!

```
➜ candidate@cks7262:~# k -n team-red get pod
NAME                                    READY   STATUS    RESTARTS   AGE
container-host-hacker-dbf989777-wm8fc   1/1     Running   0          115s

➜ candidate@cks7262:~# k -n team-red delete pod container-host-hacker-dbf989777-wm8fc --force --grace-period 0
pod "container-host-hacker-dbf989777-wm8fc" deleted

➜ candidate@cks7262:~# k -n team-red get pod
No resources found in team-red namespace.
```

Usually the *ReplicaSet* of a *Deployment* would recreate the *Pod* if deleted, here we see this doesn't happen. Let's check why:

```
➜ candidate@cks7262:~# k -n team-red get rs
NAME                              DESIRED   CURRENT   READY   AGE
container-host-hacker-dbf989777   1         0         0       5m25s

➜ candidate@cks7262:~# k -n team-red describe rs container-host-hacker-dbf989777
Name:           container-host-hacker-dbf989777
Namespace:      team-red
...
Events:
  Type     Reason            Age                   From                   Message
  ----     ------            ----                  ----                   -------
...
  Warning  FailedCreate  78s                replicaset-controller  Error creating: pods "container-host-hacker-dbf989777-x5v5t" is forbidden: violates PodSecurity "baseline:latest": hostPath volumes (volume "containerdata")
  Warning  FailedCreate  39s (x7 over 77s)  replicaset-controller  (combined from similar events): Error creating: pods "container-host-hacker-dbf989777-64q6p" is forbidden: violates PodSecurity "baseline:latest": hostPath volumes (volume "containerdata")
```

There we go! Finally we write the reason into the requested file so that scoring will be happy too!

```
# cks7262:/opt/course/4/logs
Warning  FailedCreate      2m2s (x9 over 2m40s)  replicaset-controller  (combined from similar events): Error creating: pods "container-host-hacker-dbf989777-kjfpn" is forbidden: violates PodSecurity "baseline:latest": hostPath volumes (volume "containerdata")
```

Pod Security Standards can give a great base level of security! But when one finds themselves wanting to deeper adjust the levels like `baseline` or `restricted`... this isn't possible and 3rd party solutions like OPA or Kyverno could be looked at.

Question 5 | CIS Benchmark

Solve this question on: `ssh cks3477`

You're ask to evaluate specific settings of the cluster against the CIS Benchmark recommendations. Use the tool `kube-bench` which is already installed on the nodes.

Connect to the worker node using `ssh cks3477-node1` from `cks3477`.

On the controlplane node ensure (correct if necessary) that the CIS recommendations are set for:

1. The `--profiling` argument of the kube-controller-manager
2. The ownership of directory `/var/lib/etcd`

On the worker node ensure (correct if necessary) that the CIS recommendations are set for:

3. The permissions of the kubelet configuration `/var/lib/kubelet/config.yaml`
4. The `--client-ca-file` argument of the kubelet

> ℹ Use `sudo -i` to become root which may be required for this question

Answer:

Step 1

First we ssh into the controlplane node run `kube-bench` against the controlplane components:

```
➜ ssh cks3477

➜ candidate@cks3477:~# sudo -i

➜ root@cks3477:~# kube-bench run --targets=master
...
```

```
== Summary master ==
38 checks PASS
10 checks FAIL
11 checks WARN
0 checks INFO

== Summary total ==
38 checks PASS
10 checks FAIL
11 checks WARN
0 checks INFO
```

We see some passes, fails and warnings. Let's check the required step (1) of the controller manager:

```
➜ root@cks3477:~# kube-bench run --targets=master | grep kube-controller -A 3
1.3.1 Edit the Controller Manager pod specification file /etc/kubernetes/manifests/kube-controller-manager.yaml
on the control plane node and set the --terminated-pod-gc-threshold to an appropriate threshold,
for example, --terminated-pod-gc-threshold=10

1.3.2 Edit the Controller Manager pod specification file /etc/kubernetes/manifests/kube-controller-manager.yaml
on the control plane node and set the below parameter.
--profiling=false
```

There we see 1.3.2 which suggests to set `--profiling=false`, we can check if it currently passes or fails:

```
➜ root@cks3477:~# kube-bench run --targets=master --check='1.3.2'
[INFO] 1 Control Plane Security Configuration
[INFO] 1.3 Controller Manager
[FAIL] 1.3.2 Ensure that the --profiling argument is set to false (Automated)
...
```

So to obey we do:

```
➜ root@cks3477:~# vim /etc/kubernetes/manifests/kube-controller-manager.yaml
```

Edit the corresponding line:

```yaml
# cks3477:/etc/kubernetes/manifests/kube-controller-manager.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-controller-manager
    tier: control-plane
  name: kube-controller-manager
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-controller-manager
    - --allocate-node-cidrs=true
    - --authentication-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --authorization-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --bind-address=127.0.0.1
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --cluster-cidr=10.244.0.0/16
    - --cluster-name=kubernetes
    - --cluster-signing-cert-file=/etc/kubernetes/pki/ca.crt
    - --cluster-signing-key-file=/etc/kubernetes/pki/ca.key
    - --controllers=*,bootstrapsigner,tokencleaner
    - --kubeconfig=/etc/kubernetes/controller-manager.conf
    - --leader-elect=true
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --root-ca-file=/etc/kubernetes/pki/ca.crt
    - --service-account-private-key-file=/etc/kubernetes/pki/sa.key
    - --service-cluster-ip-range=10.96.0.0/12
    - --use-service-account-credentials=true
    - --profiling=false            # add
...
```

We wait for the *Pod* to restart, then run `kube-bench` again to check if the problem was solved:

```
➜ root@cks3477:~# kube-bench run --targets=master | grep 1.3.2
[PASS] 1.3.2 Ensure that the --profiling argument is set to false (Automated)
```

Problem solved and 1.3.2 is passing:

Step 2

Next step is to check the ownership of directory `/var/lib/etcd`, so we first have a look:

```
➜ root@cks3477:~# ls -lh /var/lib | grep etcd
drwx------  3 root      root      4.0K Sep 11 20:08 etcd
```

Looks like user root and group root. Also possible to check using:

```
➜ root@cks3477:~# stat -c %U:%G /var/lib/etcd
root:root
```

But what has `kube-bench` to say about this?

```
➜ root@cks3477:~# kube-bench run --targets=master | grep "/var/lib/etcd" -B5
For example, chmod 600 <path/to/cni/files>

1.1.12 On the etcd server node, get the etcd data directory, passed as an argument --data-dir,
from the command 'ps -ef | grep etcd'.
Run the below command (based on the etcd data directory found above).
For example, chown etcd:etcd /var/lib/etcd

➜ root@cks3477:~# kube-bench run --targets=master | grep 1.1.12
[FAIL] 1.1.12 Ensure that the etcd data directory ownership is set to etcd:etcd (Automated)
1.1.12 On the etcd server node, get the etcd data directory, passed as an argument --data-dir,
```

To comply we run the following:

```
➜ root@cks3477:~# chown etcd:etcd /var/lib/etcd

➜ root@cks3477:~# ls -lh /var/lib | grep etcd
drwx------  3 etcd      etcd      4.0K Sep 11 20:08 etcd
```

This looks better. We run `kube-bench` again, and make sure test 1.1.12. is passing.

```
➜ root@cks3477:~# kube-bench run --targets=master | grep 1.1.12
[PASS] 1.1.12 Ensure that the etcd data directory ownership is set to etcd:etcd (Automated)
```

Done.

Step 3

To continue with step (3), we'll head to the worker node and ensure that the kubelet configuration file has the minimum necessary permissions as recommended:

```
➜ candidate@cks3477:~# ssh cks3477-node1

➜ candidate@cks3477-node1:~# sudo -i

➜ root@cks3477-node1:~# kube-bench run --targets=node
...
== Summary node ==
16 checks PASS
2 checks FAIL
6 checks WARN
0 checks INFO

== Summary total ==
16 checks PASS
2 checks FAIL
6 checks WARN
0 checks INFO
```

Also here some passes, fails and warnings. We check the permission level of the kubelet config file:

```
➜ root@cks3477-node1:~# stat -c %a /var/lib/kubelet/config.yaml
777
```

777 is highly permissive access level and not recommended by the `kube-bench` guidelines:

```
➜ root@cks3477-node1:~# kube-bench run --targets=node | grep /var/lib/kubelet/config.yaml -B2

4.1.9 Run the following command (using the config file location identified in the Audit step)
chmod 600 /var/lib/kubelet/config.yaml

➜ root@cks3477-node1:~# kube-bench run --targets=node | grep 4.1.9
[FAIL] 4.1.9 If the kubelet config.yaml configuration file is being used validate permissions set to 600 or more
restrictive (Automated)
4.1.9 Run the following command (using the config file location identified in the Audit step)
```

We obey and set the recommended permissions:

```
➜ root@cks3477-node1:~# chmod 600 /var/lib/kubelet/config.yaml

➜ root@cks3477-node1:~# stat -c %a /var/lib/kubelet/config.yaml
644
```

And check if test 4.1.9 is passing:

```
➜ root@cks3477-node1:~# kube-bench run --targets=node | grep 4.1.9
[PASS] 4.1.9 If the kubelet config.yaml configuration file is being used validate permissions set to 600 or more
restrictive (Automated)
```

Step 4

Finally for step (4), let's check whether `--client-ca-file` argument for the kubelet is set properly according to `kube-bench` recommendations:

```
➜ root@cks3477-node1:~# kube-bench run --targets=node | grep client-ca-file
[PASS] 4.2.3 Ensure that the --client-ca-file argument is set as appropriate (Automated)
```

This looks like 4.2.3 is passing.

To further investigate we run the following command to locate the kubelet config file, and open it:

```
➜ root@cks3477-node1:~# ps -ef | grep kubelet
root        6972       1  1 10:15 ?        00:06:26 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-
kubelet.conf --kubeconfig=/etc/kubernetes/kubele.conf --config=/var/lib/kubelet/config.yaml --container-runtime-
endpoint=unix:///var/run/containerd/containerd.sock --pod-infra-container-image=registry.k8s.io/pause:3.9

➜ root@croot@cks3477-node1:~# vim /var/lib/kubelet/config.yaml
```

```yaml
# /var/lib/kubelet/config.yaml
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
...
```

The `clientCAFile` points to the location of the certificate, which is correct.

Question 6 | Verify Platform Binaries

Solve this question on: `ssh cks3477`

There are four Kubernetes server binaries located at `/opt/course/6/binaries` on `cks3477`. You're provided with the following verified sha512 values for these:

kube-apiserver
`f417c0555bc0167355589dd1afe23be9bf909bf98312b1025f12015d1b58a1c62c9908c0067a7764fa35efdac7016a9efa8711a44425dd6692906a7c283f032c`

kube-controller-manager
`60100cc725e91fe1a949e1b2d0474237844b5862556e25c2c655a33boa8225855ec5ee22fa4927e6c46a60d43a7c4403a27268f96fbb726307d1608b44f38a60`

kube-proxy
`52f9d8ad045f8eee1d689619ef8ceef2d86d50c75a6a332653240d7ba5b2a114aca056d9e513984ade24358c9662714973c1960c62a5cb37dd375631c8a614c6`

kubelet
`4be40f2440619e990897cf956c32800dc96c2c983bf64519854a3309fa5aa21827991559f9c44595098e27e6f2ee4d64a3fdec6baba8a177881f20e3ec61e26c`

Delete those binaries that don't match with the sha512 values above.

Answer:

We check the directory:

```
➜ ssh cks3477

➜ candidate@cks3477:~# cd /opt/course/6/binaries

➜ candidate@cks3477:/opt/course/6/binaries$ ls
kube-apiserver  kube-controller-manager  kube-proxy  kubelet
```

To generate the sha512 sum of a binary we do:

```
➜ candidate@cks3477:/opt/course/6/binaries$ sha512sum kube-apiserver
f417c0555bc0167355589dd1afe23be9bf909bf98312b1025f12015d1b58a1c62c9908c0067a7764fa35efdac7016a9efa8711a44425dd6692906a7
c283f032c  kube-apiserver
```

Looking good, next:

```
➜ candidate@cks3477:/opt/course/6/binaries$ sha512sum kube-controller-manager
60100cc725e91fe1a949e1b2d0474237844b5862556e25c2c655a33b0a8225855ec5ee22fa4927e6c46a60d43a7c4403a27268f96fbb726307d1608
b44f38a60  kube-controller-manager
```

Okay, next:

```
➜ candidate@cks3477:/opt/course/6/binaries$ sha512sum kube-proxy
52f9d8ad045f8eee1d689619ef8ceef2d86d50c75a6a332653240d7ba5b2a114aca056d9e513984ade24358c9662714973c1960c62a5cb37dd37563
1c8a614c6  kube-proxy
```

Also good, and finally:

```
➜ candidate@cks3477:/opt/course/6/binaries$ sha512sum kubelet
7b720598e6a3483b45c537b57d759e3e82bc5c53b3274f681792f62e941019cde3d51a7f9b55158abf3810d506146bc0aa7cf97b36f27f341028a54
431b335be  kubelet
```

Catch! Binary `kubelet` has a different hash!

But did we actually compare everything properly before? Let's have a closer look at `kube-controller-manager` again:

```
➜ candidate@cks3477:/opt/course/6/binaries$ sha512sum kube-controller-manager > compare

➜ candidate@cks3477:/opt/course/6/binaries$ vim compare
```

Edit to only have the provided hash and the generated one in one line each:

```
# cks3477:/opt/course/6/binaries/compare
60100cc725e91fe1a949e1b2d0474237844b5862556e25c2c655a33b0a8225855ec5ee22fa4927e6c46a60d43a7c4403a27268f96fbb726307d1608
b44f38a60
60100cc725e91fe1a949e1b2d0474237844b5862556e25c2c655a33boa8225855ec5ee22fa4927e6c46a60d43a7c4403a27268f96fbb726307d1608
b44f38a60
```

Looks right at a first glance, but if we do:

```
➜ candidate@cks3477:/opt/course/6/binaries$ cat compare | uniq
60100cc725e91fe1a949e1b2d0474237844b5862556e25c2c655a33b0a8225855ec5ee22fa4927e6c46a60d43a7c4403a27268f96fbb726307d1608
b44f38a60
60100cc725e91fe1a949e1b2d0474237844b5862556e25c2c655a33boa8225855ec5ee22fa4927e6c46a60d43a7c4403a27268f96fbb726307d1608
b44f38a60
```

This shows they are different, by just one character actually.

We could also do a diff:

```
➜ candidate@cks3477:/opt/course/6/binaries$ sha512sum kube-controller-manager > compare1

➜ candidate@cks3477:/opt/course/6/binaries$ vim compare1 # REMOVE filename

➜ candidate@cks3477:/opt/course/6/binaries$ echo
60100cc725e91fe1a949e1b2d0474237844b5862556e25c2c655a33boa8225855ec5ee22fa4927e6c46a60d43a7c4403a27268f96fbb726307d1608
b44f38a60 > compare2

➜ candidate@cks3477:/opt/course/6/binaries$ diff compare1 compare2
1c1
<
60100cc725e91fe1a949e1b2d0474237844b5862556e25c2c655a33b0a8225855ec5ee22fa4927e6c46a60d43a7c4403a27268f96fbb726307d1608
b44f38a60
---
>
60100cc725e91fe1a949e1b2d0474237844b5862556e25c2c655a33boa8225855ec5ee22fa4927e6c46a60d43a7c4403a27268f96fbb726307d1608
b44f38a60
```

To complete the task we do:

```
➜ candidate@cks3477:/opt/course/6/binaries$ rm kubelet kube-controller-manager
```

Question 7 | KubeletConfiguration

Solve this question on: `ssh cks8930`

You're asked to update the cluster's `KubeletConfiguration`. Implement the following changes in the Kubeadm way that ensures new Nodes added to the cluster will receive the changes too:

1. Set `containerLogMaxSize` to `5Mi`

   Set `containerLogMaxFiles` to `3`
2. Apply the changes for the Kubelet on `cks8930`
3. Apply the changes for the Kubelet on `cks8930-node1`. Connect with `ssh cks8930-node1` from `cks8930`

> ℹ Use `sudo -i` to become root which may be required for this question

Answer:

Step 1: Update Kubelet-Config ConfigMap

A cluster created with Kubeadm will have a *ConfigMap* named `kubelet-config` in *Namespace* `kube-system`. This *ConfigMap* will be used if new *Nodes* are added to the cluster. There is information about that process in the docs.

Let's find that *ConfigMap* and perform the requested changes:

```
➜ ssh cks8930

➜ candidate@cks8930:~# k -n kube-system edit cm kubelet-config
```

```yaml
# kubectl -n kube-system edit cm kubelet-config
apiVersion: v1
data:
  kubelet: |
    apiVersion: kubelet.config.k8s.io/v1beta1
    kind: KubeletConfiguration
    ...
    volumeStatsAggPeriod: 0s
    containerLogMaxSize: 5Mi
    containerLogMaxFiles: 3
kind: ConfigMap
metadata:
  name: kubelet-config
  namespace: kube-system
...
```

Above we can see that we simply added the two new arguments to `data.kubelet`.

A new *Node* added to the cluster, both control plane and worker, would use this *KubeletConfiguration* containing the changes. That *KubeletConfiguration* from the *ConfigMap* will also be used during a `kubeadm upgrade`.

In the next steps we'll see that the Kubelet-Config of the control plane and worker node remain unchanged so far.

Step 2: Update Control Plane Kubelet-Config

To find the Kubelet-Config path we can check the Kubelet process:

```
➜ candidate@cks8930:~# sudo -i

➜ root@cks8930:~# ps aux | grep kubelet
root         7418  2.0  4.8 1927756 98748 ?       Ssl  11:38   1:56 /usr/bin/kubelet --bootstrap-
kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --
config=/var/lib/kubelet/config.yaml
...
```

Above we see it's specified via the argument `--config=/var/lib/kubelet/config.yaml`. We could also check the Kubeadm config for the Kubelet:

```
➜ root@cks8930:~# find / | grep kubeadm
/var/lib/dpkg/info/kubeadm.md5sums
/var/lib/dpkg/info/kubeadm.list
/var/lib/kubelet/kubeadm-flags.env
/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
...

➜ root@cks8930:~# cat /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --
kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
...
```

Above we see the argument `--config` being set. And we should see that our changes are still missing in that file:

```
➜ root@cks8930:~# grep containerLog /var/lib/kubelet/config.yaml

➜ root@cks8930:~#
```

We go ahead and download the latest Kubelet-Config, possible with `--dry-run` at first:

```
➜ root@cks8930:~# kubeadm upgrade node phase kubelet-config --dry-run
...

➜ root@cks8930:~# kubeadm upgrade node phase kubelet-config
[upgrade] Reading configuration from the cluster...
[upgrade] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[upgrade] Backing up kubelet config file to /etc/kubernetes/tmp/kubeadm-kubelet-config1186317096/config.yaml
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[upgrade] The configuration for this node was successfully updated!
[upgrade] Now you should go ahead and upgrade the kubelet package using your package manager.

➜ root@cks8930:~# grep containerLog /var/lib/kubelet/config.yaml
containerLogMaxFiles: 3
containerLogMaxSize: 5Mi
```

Sweet! Now we just need to restart the Kubelet:

```
➜ root@cks8930:~# service kubelet restart
```

(Optional) See the current Kubelet-Config of a Node

It is necessary to restart the Kubelet in order for updates in `/var/lib/kubelet/config.yaml` to take effect. We could verify this with (docs):

```
➜ root@cks8930:~# kubectl get --raw "/api/v1/nodes/cks8930/proxy/configz" | jq
...
    "containerLogMaxSize": "5Mi",
    "containerLogMaxFiles": 3,
...

➜ root@cks8930:~# kubectl get --raw "/api/v1/nodes/cks8930-node1/proxy/configz" | jq
...
    "containerLogMaxSize": "10Mi",
    "containerLogMaxFiles": 5,
...
```

For *Node* `cks8930-node1` the default values are still configured.

Step 3: Update Worker Node Kubelet-Config

We should see that the existing Kubelet-Config on the worker node is still unchanged:

```
➜ root@cks8930:~# ssh cks8930-node1

➜ root@cks8930-node1:~# grep containerLog /var/lib/kubelet/config.yaml

➜ root@cks8930-node1:~#
```

So we go ahead and apply the updates:

```
➜ root@cks8930-node1:~# kubeadm upgrade node phase kubelet-config
[upgrade] Reading configuration from the cluster...
[upgrade] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[upgrade] Backing up kubelet config file to /etc/kubernetes/tmp/kubeadm-kubelet-config948054586/config.yaml
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[upgrade] The configuration for this node was successfully updated!
[upgrade] Now you should go ahead and upgrade the kubelet package using your package manager.

➜ root@cks8930-node1:~# grep containerLog /var/lib/kubelet/config.yaml
containerLogMaxFiles: 3
containerLogMaxSize: 5Mi

➜ root@cks8930-node1:~# service kubelet restart
```

And optionally for admins with trust issues (or the ones that might forget to restart the Kubelets):

```
➜ root@cks8930-node1:~# kubectl get --raw "/api/v1/nodes/cks8930-node1/proxy/configz" | jq
...
    "containerLogMaxSize": "5Mi",
    "containerLogMaxFiles": 3,
...
```

Task completed.

Question 8 | CiliumNetworkPolicy

Solve this question on: `ssh cks7262`

In *Namespace* `team-orange` a Default-Allow strategy for all *Namespace*-internal traffic was chosen. There is an existing *CiliumNetworkPolicy* `default-allow` which assures this and which should not be altered. That policy also allows cluster internal DNS resolution.

Now it's time to deny and authenticate certain traffic. Create 3 *CiliumNetworkPolicies* in *Namespace* `team-orange` to implement the following requirements:

1. Create a `Layer 3` policy named `p1` to:

   Deny outgoing traffic from *Pods* with label `type=messenger` to *Pods* behind *Service* `database`

2. Create a `Layer 4` policy named `p2` to:

   Deny outgoing `ICMP` traffic from *Deployment* `transmitter` to *Pods* behind *Service* `database`

3. Create a `Layer 3` policy named `p3` to:

   Enable Mutual Authentication for outgoing traffic from *Pods* with label `type=database` to *Pods* with label `type=messenger`

> ℹ All *Pods* in the *Namespace* run plain Nginx images with open port 80. This allows simple connectivity tests like: `k -n team-orange exec POD_NAME -- curl database`

Answer:

A great way to inspect and learn writing *NetworkPolices* and *CiliumNetworkPolicies* is the [Network Policy Editor](), but it's not an allowed resource during the exam.

Overview

First we have a look at existing resources in *Namespace* `team-orange`:

```
➜ ssh cks7262

➜ candidate@cks7262:~$ k -n team-orange get pod --show-labels -owide
NAME                          ...     IP            ...   LABELS
database-0                    ...   10.244.2.13     ...   ...,type=database
messenger-57f557cd65-rhzd7    ...   10.244.1.126    ...   ...,type=messenger
messenger-57f557cd65-xcqwz    ...   10.244.2.70     ...   ...,type=messenger
transmitter-866696fc57-6ccgr ...    10.244.1.152    ...   ...,type=transmitter
transmitter-866696fc57-d8qk4 ...    10.244.2.214    ...   ...,type=transmitter

➜ candidate@cks7262:~$ k -n team-orange get svc,ep
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/database   ClusterIP   10.108.172.58   <none>        80/TCP    8m29s

NAME                      ENDPOINTS         AGE
```

```
endpoints/database   10.244.2.13:80   8m29s
```

These are the existing *Pods* and the *Service* we should work with. We can see that the `database` *Service* points to the `database-0` *Pod*. And this is the existing `default-allow` policy:

```yaml
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: default-allow
  namespace: team-orange
spec:
  endpointSelector:
    matchLabels: {}              # Apply this policy to all Pods in Namespace team-orange
  egress:
  - toEndpoints:
    - {}                         # ALLOW egress to all Pods in Namespace team-orange
  - toEndpoints:
      - matchLabels:
          io.kubernetes.pod.namespace: kube-system
          k8s-app: kube-dns
    toPorts:
      - ports:
          - port: "53"
            protocol: UDP
        rules:
          dns:
            - matchPattern: "*"
  ingress:
  - fromEndpoints:               # ALLOW ingress from all Pods in Namespaace team-orange
    - {}
```

CiliumNetworkPolicies behave like vanilla *NetworkPolicies*: once one egress rule exists, all other egress is forbidden. This is also the case for egressDeny rules: once one egressDeny rule exists, all other egress is also forbidden, unless allowed by an egress rule. This is why a Default-Allow policy like this one is necessary in this scenario. The behaviour explained above for egress is also the case for ingress.

Policy 1

Without any changes we check the connection from a `type=messenger` *Pod* to the *Service* `database`:

```
➜ candidate@cks7262:~$ k -n team-orange exec messenger-57f557cd65-rhzd7 -- curl -m 2 database
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
...
```

This works because of the K8s DNS resolution of the `database` *Service*, we should see the same result when using the *Service* IP:

```
➜ candidate@cks7262:~$ k -n team-orange exec messenger-57f557cd65-rhzd7 -- curl -m 2 --head 10.108.172.58
HTTP/1.1 200 OK
...
```

This works, we just used the `--head` for curl to only show the HTTP response code which should be sufficient. And same should work if we contact the `database-0` *Pod* IP directly:

```
➜ candidate@cks7262:~$ k -n team-orange exec messenger-57f557cd65-rhzd7 -- curl -m 2 --head 10.244.2.13
HTTP/1.1 200 OK
...
```

Connectivity works without restriction. Now we create a deny policy as requested:

```
➜ candidate@cks7262:~$ vim 8_p1.yaml
```

```yaml
# cks7262:~/8_p1.yaml
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: p1
  namespace: team-orange
spec:
  endpointSelector:
    matchLabels:
      type: messenger
  egressDeny:
  - toEndpoints:
```

```
    - matchLabels:
        type: database  # we use the label of the Pods behind the Service "database"
```

```
➜ candidate@cks7262:~$ k -f 8_p1.yaml apply
ciliumnetworkpolicy.cilium.io/p1 created

➜ candidate@cks7262:~$ k -n team-orange get cnp
NAME            AGE
default-allow   9m16s
p1              3s
```

Let's test connection to the *Service* by name and IP:

```
➜ candidate@cks7262:~$ k -n team-orange exec messenger-57f557cd65-rhzd7 -- curl -m 2 --head database
curl: (28) Resolving timed out after 2002 milliseconds
command terminated with exit code 28

➜ candidate@cks7262:~$ k -n team-orange exec messenger-57f557cd65-rhzd7 -- curl -m 2 --head 10.108.172.58
curl: (28) Connection timed out after 2002 milliseconds
command terminated with exit code 28
```

Connection timing out. And we test connection to the `database-0` *Pod* IP directly:

```
➜ candidate@cks7262:~$ k -n team-orange exec messenger-57f557cd65-rhzd7 -- curl -m 2 --head 10.244.2.13
curl: (28) Connection timed out after 2002 milliseconds
command terminated with exit code 28
```

Also timing out. But do other connections still work? We try to contact a `type=transmitter` *Pod*:

```
➜ candidate@cks7262:~$ k -n team-orange exec messenger-57f557cd65-rhzd7 -- curl -m 2 --head 10.244.1.152
HTTP/1.1 200 OK
...
```

Looks great.

Policy 2

Now we should prevent ICMP (Pings) from *Deployment* `transmitter` to *Pods* behind *Service* `database`. Before we do this we check that ICMP currently works:

```
➜ candidate@cks7262:~$ k -n team-orange get pod --show-labels -owide
NAME                          ...    IP            ...   LABELS
database-0                    ...   10.244.2.13    ...   ...,type=database
messenger-57f557cd65-rhzd7    ...   10.244.1.126   ...   ...,type=messenger
messenger-57f557cd65-xcqwz    ...   10.244.2.70    ...   ...,type=messenger
transmitter-866696fc57-6ccgr ...   10.244.1.152   ...   ...,type=transmitter
transmitter-866696fc57-d8qk4 ...   10.244.2.214   ...   ...,type=transmitter

➜ candidate@cks7262:~$ k -n team-orange get svc
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/database   ClusterIP   10.108.172.58   <none>        80/TCP    8m29s

➜ candidate@cks7262:~$ k -n team-orange exec manager-bd89c64cc-76lxk -- ping 10.244.2.13
PING 10.244.2.13 (10.244.2.13): 56 data bytes
64 bytes from 10.244.2.13: seq=0 ttl=63 time=2.555 ms
64 bytes from 10.244.2.13: seq=1 ttl=63 time=0.102 ms
...
```

Works. Now to restrict it:

```
➜ candidate@cks7262:~$ vim 8_p2.yaml
```

```
# cks7262:~/8_p2.yaml
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: p2
  namespace: team-orange
spec:
  endpointSelector:
    matchLabels:
      type: transmitter # we use the label of the Pods behind Deployment "transmitter"
  egressDeny:
  - toEndpoints:
    - matchLabels:
        type: database  # we use the label of the Pods behind the Service "database"
    icmps:
    - fields:
      - type: 8
        family: IPv4
      - type: EchoRequest
```

```
        family: IPv6
```

```
➜ candidate@cks7262:~$ k -f 8_p2.yaml apply
ciliumnetworkpolicy.cilium.io/p2 created

➜ candidate@cks7262:~$ k -n team-orange get cnp
NAME            AGE
default-allow   31m
p1              22m
p2              7s

➜ candidate@cks7262:~$ k -n team-orange exec transmitter-866696fc57-6ccgr -- ping -w 2 10.244.2.13
PING 10.244.2.13 (10.244.2.13): 56 data bytes

--- 10.244.2.13 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
command terminated with exit code 1
```

Above we see that the ping command failed because we used the `-w 2` to set a timeout. Policy works! But do other connections still work as they should?

We try to connect to the `database` *Service* and `database-0` *Pod* which should still work because it's not ICMP:

```
➜ candidate@cks7262:~$ k -n team-orange exec transmitter-866696fc57-6ccgr -- curl -m 2 --head database
HTTP/1.1 200 OK
...

➜ candidate@cks7262:~$ k -n team-orange exec transmitter-866696fc57-6ccgr -- curl -m 2 --head 10.244.2.13
HTTP/1.1 200 OK
...
```

Just as expected. And we try to connect to and ping a `type=messenger` *Pod*:

```
➜ candidate@cks7262:~$ k -n team-orange exec transmitter-866696fc57-6ccgr -- ping 10.244.1.126
PING 10.244.1.126 (10.244.1.126): 56 data bytes
64 bytes from 10.244.1.126: seq=0 ttl=63 time=1.577 ms
64 bytes from 10.244.1.126: seq=1 ttl=63 time=0.111 ms

➜ candidate@cks7262:~$ k -n team-orange exec transmitter-866696fc57-6ccgr -- curl -m 2 --head 10.244.1.126
HTTP/1.1 200 OK
...
```

Awesome!

Policy 3

Now to the final policy:

```
➜ candidate@cks7262:~$ vim 8_p3.yaml
```

```yaml
# cks7262:~/8_p3.yaml
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: p3
  namespace: team-orange
spec:
  endpointSelector:
    matchLabels:
      type: database
  egress:
  - toEndpoints:
    - matchLabels:
        type: messenger
    authentication:
      mode: "required"     # Enable Mutual Authentication
```

```
➜ candidate@cks7262:~$ k -f 8_p3.yaml apply
ciliumnetworkpolicy.cilium.io/p3 created

➜ candidate@cks7262:~$ k -n team-orange get cnp
NAME            AGE
default-allow   126m
p1              11m
p2              11m
p3              8s
```

Cilium ftw!

Question 9 | AppArmor Profile

Solve this question on: `ssh cks7262`

Some containers need to run more secure and restricted. There is an existing AppArmor profile located at `/opt/course/9/profile` on `cks7262` for this.

1. Install the AppArmor profile on *Node* `cks7262-node1`.

   Connect using `ssh cks7262-node1` from `cks7262`
2. Add label `security=apparmor` to the *Node*
3. Create a *Deployment* named `apparmor` in *Namespace* `default` with:
   - One replica of image `nginx:1.27.1`
   - NodeSelector for `security=apparmor`
   - Single container named `c1` with the AppArmor profile enabled only for this container

   The *Pod* might not run properly with the profile enabled. Write the logs of the *Pod* into `/opt/course/9/logs` on `cks7262` so another team can work on getting the application running.

> Use `sudo -i` to become root which may be required for this question

Answer:

https://kubernetes.io/docs/tutorials/clusters/apparmor

Step 1

First we have a look at the provided profile:

```
➜ ssh cks7262

➜ candidate@cks7262:~# vim /opt/course/9/profile
```

```
# cks7262:/opt/course/9/profile

#include <tunables/global>

profile very-secure flags=(attach_disconnected) {
  #include <abstractions/base>

  file,

  # Deny all file writes.
  deny /** w,
}
```

Very simple profile named `very-secure` which denies all file writes. Next we copy it onto the *Node*:

```
➜ candidate@cks7262:~# scp /opt/course/9/profile cks7262-node1:~/
profile                                    100%  161   329.9KB/s   00:00

➜ candidate@cks7262:~# ssh cks7262-node1

➜ cadidate@cks7262-node1:~# ls
profile
```

And install it:

```
➜ cadidate@cks7262-node1:~# sudo apparmor_parser -q ./profile
```

Verify it has been installed:

```
➜ cadidate@cks7262-node1:~# sudo apparmor_status
apparmor module is loaded.
7 profiles are loaded.
2 profiles are in enforce mode.
   cri-containerd.apparmor.d
   very-secure
0 profiles are in complain mode.
0 profiles are in prompt mode.
0 profiles are in kill mode.
5 profiles are in unconfined mode.
```

```
    firefox
    opera
    steam
    stress-ng
    thunderbird
36 processes have profiles defined.
36 processes are in enforce mode.
    /usr/local/apache2/bin/httpd (13154) cri-containerd.apparmor.d
...
0 processes are in complain mode.
0 processes are in prompt mode.
0 processes are in kill mode.
0 processes are unconfined but have a profile defined.
0 processes are in mixed mode.
```

There we see among many others the `very-secure` one, which is the name of the profile specified in `/opt/course/9/profile`.

Step 2

We label the *Node*:

```
k label -h # show examples

k label node cks7262-node1 security=apparmor
```

Step 3

Now we can go ahead and create the *Deployment* which uses the profile.

```
k create deploy apparmor --image=nginx:1.27.1 --dry-run=client -o yaml > 9_deploy.yaml

vim 9_deploy.yaml
```

```yaml
# 9_deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: apparmor
  name: apparmor
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: apparmor
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: apparmor
    spec:
      nodeSelector:                             # add
        security: apparmor                      # add
      containers:
      - image: nginx:1.27.1
        name: c1                                # change
        securityContext:                        # add
          appArmorProfile:                      # add
            type: Localhost                     # add
            localhostProfile: very-secure       # add
```

```
k -f 9_deploy.yaml create
```

What's the damage?

```
➜ candidate@cks7262:~# k get pod -owide | grep apparmor
apparmor-56b8498684-nshbp     0/1     CrashLoopBackOff  ...   cks7262-node1

➜ candidate@cks7262:~# k logs apparmor-56b8498684-nshbp
/docker-entrypoint.sh: 13: /docker-entrypoint.sh: cannot create /dev/null: Permission denied
/docker-entrypoint.sh: No files found in /docker-entrypoint.d/, skipping configuration
2024/09/07 16:19:08 [emerg] 1#1: mkdir() "/var/cache/nginx/client_temp" failed (13: Permission denied)
nginx: [emerg] mkdir() "/var/cache/nginx/client_temp" failed (13: Permission denied)
```

This looks alright, the *Pod* is running on `cks7262-node1` because of the nodeSelector. The AppArmor profile simply denies all filesystem writes, but Nginx needs to write into some locations to run, hence the errors.

It looks like our profile is running but we can confirm this as well by inspecting the container directly on the worker node:

```
➜ candidate@cks7262:~# ssh cks7262-node1

➜ candidate@cks7262-node1:~# sudo -i

➜ root@cks7262-node1:~# crictl pods | grep apparmor
42e0152b4f1d6       44 seconds ago      Ready    apparmor-56b8498684-nshbp  ...

➜ root@cks7262-node1:~# crictl ps -a | grep 42e0152b4f1d6
CONTAINER      ...   STATE     NAME  ...  POD ID          POD
c9f0c4a8f4d4a  ...   Exited    c1    ...  42e0152b4f1d6   apparmor-56b8498684-nshbp

➜ root@cks7262-node1:~# crictl inspect c9f0c4a8f4d4a | grep -i profile
            "profile_type": 1
            "profile_type": 2,
          "apparmor_profile": "localhost/very-secure"
        "apparmorProfile": "very-secure",
```

First we find the *Pod* by it's name and get the pod-id. Next we use `crictl ps -a` to also show stopped containers. Then `crictl inspect` shows that the container is using our AppArmor profile. Notice **to be fast** between `ps` and `inspect` because K8s will restart the *Pod* periodically when in error state.

To complete the task we write the logs into the required location:

```
➜ candidate@cks7262:~# k logs apparmor-56b8498684-nshbp > /opt/course/9/logs
```

Fixing the errors is the job of another team, lucky us.

Question 10 | Container Runtime Sandbox gVisor

Solve this question on: `ssh cks7262`

Team purple wants to run some of their workloads more secure. Worker node `cks7262-node2` has containerd already configured to support the runsc/gvisor runtime.

Connect to the worker node using `ssh cks7262-node2` from `cks7262`.

1. Create a *RuntimeClass* named `gvisor` with handler `runsc`
2. Create a *Pod* that uses the *RuntimeClass*. The *Pod* should be in *Namespace* `team-purple`, named `gvisor-test` and of image `nginx:1.27.1`. Ensure the *Pod* runs on `cks7262-node2`
3. Write the output of the `dmesg` command of the successfully started *Pod* into `/opt/course/10/gvisor-test-dmesg` on `cks7262`

Answer:

We check the nodes and we can see that all are using containerd:

```
➜ ssh cks7262

➜ candidate@cks7262:~$ k get node
NAME                    STATUS   ROLES              ... CONTAINER-RUNTIME
cks7262                 Ready    control-plane      ... containerd://1.7.12
cks7262-node1           Ready    <none>             ... containerd://1.7.12
cks7262-node2           Ready    <none>             ... containerd://1.7.12
```

But, according to the question text, just one has containerd configured to work with runsc/gvisor runtime which is `cks7262-node2`.

(Optionally) we can ssh into the worker node and check if containerd+runsc is configured:

```
➜ candidate@cks7262:~$ ssh cks7262-node2

➜ cadidate@cks7262-node2:~# runsc --version
runsc version release-20240820.0
spec: 1.1.0-rc.1

➜ cadidate@cks7262-node2:~# cat /etc/containerd/config.toml | grep runsc
  [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runsc]
    runtime_type = "io.containerd.runsc.v1"
```

Step 1

Now we best head to the k8s docs for *RuntimeClasses* https://kubernetes.io/docs/concepts/containers/runtime-class, steal an example and create the gvisor one:

```
vim 10_rtc.yaml
```

```yaml
# 10_rtc.yaml
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: gvisor
handler: runsc
```

```
k -f 10_rtc.yaml create
```

Step 2

And the required *Pod*:

```
k -n team-purple run gvisor-test --image=nginx:1.27.1 --dry-run=client -o yaml > 10_pod.yaml

vim 10_pod.yaml
```

```yaml
# 10_pod.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: gvisor-test
  name: gvisor-test
  namespace: team-purple
spec:
  nodeName: cks7262-node2 # add
  runtimeClassName: gvisor   # add
  containers:
  - image: nginx:1.27.1
    name: gvisor-test
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

```
k -f 10_pod.yaml create
```

After creating the pod we should check if it's running and if it uses the gvisor sandbox:

```
➜ candidate@cks7262:~$ k -n team-purple get pod gvisor-test
NAME          READY   STATUS    RESTARTS   AGE
gvisor-test   1/1     Running   0          30s

➜ candidate@cks7262:~$ k -n team-purple exec gvisor-test -- dmesg
[    0.000000] Starting gVisor...
[    0.336731] Waiting for children...
[    0.807396] Rewriting operating system in Javascript...
[    0.838661] Committing treasure map to memory...
[    1.082234] Adversarially training Redcode AI...
[    1.452222] Synthesizing system calls...
[    1.751229] Daemonizing children...
[    2.198949] Verifying that no non-zero bytes made their way into /dev/zero...
[    2.381878] Singleplexing /dev/ptmx...
[    2.398376] Checking naughty and nice process list...
[    2.544323] Creating cloned children...
[    3.010573] Setting up VFS...
[    3.467349] Setting up FUSE...
[    3.738725] Ready!
```

Looking deluxe.

Step 3

And as required we finally write the `dmesg` output into the file on `cks7262`:

```
➜ candidate@cks7262:~$ k -n team-purple exec gvisor-test > /opt/course/10/gvisor-test-dmesg -- dmesg
```

Question 11 | Secrets in ETCD

Solve this question on: `ssh cks7262`

There is an existing *Secret* called `database-access` in *Namespace* `team-green`.

1. Read the complete *Secret* content directly from ETCD (using `etcdctl`) and store it into `/opt/course/11/etcd-secret-content` on `cks7262`
2. Write the plain and decoded *Secret's* value of key "pass" into `/opt/course/11/database-password` on `cks7262`

> Use `sudo -i` to become root which may be required for this question

Answer:

Let's try to get the *Secret* value directly from ETCD, which will work since it isn't encrypted.

First, we ssh into the controlplane node where ETCD is running in this setup and check if `etcdctl` is installed and list it's options:

```
➜ ssh cks7262

➜ candidate@cks7262:~# sudo -i

➜ root@cks7262:~# etcdctl
NAME:
   etcdctl - A simple command line client for etcd.

WARNING:
   Environment variable ETCDCTL_API is not set; defaults to etcdctl v2.
   Set environment variable ETCDCTL_API=3 to use v3 API or ETCDCTL_API=2 to use v2 API.

USAGE:
   etcdctl [global options] command [command options] [arguments...]
...
   --cert-file value   identify HTTPS client using this SSL certificate file
   --key-file value    identify HTTPS client using this SSL key file
   --ca-file value     verify certificates of HTTPS-enabled servers using this CA bundle
...
```

Among others we see arguments to identify ourselves. The apiserver connects to ETCD, so we can run the following command to get the path of the necessary .crt and .key files:

```
cat /etc/kubernetes/manifests/kube-apiserver.yaml | grep etcd
```

The output is as follows :

```
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379 # optional since we're on same node
```

With this information we query ETCD for the secret value:

```
ETCDCTL_API=3 etcdctl \
--cert /etc/kubernetes/pki/apiserver-etcd-client.crt \
--key /etc/kubernetes/pki/apiserver-etcd-client.key \
--cacert /etc/kubernetes/pki/etcd/ca.crt get /registry/secrets/team-green/database-access
```

ETCD in Kubernetes stores data under `/registry/{type}/{namespace}/{name}`. This is how we came to look for `/registry/secrets/team-green/database-access`. There is also an example on a page [in the k8s documentation]() which you could access during the exam.

The task requires to store the output on our terminal. For this we can simply copy&paste the content into the requested location `/opt/course/11/etcd-secret-content` on `cks7262`.

```
# cks7262:/opt/course/11/etcd-secret-content

/registry/secrets/team-green/database-access
k8s


v1Secret

database-access
team-green"*$a01ef408-0a40-4fee-bd26-7adf346b3d222bB
0kubectl.kubernetes.io/last-applied-configuration{"apiVersion":"v1","data":
{"pass":"Y29uZmlkZW50aWFs"},"kind":"Secret","metadata":{"annotations":{},"name":"database-access","namespace":"team-
green"}}

kubectl-client-side-applyUpdatevFieldsV1:
{"f:data":{".":{},"f:pass":{}},"f:metadata":{"f:annotations":{".":{},"f:kubectl.kubernetes.io/last-applied-
configuration":{}}},"f:type":{}}B
pass
```

```
    confidentialOpaque"
```

We're also required to store the plain and "decrypted" database password. For this we can copy the base64-encoded value from the ETCD output and run on our terminal:

```
➜ root@cks7262:~# echo Y29uZmlkZW50aWFs | base64 -d > /opt/course/11/database-password

➜ root@cks7262:~# cat /opt/course/11/database-password
confidential
```

Question 12 | Hack Secrets

Solve this question on: `ssh cks3477`

You're asked to investigate a possible permission escape using the pre-defined context. The context authenticates as user `restricted` which has only limited permissions and shouldn't be able to read *Secret* values.

1. Switch to the restricted context with:

   ```
   k config use-context restricted@infra-prod
   ```

2. Try to find the password-key values of the *Secrets* `secret1`, `secret2` and `secret3` in *Namespace* `restricted` using context `restricted@infra-prod`
3. Write the decoded plaintext values into files `/opt/course/12/secret1`, `/opt/course/12/secret2` and `/opt/course/12/secret3` on `cks3477`
4. Switch back to the default context with:

   ```
   k config use-context kubernetes-admin@kubernetes
   ```

Answer:

First we should explore the boundaries, we can try:

```
➜ ssh cks3477

➜ candidate@cks3477:~# k config use-context restricted@infra-prod
Switched to context "restricted@infra-prod".

➜ candidate@cks3477:~# k -n restricted get role,rolebinding,clusterrole,clusterrolebinding
Error from server (Forbidden): roles.rbac.authorization.k8s.io is forbidden: User "restricted" cannot list resource
"roles" in API group "rbac.authorization.k8s.io" in the namespace "restricted"
Error from server (Forbidden): rolebindings.rbac.authorization.k8s.io is forbidden: User "restricted" cannot list
resource "rolebindings" in API group "rbac.authorization.k8s.io" in the namespace "restricted"
Error from server (Forbidden): clusterroles.rbac.authorization.k8s.io is forbidden: User "restricted" cannot list
resource "clusterroles" in API group "rbac.authorization.k8s.io" at the cluster scope
Error from server (Forbidden): clusterrolebindings.rbac.authorization.k8s.io is forbidden: User "restricted" cannot
list resource "clusterrolebindings" in API group "rbac.authorization.k8s.io" at the cluster scope
```

No permissions to view RBAC resources. So we try the obvious:

```
➜ candidate@cks3477:~# k -n restricted get secret
Error from server (Forbidden): secrets is forbidden: User "restricted" cannot list resource "secrets" in API group ""
in the namespace "restricted"

➜ candidate@cks3477:~# k -n restricted get secret -o yaml
apiVersion: v1
items: []
kind: List
metadata:
  resourceVersion: ""
Error from server (Forbidden): secrets is forbidden: User "restricted" cannot list resource "secrets" in API group ""
in the namespace "restricted"
```

We're not allowed to get or list any *Secrets*.

Secret 1

What can we see though?

```
➜ candidate@cks3477:~# k -n restricted get all
NAME                        READY   STATUS    RESTARTS   AGE
pod1-fd5d64b9c-pcx6q        1/1     Running   0          37s
pod2-6494f7699b-4hks5       1/1     Running   0          37s
pod3-748b48594-24s76        1/1     Running   0          37s
Error from server (Forbidden): replicationcontrollers is forbidden: User "restricted" cannot list resource
"replicationcontrollers" in API group "" in the namespace "restricted"
Error from server (Forbidden): services is forbidden: User "restricted" cannot list resource "services" in API group ""
in the namespace "restricted"
...
```

There are some *Pods*, lets check these out regarding *Secret* access:

```
k -n restricted get pod -o yaml | grep -i secret
```

This output provides us with enough information to do:

```
➜ candidate@cks3477:~# k -n restricted exec pod1-fd5d64b9c-pcx6q -- cat /etc/secret-volume/password
you-are

➜ candidate@cks3477:~# echo you-are > /opt/course/12/secret1
```

Secret 2

And for the second *Secret*:

```
➜ candidate@cks3477:~# k -n restricted exec pod2-6494f7699b-4hks5 -- env | grep PASS
PASSWORD=an-amazing

➜ candidate@cks3477:~# echo an-amazing > /opt/course/12/secret2
```

Secret 3

None of the *Pods* seem to mount `secret3` though. Can we create or edit existing *Pods* to mount `secret3`?

```
➜ candidate@cks3477:~# k -n restricted run test --image=nginx
Error from server (Forbidden): pods is forbidden: User "restricted" cannot create resource "pods" in API group "" in
the namespace "restricted"

➜ candidate@cks3477:~# k -n restricted auth can-i create pods
no
```

Doesn't look like it.

But the *Pods* seem to be able to access the *Secrets*, we can try to use a *Pod's ServiceAccount* to access the third *Secret*. We can actually see (like using `k -n restricted get pod -o yaml | grep automountServiceAccountToken`) that only *Pod* `pod3-*` has the *ServiceAccount* token mounted:

```
➜ candidate@cks3477:~# k -n restricted exec -it pod3-748b48594-24s76 -- sh

➜ / # mount | grep serviceaccount
tmpfs on /run/secrets/kubernetes.io/serviceaccount type tmpfs (ro,relatime)

➜ / # ls /run/secrets/kubernetes.io/serviceaccount
ca.crt     namespace  token
```

ℹ You should have knowledge about *ServiceAccounts* and how they work with *Pods* like described in the docs

We can see all necessary information to contact the apiserver manually (described in the docs):

```
➜ / # curl https://kubernetes.default/api/v1/namespaces/restricted/secrets -H "Authorization: Bearer $(cat
/run/secrets/kubernetes.io/serviceaccount/token)" -k
...
    {
      "metadata": {
        "name": "secret3",
        "namespace": "restricted",
...
          }
        ]
      },
      "data": {
        "password": "cEVuRXRSYVRpT24tdEVzVGVSCg=="
```

```
      },
      "type": "Opaque"
    }
...
```

Let's encode it and write it into the requested location:

```
➜ candidate@cks3477:~# echo cEVuRXRSYVRpT24tdEVzVGVSCg== | base64 -d
pEnEtRaTiOn-tEsTeR

➜ candidate@cks3477:~# echo cEVuRXRSYVRpT24tdEVzVGVSCg== | base64 -d > /opt/course/12/secret3
```

This will give us:

```
# cks3477:/opt/course/12/secret1
you-are
```

```
# cks3477:/opt/course/12/secret2
an-amazing
```

```
# cks3477:/opt/course/12/secret3
pEnEtRaTiOn-tEsTeR
```

We hacked all *Secrets*! It can be tricky to get RBAC right and secure.

> ℹ One thing to consider is that giving the permission to "list" *Secrets*, will also allow the user to read the *Secret* values like using `kubectl get secrets -o yaml` even without the "get" permission set.

Finally we switch back to the original context:

```
➜ candidate@cks3477:~$ k config use-context kubernetes-admin@kubernetes
Switched to context "kubernetes-admin@kubernetes".
```

Question 13 | Restrict access to Metadata Server

Solve this question on: `ssh cks3477`

There is a metadata service available at `http://192.168.100.21:32000` on which *Nodes* can reach sensitive data, like cloud credentials for initialisation. By default, all *Pods* in the cluster also have access to this endpoint. The DevSecOps team has asked you to restrict access to this metadata server.

In *Namespace* `metadata-access`:

1. Create a *NetworkPolicy* named `metadata-deny` which prevents egress to `192.168.100.21` for all *Pods* but still allows access to everything else
2. Create a *NetworkPolicy* named `metadata-allow` which allows *Pods* having label `role: metadata-accessor` to access endpoint `192.168.100.21`

There are existing *Pods* in the target *Namespace* with which you can test your policies, but don't change their labels.

Answer:

> ℹ Using a *NetworkPolicy* with ipBlock+except like done in our solution might cause security issues because of too open permissions that can't be further restricted. A better solution might be using a *CiliumNetworkPolicy*. Check the end of our solution for more information about this.

A great way to inspect and learn writing *NetworkPolices* is the [Network Policy Editor](), but it's not an allowed resource during the exam. Regarding Metadata Server security there was a [famous hack at Shopify]() which was based on revealed information via metadata for *Nodes*.

Check metadata server

Check the *Pods* in the *Namespace* `metadata-access` and their labels:

```
➜ ssh cks3477

➜ candidate@cks3477:~# k -n metadata-access get pods --show-labels
NAME                       ...   LABELS
pod1-56769f56fd-jd6sb   ...   app=pod1,pod-template-hash=56769f56fd
pod2-6f585c6f45-r6qqt   ...   app=pod2,pod-template-hash=6f585c6f45
pod3-67f7488665-7tn8x   ...   app=pod3,pod-template-hash=67f7488665,role=metadata-accessor
```

There are three *Pods* in the *Namespace* and one of them has the label `role=metadata-accessor`.

Check access to the metadata server from the *Pods*:

```
➜ candidate@cks3477:~# k exec -it -n metadata-access pod1-56769f56fd-jd6sb -- curl http://192.168.100.21:32000
metadata server

➜ candidate@cks3477:~# k exec -it -n metadata-access pod2-6f585c6f45-r6qqt -- curl http://192.168.100.21:32000
metadata server

➜ candidate@cks3477:~# k exec -it -n metadata-access pod3-67f7488665-7tn8x -- curl http://192.168.100.21:32000
metadata server
```

All three are able to access the metadata server.

Step 1

To restrict the access, we create a *NetworkPolicy* to deny access to the specific IP.

```
vim 13_metadata-deny.yaml
```

```
# 13_metadata-deny.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: metadata-deny
  namespace: metadata-access
spec:
  podSelector: {}
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
        except:
        - 192.168.100.21/32
```

```
k -f 13_metadata-deny.yaml apply
```

ℹ️ You should know about general [default-deny K8s NetworkPolcies](https://kubernetes.io).

Verify that access to the metadata server has been blocked:

```
➜ candidate@cks3477:~# k exec -it -n metadata-access pod1-56769f56fd-jd6sb -- curl -m 2 http://192.168.100.21:32000
curl: (28) Connection timed out after 2001 milliseconds
command terminated with exit code 28

➜ candidate@cks3477:~# k exec -it -n metadata-access pod2-6f585c6f45-r6qqt -- curl -m 2 http://192.168.100.21:32000
curl: (28) Connection timed out after 2001 milliseconds
command terminated with exit code 28

➜ candidate@cks3477:~# k exec -it -n metadata-access pod3-67f7488665-7tn8x -- curl -m 2 http://192.168.100.21:32000
curl: (28) Connection timed out after 2001 milliseconds
command terminated with exit code 28
```

But other endpoints are still reachable, like for example https://kubernetes.io:

```
➜ candidate@cks3477:~# k exec -it -n metadata-access pod1-56769f56fd-jd6sb -- curl --head -m 2 https://kubernetes.io
HTTP/2 200
accept-ranges: bytes
age: 9505
cache-control: public,max-age=0,must-revalidate
cache-status: "Netlify Edge"; hit
content-type: text/html; charset=UTF-8
date: Sun, 08 Sep 2024 11:37:09 GMT
etag: "be145d012d94f830fd1298f163db8ce4-ssl"
server: Netlify
strict-transport-security: max-age=31536000
```

```
x-nf-request-id: 01J78PRV7SREHYF5FY6EDXXXZM
content-length: 25304

➜ candidate@cks3477:~# k exec -it -n metadata-access pod2-6f585c6f45-r6qqt -- curl --head -m 2 https://kubernetes.io
HTTP/2 200
accept-ranges: bytes
age: 9542
cache-control: public,max-age=0,must-revalidate
cache-status: "Netlify Edge"; hit
content-type: text/html; charset=UTF-8
date: Sun, 08 Sep 2024 11:37:46 GMT
etag: "be145d012d94f830fd1298f163db8ce4-ssl"
server: Netlify
strict-transport-security: max-age=31536000
x-nf-request-id: 01J78PSZACQF3XBA9Y2W112KYZ
content-length: 25304

➜ candidate@cks3477:~# k exec -it -n metadata-access pod3-67f7488665-7tn8x -- curl --head -m 2 https://kubernetes.io
HTTP/2 200
accept-ranges: bytes
age: 9548
cache-control: public,max-age=0,must-revalidate
cache-status: "Netlify Edge"; hit
content-type: text/html; charset=UTF-8
date: Sun, 08 Sep 2024 11:37:52 GMT
etag: "be145d012d94f830fd1298f163db8ce4-ssl"
server: Netlify
strict-transport-security: max-age=31536000
x-nf-request-id: 01J78PT5DWH8TDXTAV21H029A2
content-length: 25304
```

Looking good.

Step 2

Now create another *NetworkPolicy* that allows access to the metadata server from *Pods* with label `role=metadata-accessor`.

```
vim 13_metadata-allow.yaml
```

```yaml
# 13_metadata-allow.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: metadata-allow
  namespace: metadata-access
spec:
  podSelector:
    matchLabels:
      role: metadata-accessor
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
        cidr: 192.168.100.21/32
```

```
k -f 13_metadata-allow.yaml apply
```

Verify that required *Pod* has access to metadata endpoint and others do not:

```
➜ candidate@cks3477:~# k exec -it -n metadata-access pod1-56769f56fd-jd6sb -- curl -m 2 http://192.168.100.21:32000
curl: (28) Connection timed out after 2001 milliseconds
command terminated with exit code 28

➜ candidate@cks3477:~# k exec -it -n metadata-access pod2-6f585c6f45-r6qqt -- curl -m 2 http://192.168.100.21:32000
curl: (28) Connection timed out after 2001 milliseconds
command terminated with exit code 28

➜ candidate@cks3477:~# k exec -it -n metadata-access pod3-67f7488665-7tn8x -- curl -m 2 http://192.168.100.21:32000
metadata server
```

It only works for the *Pod* having the label. With this we implemented the required security restrictions.

NetworkPolicy explanation

If a *Pod* doesn't have a matching *NetworkPolicy* then all traffic is allowed from and to it. Once a *Pod* has a matching *NP* then the contained rules are additive. This means that for *Pods* having label `metadata-accessor` the rules will be combined to:

```yaml
# merged policies into one for pods with label metadata-accessor
spec:
  podSelector: {}
```

```
  policyTypes:
  - Egress
  egress:
  - to: # first rule
    - ipBlock: # condition 1
        cidr: 0.0.0.0/0
        except:
        - 192.168.100.21/32
  - to: # second rule
    - ipBlock: # condition 1
        cidr: 192.168.100.21/32
```

We can see that the merged *NP* contains two separate rules with one condition each. We could read it as:

```
Allow outgoing traffic if:
(destination is 0.0.0.0/0 but not 192.168.100.21/32) OR (destination is 192.168.100.21/32)
```

Hence it allows *Pods* with label `metadata-accessor` to access everything.

Security Implications of this solution

Using a *NetworkPolicy* with ipBlock+except like done in our solution might cause security issues because of too open permissions that can't be further restricted. Because with vanilla Kubernetes *NetworkPolicies* it's **only possible to allow** certain ingress/egress. Once one egress rule exists, all other egress is forbidden, same for ingress.

Let's say we want to restrict the *NetworkPolicy* `metadata-deny` further, how would that be possible? We already specified one egress rule which allows outgoing traffic to ALL IPs using `0.0.0.0/0`, except one. If we now add another rule, all we can do is to allow more stuff:

```
# 13_metadata-deny.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: metadata-deny
  namespace: metadata-access
spec:
  podSelector: {}
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
        except:
        - 192.168.100.21/32
  - to:                             # ADD
    - namespaceSelector:            # ADD
        matchLabels:                # ADD
          project: myproject        # ADD
```

Above we added one additional egress rule to allow outgoing connection into a certain *Namespace*. If **only** that new rule would exist, then all other egress would be forbidden. But because both egress rules exist it could be read as:

```
Allow outgoing traffic if:
(destination is 0.0.0.0/0 but not 192.168.100.21/32)
   OR
(destination namespace has label project: myproject)
```

So once we allow egress/ingress using a too open ipBlock, we can't further restrict traffic which could be a big issue. A better solution might be for example using a *CiliumNetworkPolicy* which is able to define deny rules using `egressDeny` ([docs][https://doc.crds.dev/github.com/cilium/cilium/cilium.io/CiliumNetworkPolicy/v2]).

Question 14 | Syscall Activity

Solve this question on: `ssh cks7262`

There are *Pods* in *Namespace* `team-yellow`. A security investigation noticed that some processes running in these *Pods* are using the Syscall `kill`, which is forbidden by an internal policy of Team Yellow.

Find the offending *Pod(s)* and remove these by reducing the replicas of the parent *Deployment* to 0.

You can connect to the worker nodes using `ssh cks7262-node1` and `ssh cks7262-node2` from `cks7262`.

Answer:

Syscalls are used by processes running in Userspace to communicate with the Linux Kernel. There are many available syscalls: https://man7.org/linux/man-pages/man2/syscalls.2.html. It makes sense to restrict these for container processes and Docker/Containerd already restrict some by default, like the `reboot` Syscall. Restricting even more is possible for example using Seccomp or AppArmor.

Find processes of Pod

For this task we should simply find out which binary process executes a specific Syscall. Processes in containers are simply run on the same Linux operating system, but isolated. That's why we first check on which nodes the *Pods* are running:

```
➜ ssh cks7262

➜ candidate@cks7262:~# k -n team-yellow get pod -owide
NAME                           ...       NODE               NOMINATED NODE   ...
collector1-8d9dbc99f-hswfn     ...       cks7262-node1   <none>           <none>
collector1-8d9dbc99f-kwjtf     ...       cks7262-node1   <none>           <none>
collector2-66547ddfb5-5mvtz    ...       cks7262-node1   <none>           <none>
collector3-6ffb899c79-kwcxv    ...       cks7262-node1   <none>           <none>
collector3-6ffb899c79-lxm79    ...       cks7262-node1   <none>           <none>
```

All on `cks7262-node1`, hence we ssh into it and find the processes for the first *Deployment* `collector1` .

```
➜ candidate@cks7262:~# ssh cks7262-node1

➜ candidate@cks7262-node1:~# sudo -i

➜ root@cks7262-node1:~# crictl pods --name collector1
POD ID              CREATED             STATE        NAME                          ...
a61e29997e607       17 minutes ago      Ready        collector1-8d9dbc99f-kwjtf   ...
8b0c315bf5ccd       17 minutes ago      Ready        collector1-8d9dbc99f-hswfn   ...

➜ root@cks7262-node1:~# crictl ps --pod a61e29997e607
CONTAINER ID        IMAGE            ...    POD ID          POD
e18e766d288ac       71136cb0add32    ...    a61e29997e607   collector1-8d9dbc99f-kwjtf

➜ root@cks7262-node1:~# crictl inspect e18e766d288ac | grep args -A1
        "args": [
          "./collector1-process"
```

1. Using `crictl pods` we first searched for the *Pods* of *Deployment* `collector1`, which has two replicas
2. We then took one pod-id to find it's containers using `crictl ps`
3. And finally we used `crictl inspect` to find the process name, which is `collector1-process`.

   We can find the process PIDs (two because there are two *Pods*):

   ```
   ➜ root@cks7262-node1:~# ps aux | grep collector1-process
   root       13980  0.0  0.0 702216   384 ?   ...    ./collector1-process
   root       14079  0.0  0.0 702216   512 ?   ...    ./collector1-process
   ```

4. Or we could check for the PID with `crictl inspect`:

   ```
   ➜ root@cks7262-node1:~# crictl inspect e18e766d288ac | grep pid
       "pid": 14079,
               "pid": 1
               "type": "pid"
   ```

We *should* only have to check one of the PIDs because it's the same kind of *Pod*, just a second replica of the *Deployment*.

Check Syscalls of collector1

Using the PIDs we can call `strace` to find Sycalls:

```
➜ root@cks7262-node1:~# ps aux | grep collector1-process
root       13980  0.0  0.0 702216   384 ?   ...    ./collector1-process
root       14079  0.0  0.0 702216   512 ?   ...    ./collector1-process

➜ root@cks7262-node1:~# strace -p 14079
strace: Process 14079 attached
epoll_pwait(3, [], 128, 529, NULL, 1)   = 0
epoll_pwait(3, [], 128, 995, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
...
futex(0x4d7e68, FUTEX_WAIT_PRIVATE, 0, NULL) = 0
kill(666, SIGTERM)                      = -1 ESRCH (No such process)
futex(0x4d7e68, FUTEX_WAIT_PRIVATE, 0, NULL) = 0
kill(666, SIGTERM)                      = -1 ESRCH (No such process)
--- SIGURG {si_signo=SIGURG, si_code=SI_TKILL, si_pid=1, si_uid=0} ---
...
```

First try and already a catch! We see it uses the forbidden Syscall by calling `kill(666, SIGTERM)`.

Check Syscalls of collector2

Next let's check the *Deployment* `collector2` processes:

```
➜ root@cks7262-node1:~# ps aux | grep collector2-process
root       14046  0.0  0.0 702224   512 ?        Ssl  10:55   0:00 ./collector2-process

➜ root@cks7262-node1:~# strace -p 14046
strace: Process 14046 attached
futex(0x4d9e68, FUTEX_WAIT_PRIVATE, 0, NULL) = 0
futex(0x4d9e68, FUTEX_WAIT_PRIVATE, 0, NULL) = 0
futex(0x4d9e68, FUTEX_WAIT_PRIVATE, 0, NULL) = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 998, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
...
```

Looks alright.

Check Syscalls of collector3

What about the `collector3` *Deployment*:

```
➜ root@cks7262-node1:~# ps aux | grep collector3-process
root       14013  0.0  0.0 702480   640 ?        Ssl  10:55   0:00 ./collector3-process
root       14216  0.0  0.0 702480   640 ?        Ssl  10:55   0:00 ./collector3-process

➜ root@cks7262-node1:~# strace -p 14013
strace: Process 14013 attached
epoll_pwait(3, [], 128, 762, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 998, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
epoll_pwait(3, [], 128, 999, NULL, 1)   = 0
futex(0x4d9e68, FUTEX_WAIT_PRIVATE, 0, NULL) = 0
futex(0x4d9e68, FUTEX_WAIT_PRIVATE, 0, NULL) = 0
futex(0x4d9e68, FUTEX_WAIT_PRIVATE, 0, NULL) = 0
...
```

Also nothing about the forbidden Syscall.

Scale down Deployment

So we finish the task:

```
➜ root@cks7262:~# k -n team-yellow scale deploy collector1 --replicas 0
```

And the world is a bit safer again.

Question 15 | Configure TLS on Ingress

Solve this question on: `ssh cks7262`

In *Namespace* `team-pink` there is an existing Nginx *Ingress* resources named `secure` which accepts two paths `/app` and `/api` which point to different ClusterIP *Services*.

From your main terminal you can connect to it using for example:

- HTTP: `curl -v http://secure-ingress.test:31080/app`
- HTTPS: `curl -kv https://secure-ingress.test:31443/app`

Right now it uses a default generated TLS certificate by the Nginx Ingress Controller.

You're asked to instead use the key and certificate provided at `/opt/course/15/tls.key` and `/opt/course/15/tls.crt`. As it's a self-signed certificate you need to use `curl -k` when connecting to it.

Answer:

Investigate

We can get the IP address of the *Ingress* and we see it's the same one to which `secure-ingress.test` is pointing to:

```
➜ ssh cks7262

➜ candidate@cks7262:~# k -n team-pink get ing secure
NAME     CLASS    HOSTS                 ADDRESS          PORTS   AGE
secure   <none>   secure-ingress.test   192.168.100.12   80      7m11s

➜ candidate@cks7262:~# ping secure-ingress.test
PING cks7262-node1 (192.168.100.12) 56(84) bytes of data.
64 bytes from cks7262-node1 (192.168.100.12): icmp_seq=1 ttl=64 time=0.316 ms
```

Now, let's try to access the paths `/app` and `/api` via HTTP:

```
➜ candidate@cks7262:~# curl http://secure-ingress.test:31080/app
This is the backend APP!

➜ candidate@cks7262:~# curl http://secure-ingress.test:31080/api
This is the API Server!
```

What about HTTPS?

```
➜ candidate@cks7262:~# curl https://secure-ingress.test:31443/api
curl: (60) SSL certificate problem: unable to get local issuer certificate
More details here: https://curl.haxx.se/docs/sslcerts.html
...

➜ candidate@cks7262:~# curl -k https://secure-ingress.test:31443/api
This is the API Server!
```

HTTPS seems to be already working if we accept self-signed certificated using `-k`. But what kind of certificate is used by the server?

```
➜ candidate@cks7262:~# curl -kv https://secure-ingress.test:31443/api
...
* Server certificate:
*  subject: O=Acme Co; CN=Kubernetes Ingress Controller Fake Certificate
*  start date: Sep  8 10:55:34 2024 GMT
*  expire date: Sep  8 10:55:34 2025 GMT
*  issuer: O=Acme Co; CN=Kubernetes Ingress Controller Fake Certificate
*  SSL certificate verify result: self-signed certificate (18), continuing anyway.
*   Certificate level 0: Public key type RSA (2048/112 Bits/secBits), signed using sha256WithRSAEncryption
...
```

It seems to be "Kubernetes Ingress Controller Fake Certificate".

Implement own TLS certificate

First, let us generate a *Secret* using the provided key and certificate:

```
➜ candidate@cks7262:~# cd /opt/course/15

➜ candidate@cks7262:/opt/course/15$ ls
tls.crt  tls.key

➜ candidate@cks7262:/opt/course/15$ k -n team-pink create secret tls tls-secret --key tls.key --cert tls.crt
secret/tls-secret created
```

Now, we configure the *Ingress* to make use of this *Secret*:

```
➜ candidate@cks7262:~# k -n team-pink get ing secure -oyaml > 15_ing_bak.yaml

➜ candidate@cks7262:~# k -n team-pink edit ing secure
```

```
# kubectl -n team-pink edit ing secure
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
...
```

```
  generation: 1
  name: secure
  namespace: team-pink
...
spec:
  tls:                            # add
    - hosts:                      # add
      - secure-ingress.test       # add
      secretName: tls-secret      # add
  rules:
  - host: secure-ingress.test
    http:
      paths:
      - backend:
          service:
            name: secure-app
            port: 80
        path: /app
        pathType: ImplementationSpecific
      - backend:
          service:
            name: secure-api
            port: 80
        path: /api
        pathType: ImplementationSpecific
...
```

After adding the changes we check the *Ingress* resource again:

```
➜ candidate@cks7262:~# k -n team-pink get ing
NAME      CLASS    HOSTS                 ADDRESS          PORTS     AGE
secure    <none>   secure-ingress.test   192.168.100.12   80, 443   25m
```

It now actually lists port 443 for HTTPS. To verify:

```
➜ candidate@cks7262:~# curl -k https://secure-ingress.test:31443/api
This is the API Server!

➜ candidate@cks7262:~# curl -kv https://secure-ingress.test:31443/api
...
* Server certificate:
*  subject: CN=secure-ingress.test; O=secure-ingress.test
*  start date: Sep 25 18:22:10 2020 GMT
*  expire date: Sep 20 18:22:10 2040 GMT
*  issuer: CN=secure-ingress.test; O=secure-ingress.test
*  SSL certificate verify result: self-signed certificate (18), continuing anyway.
*   Certificate level 0: Public key type RSA (2048/112 Bits/secBits), signed using sha256WithRSAEncryption
...
```

We can see that the provided certificate is now being used by the *Ingress* for TLS termination. We still use `curl -k` because the provided certificate is self signed.

Question 16 | Docker Image Attack Surface

Solve this question on: `ssh cks7262`

There is a *Deployment* `image-verify` in *Namespace* `team-blue` which runs image `registry.killer.sh:5000/image-verify:v1`. DevSecOps has asked you to improve this image by:

1. Changing the base image to `alpine:3.12`
2. Not installing `curl`
3. Updating `nginx` to use the version constraint `>=1.18.0`
4. Running the main process as user `myuser`

Do not add any new lines to the Dockerfile, just edit existing ones. The file is located at `/opt/course/16/image/Dockerfile`.

Tag your version as `v2`. You can build, tag and push using:

```
cd /opt/course/16/image
podman build -t registry.killer.sh:5000/image-verify:v2 .
podman run registry.killer.sh:5000/image-verify:v2 # to test your changes
podman push registry.killer.sh:5000/image-verify:v2
```

Make the *Deployment* use your updated image tag `v2`.

Answer:

We should have a look at the Docker Image at first:

```
➜ ssh cks7262

➜ candidate@cks7262:~# cd /opt/course/16/image

➜ candidate@cks7262:/opt/course/16/image$ cp Dockerfile Dockerfile.bak

➜ candidate@cks7262:/opt/course/16/image$ vim Dockerfile
```

```
# cks7262:/opt/course/16/image/Dockerfile
FROM alpine:3.4
RUN apk update && apk add vim curl nginx=1.10.3-r0
RUN addgroup -S myuser && adduser -S myuser -G myuser
COPY ./run.sh run.sh
RUN ["chmod", "+x", "./run.sh"]
USER root
ENTRYPOINT ["/bin/sh", "./run.sh"]
```

Very simple Dockerfile which seems to execute a script `run.sh` :

```
# cks7262:/opt/course/16/image/run.sh
while true; do date; id; echo; sleep 1; done
```

So it only outputs current date and credential information in a loop. We can see that output in the existing *Deployment* `image-verify`:

```
➜ candidate@cks7262:~# k -n team-blue logs -f -l id=image-verify
Sun Sep  8 12:10:30 UTC 2024
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout),26(tape),27(video)
```

We see it's running as root.

Next we update the `Dockerfile` according to the requirements:

```
# /opt/course/16/image/Dockerfile

# change
FROM alpine:3.12

# change
RUN apk update && apk add vim nginx>=1.18.0

RUN addgroup -S myuser && adduser -S myuser -G myuser
COPY ./run.sh run.sh
RUN ["chmod", "+x", "./run.sh"]

# change
USER myuser

ENTRYPOINT ["/bin/sh", "./run.sh"]
```

Then we build the new image:

```
➜ candidate@cks7262:/opt/course/16/image$ podman build -t registry.killer.sh:5000/image-verify:v2 .

STEP 1/7: FROM alpine:3.12
Resolved "alpine" as an alias (/etc/containers/registries.conf.d/shortnames.conf)
Trying to pull docker.io/library/alpine:3.12...
Getting image source signatures
Copying blob 1b7ca6aea1dd done   |
Copying config 24c8ece58a done   |
Writing manifest to image destination
STEP 2/7: RUN apk update && apk add vim nginx>=1.18.0
fetch http://dl-cdn.alpinelinux.org/alpine/v3.12/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.12/community/x86_64/APKINDEX.tar.gz
v3.12.12-52-g800c17231ad [http://dl-cdn.alpinelinux.org/alpine/v3.12/main]
v3.12.12-52-g800c17231ad [http://dl-cdn.alpinelinux.org/alpine/v3.12/community]
OK: 12767 distinct packages available
--> 87781619777d
STEP 3/7: RUN addgroup -S myuser && adduser -S myuser -G myuser
--> ae553aeea607
STEP 4/7: COPY ./run.sh run.sh
--> 943d90848b52
STEP 5/7: RUN ["chmod", "+x", "./run.sh"]
--> 224656b3ddd8
STEP 6/7: USER myuser
--> 48de19088ba3
STEP 7/7: ENTRYPOINT ["/bin/sh", "./run.sh"]
COMMIT registry.killer.sh:5000/image-verify:v2
```

```
--> 09516fa460aa
Successfully tagged registry.killer.sh:5000/image-verify:v2
09516fa460aa74e13cf3dc64f2cfeeeeffa2e80c0b9a40fec1429fb8890f0e5e
```

We can then test our changes by running the container locally:

```
➜ candidate@cks7262:/opt/course/16/image$ podman run registry.killer.sh:5000/image-verify:v2

Sun Sep  8 12:11:30 UTC 2024
uid=101(myuser) gid=102(myuser) groups=102(myuser)

Sun Sep  8 12:11:31 UTC 2024
uid=101(myuser) gid=102(myuser) groups=102(myuser)

Sun Sep  8 12:11:32 UTC 2024
uid=101(myuser) gid=102(myuser) groups=102(myuser)
```

Looking good, so we push:

```
➜ candidate@cks7262:/opt/course/16/image$ podman push registry.killer.sh:5000/image-verify:v2

Getting image source signatures
Copying blob 6a1c1a1200d3 done   |
Copying blob fd5841c2ff0f done   |
Copying blob 761b8fb2b1d2 skipped: already exists
Copying blob aed9d43cb02e done   |
Copying blob 1ad27bdd166b done   |
Copying config 09516fa460 done   |
Writing manifest to image destination
```

And we update the *Deployment* to use the new image:

```
k -n team-blue edit deploy image-verify
```

```yaml
# kubectl -n team-blue edit deploy image-verify
apiVersion: apps/v1
kind: Deployment
metadata:
...
spec:
...
  template:
...
    spec:
      containers:
      - image: registry.killer.sh:5000/image-verify:v2 # change
```

And afterwards we can verify our changes by looking at the *Pod* logs:

```
➜ candidate@cks7262:~# k -n team-blue logs -f -l id=image-verify
Sun Sep  8 12:13:12 UTC 2024
uid=101(myuser) gid=102(myuser) groups=102(myuser)

Sun Sep  8 12:13:13 UTC 2024
uid=101(myuser) gid=102(myuser) groups=102(myuser)

Sun Sep  8 12:13:14 UTC 2024
uid=101(myuser) gid=102(myuser) groups=102(myuser)

Sun Sep  8 12:13:15 UTC 2024
uid=101(myuser) gid=102(myuser) groups=102(myuser)
```

Also to verify our changes even further:

```
➜ candidate@cks7262:~# k -n team-blue exec image-verify-55fbcd4c9b-x2flc -- curl
error: Internal error occurred: Internal error occurred: error executing command in container: failed to exec in
container: failed to start exec "47d8d3e96b8d214bf0f5d3f75d79fb5d52d351246de45ce4740559e7baa74a20": OCI runtime exec
failed: exec failed: unable to start container process: exec: "curl": executable file not found in $PATH: unknown

➜ candidate@cks7262:~# k -n team-blue exec image-verify-6cd88b645f-8d5cn -- nginx -v
nginx version: nginx/1.18.0
```

Another task solved.

Question 17 | Audit Log Policy

Solve this question on: `ssh cks3477`

Audit Logging has been enabled in the cluster with an Audit *Policy* located at `/etc/kubernetes/audit/policy.yaml` on `cks3477`.

1. Change the configuration so that only one backup of the logs is stored.
2. Alter the *Policy* in a way that it only stores logs:
    - From *Secret* resources, level Metadata
    - From "system:nodes" userGroups, level RequestResponse

   After you altered the *Policy* make sure to empty the log file so it only contains entries according to your changes, like using `echo > /etc/kubernetes/audit/logs/audit.log`.

> ℹ You can use `jq` to render json more readable, like `cat data.json | jq`

> ℹ Use `sudo -i` to become root which may be required for this question

Answer:

Step 1

First we check the apiserver configuration and change as requested:

```
➜ ssh cks3477

➜ candidate@cks3477:~# sudo -i

➜ root@cks3477:~# cp /etc/kubernetes/manifests/kube-apiserver.yaml ~/17_kube-apiserver.yaml # backup

➜ root@cks3477:~# vim /etc/kubernetes/manifests/kube-apiserver.yaml
```

```
# cks3477/etc/kubernetes/manifests/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
    - --audit-policy-file=/etc/kubernetes/audit/policy.yaml
    - --audit-log-path=/etc/kubernetes/audit/logs/audit.log
    - --audit-log-maxsize=5
    - --audit-log-maxbackup=1                                    # CHANGE
    - --advertise-address=192.168.100.21
    - --allow-privileged=true
...
```

> ℹ You should know how to enable Audit Logging completely yourself [as described in the docs](). Feel free to try this in another cluster in this environment.

Wait for the apiserver container to be restarted for example with:

```
watch crictl ps
```

Step 2

Now we look at the existing *Policy*:

```
➜ root@cks3477:~# vim /etc/kubernetes/audit/policy.yaml
```

```yaml
# cks3477:/etc/kubernetes/audit/policy.yaml
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: Metadata
```

We can see that this simple *Policy* logs everything on Metadata level. So we change it to the requirements:

```yaml
# cks3477:/etc/kubernetes/audit/policy.yaml
apiVersion: audit.k8s.io/v1
kind: Policy
rules:

# log Secret resources audits, level Metadata
- level: Metadata
  resources:
  - group: ""
    resources: ["secrets"]

# log node related audits, level RequestResponse
- level: RequestResponse
  userGroups: ["system:nodes"]

# for everything else don't log anything
- level: None
```

After saving the changes we have to restart the apiserver:

```
➜ root@cks3477:~# cd /etc/kubernetes/manifests/

➜ root@cks3477:/etc/kubernetes/manifests# mv kube-apiserver.yaml ..

➜ root@cks3477:/etc/kubernetes/manifests# watch crictl ps # wait for apiserver gone

➜ root@cks3477:/etc/kubernetes/manifests# echo > /etc/kubernetes/audit/logs/audit.log

➜ root@cks3477:/etc/kubernetes/manifests# mv ../kube-apiserver.yaml .

➜ root@cks3477:/etc/kubernetes/manifests# watch crictl ps # wait for apiserver created
```

That should be it.

Check the Audit Logs

Once the apiserver is running again we can check the new logs and scroll through some entries:

```
cat /etc/kubernetes/audit/logs/audit.log | tail | jq
```


```json
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "level": "Metadata",
  "auditID": "aac47b4d-d1fe-4ab8-a0eb-f7843a89560f",
  "stage": "RequestReceived",
  "requestURI": "/api/v1/namespaces/restricted/secrets?
allowWatchBookmarks=true&fieldSelector=metadata.name%3Dsecret1&resourceVersion=9028&timeout=9m45s&timeoutSeconds=585&wa
tch=true",
  "verb": "watch",
  "user": {
    "username": "system:node:cks3477-node1",
    "groups": [
      "system:nodes",
      "system:authenticated"
    ]
  },
  "sourceIPs": [
    "192.168.100.22"
  ],
  "userAgent": "kubelet/v1.31.1 (linux/amd64) kubernetes/a51b3b7",
  "objectRef": {
    "resource": "secrets",
    "namespace": "restricted",
    "name": "secret1",
    "apiVersion": "v1"
  },
  "requestReceivedTimestamp": "2024-09-08T12:20:43.920816Z",
  "stageTimestamp": "2024-09-08T12:20:43.920816Z"
}
```


Above we logged a watch action by Kubelet for *Secrets*, level Metadata.

```
{
```

```
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "level": "RequestResponse",
  "auditID": "80577862-91da-4bc4-bb9d-b1ebffdc0dda",
  "stage": "ResponseComplete",
  "requestURI": "/api/v1/nodes/cks3477?resourceVersion=0&timeout=10s",
  "verb": "get",
  "user": {
    "username": "system:node:cks3477",
    "groups": [
      "system:nodes",
      "system:authenticated"
    ]
  },
  "sourceIPs": [
    "192.168.100.21"
  ],
  "userAgent": "kubelet/v1.31.1 (linux/amd64) kubernetes/a51b3b7",
  "objectRef": {
    "resource": "nodes",
    "name": "cks3477",
    "apiVersion": "v1"
  },
  "responseStatus": {
    "metadata": {},
    "code": 200
  },
  "responseObject": {
    ...
  },
  "requestReceivedTimestamp": "2024-09-08T12:20:43.961117Z",
  "stageTimestamp": "2024-09-08T12:20:43.991929Z",
  "annotations": {
    "authorization.k8s.io/decision": "allow",
    "authorization.k8s.io/reason": ""
  }
}
```

And in the one above we logged a get action by system:nodes for *Nodes*, level RequestResponse.

Because all JSON entries are written in a single line in the file we could also run some simple verifications on our *Policy*:

```
# shows Secret entries
cat audit.log | grep '"resource":"secrets"' | wc -l

# confirms Secret entries are only of level Metadata
cat audit.log | grep '"resource":"secrets"' | grep -v '"level":"Metadata"' | wc -l

# shows RequestResponse level entries
cat audit.log | grep -v '"level":"RequestResponse"' | wc -l

# shows RequestResponse level entries are only for system:nodes
cat audit.log | grep '"level":"RequestResponse"' | grep -v "system:nodes" | wc -l
```

Looks like our job is done.

Question 18 | SBOM

Solve this question on: `ssh cks8930`

Your team received Software Bill Of Materials (SBOM) requests and you have been selected to generate some documents and scans:

1. Using `bom`:

   Generate a SPDX-Json SBOM of image `registry.k8s.io/kube-apiserver:v1.31.0`

   Store it at `/opt/course/18/sbom1.json` on `cks8930`
2. Using `trivy`:

   Generate a CycloneDX SBOM of image `registry.k8s.io/kube-controller-manager:v1.31.0`

   Store it at `/opt/course/18/sbom2.json` on `cks8930`
3. Using `trivy`:

   Scan the existing SPDX-Json SBOM at `/opt/course/18/sbom_check.json` on `cks8930` for known vulnerabilities. Save the result in Json format at `/opt/course/18/sbom_check_result.json` on `cks8930`

Answer:

SBOMs are like an ingredients list for food, just for software. So let's prepare something tasty!

Step 1: Create SBOM with Bom

The tool is https://github.com/kubernetes-sigs/bom.

```
➜ ssh cks8930

➜ candidate@cks8930:~$ bom
bom (Bill of Materials)
...
Usage:
  bom [command]

Available Commands:
  completion  Generate the autocompletion script for the specified shell
  document    bom document → Work with SPDX documents
  generate    bom generate → Create SPDX SBOMs
  help        Help about any command
  validate    bom validate → Check artifacts against an sbom
  version     Prints the version
  ...
```

We want to generate a new document and running `bom generate` should give us enough hints on how we can do this:

```
➜ candidate@cks8930:~$ bom generate --image registry.k8s.io/kube-apiserver:v1.31.0 --format json
INFO bom v0.6.0: Generating SPDX Bill of Materials
INFO Processing image reference: registry.k8s.io/kube-apiserver:v1.31.0
INFO Reference registry.k8s.io/kube-apiserver:v1.31.0 points to an index
INFO Reference image index points to 4 manifests
INFO Adding image registry.k8s.io/kube-
apiserver@sha256:64c595846c29945f619a1c3d420a8bfac87e93cb8d3641e222dd9ac412284001 (amd64/linux)
...
defined
{
  "SPDXID": "SPDXRef-DOCUMENT",
  "name": "SBOM-SPDX-f1e98645-98b1-41e3-89c6-800bebd8262c",
  "spdxVersion": "SPDX-2.3",
  "creationInfo": {
    "created": "2024-09-10T16:25:40Z",
    "creators": [
      "Tool: bom-v0.6.0"
    ],
    "licenseListVersion": "3.21"
  },
  "dataLicense": "CC0-1.0",
  "documentNamespace": "https://spdx.org/spdxdocs/k8s-releng-bom-2c6dd735-0888-4776-9644-09e690ded389",
  "documentDescribes": [
    "SPDXRef-Package-sha256-470179274deb9dc3a81df55cfc24823ce153147d4ebf2ed649a4f271f51eaddf"
  ],
  "packages": [
    {
...
```

Now we can also specify the output at the required location:

```
➜ candidate@cks8930:~$ bom generate --image registry.k8s.io/kube-apiserver:v1.31.0 --format json --output
/opt/course/18/sbom1.json
INFO bom v0.6.0: Generating SPDX Bill of Materials
INFO Processing image reference: registry.k8s.io/kube-apiserver:v1.31.0
INFO Reference registry.k8s.io/kube-apiserver:v1.31.0 points to an index
INFO Reference image index points to 4 manifests
INFO Adding image registry.k8s.io/kube-
apiserver@sha256:64c595846c29945f619a1c3d420a8bfac87e93cb8d3641e222dd9ac412284001 (amd64/linux)
...

➜ candidate@cks8930:~$ vim /opt/course/18/sbom1.json
```

```
# cks8930:/opt/course/18/sbom1.json
{
  "SPDXID": "SPDXRef-DOCUMENT",
  "name": "SBOM-SPDX-4b2df9c5-0526-471a-88d4-72cd41408f6e",
  "spdxVersion": "SPDX-2.3",
  "creationInfo": {
    "created": "2024-09-10T16:27:49Z",
    "creators": [
      "Tool: bom-v0.6.0"
    ],
    "licenseListVersion": "3.21"
  },
  "dataLicense": "CC0-1.0",
```

```
  "documentNamespace": "https://spdx.org/spdxdocs/k8s-releng-bom-5389c436-97e9-448c-95b0-bceaa602b4c0",
  "documentDescribes": [
    "SPDXRef-Package-sha256-470179274deb9dc3a81df55cfc24823ce153147d4ebf2ed649a4f271f51eaddf"
  ],
  "packages": [
    {
...
```

Using `bom document` it's for example possible to visualize SBOMs as well as query them for information, could become handy!

Step 2: Create SBOM with Trivy

Trivy the security scanner can also create and work with SBOMs. The usage is similar to scanning images for vulnerabilities, which would be:

```
➜ candidate@cks8930:~$ trivy image registry.k8s.io/kube-controller-manager:v1.31.0
2024-09-10T15:38:31Z    INFO    Need to update DB
2024-09-10T15:38:31Z    INFO    Downloading DB...       repository="ghcr.io/aquasecurity/trivy-db:2"
52.89 MiB / 52.89 MiB [-----------------------------------------------------------------------------------------------------
---------------------] 100.00% 8.89 MiB p/s 6.1s
2024-09-10T15:38:37Z    INFO    Vulnerability scanning is enabled
2024-09-10T15:38:37Z    INFO    Secret scanning is enabled
2024-09-10T15:38:37Z    INFO    If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2024-09-10T15:38:37Z    INFO    Please see also
https://aquasecurity.github.io/trivy/v0.51/docs/scanner/secret/#recommendation for faster secret detection
2024-09-10T15:38:41Z    INFO    Detected OS     family="debian" version="12.5"
2024-09-10T15:38:41Z    INFO    [debian] Detecting vulnerabilities...   os_version="12" pkg_num=3
2024-09-10T15:38:41Z    INFO    Number of language-specific files       num=2
2024-09-10T15:38:41Z    INFO    [gobinary] Detecting vulnerabilities...

registry.k8s.io/kube-controller-manager:v1.31.0 (debian 12.5)

Total: 0 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 0, CRITICAL: 0)
...
```

Here we can specify an output file and format:

```
➜ candidate@cks8930:~$ trivy image --help | grep format
  $ trivy image --format json --output result.json alpine:3.15
  # Generate a report in the CycloneDX format
  $ trivy image --format cyclonedx --output result.cdx alpine:3.15
  -f, --format string              format (table,json,template,sarif,cyclonedx,spdx,spdx-json,github,cosign-vuln)
(default "table")
...

➜ candidate@cks8930:~$ trivy image --format cyclonedx --output /opt/course/18/sbom2.json registry.k8s.io/kube-
controller-manager:v1.31.0
2024-09-10T16:20:21Z    INFO    "--format cyclonedx" disables security scanning. Specify "--scanners vuln" explicitly
if you want to include vulnerabilities in the CycloneDX report.
2024-09-10T16:20:24Z    INFO    Detected OS     family="debian" version="12.5"
2024-09-10T16:20:24Z    INFO    Number of language-specific files       num=2

candidate@cks8930:~$ vim /opt/course/18/sbom2.json
```

```
# cks8930:/opt/course/18/sbom2.json
{
  "$schema": "http://cyclonedx.org/schema/bom-1.5.schema.json",
  "bomFormat": "CycloneDX",
  "specVersion": "1.5",
  "serialNumber": "urn:uuid:70b535ca-0033-47aa-8648-27095d982eca",
  "version": 1,
  "metadata": {
    "timestamp": "2024-09-10T16:20:24+00:00",
    "tools": {
      "components": [
        {
          "type": "application",
          "group": "aquasecurity",
          "name": "trivy",
          "version": "0.51.2"
        }
      ]
    },
...
```

Step 3: Scan SBOM with Trivy

With Trivy we can also scan SBOM documents instead of images directly, we do this with the provided file:

```
➜ candidate@cks8930:~$ trivy sbom /opt/course/18/sbom_check.json
2024-09-10T15:50:05Z    INFO    Vulnerability scanning is enabled
2024-09-10T15:50:05Z    INFO    Detected SBOM format    format="spdx-json"
2024-09-10T15:50:06Z    INFO    Detected OS     family="debian" version="11.8"
```

```
2024-09-10T15:50:06Z    INFO    [debian] Detecting vulnerabilities...   os_version="11" pkg_num=3
2024-09-10T15:50:06Z    INFO    Number of language-specific files       num=6
2024-09-10T15:50:06Z    INFO    [gobinary] Detecting vulnerabilities...

/opt/course/18/sbom_check.json (debian 11.8)

Total: 0 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 0, CRITICAL: 0)


 (gobinary)

Total: 14 (UNKNOWN: 0, LOW: 0, MEDIUM: 11, HIGH: 2, CRITICAL: 1)

┌─────────────────────────────┬────────────────┬──────────┬─────────┬───────────────────┐
│          Library            │ Vulnerability  │ Severity │ Status  │ Installed Version
├─────────────────────────────┼────────────────┼──────────┼─────────┼───────────────────┤
│ golang.org/x/net            │ CVE-2023-45288 │ MEDIUM   │ fixed   │ v0.17.0
...
```

By default Trivy uses a human readable format, but we can change it to Json:

```
➜ candidate@cks8930:~$ trivy sbom --format json /opt/course/18/sbom_check.json
2024-09-10T15:53:31Z    INFO    Vulnerability scanning is enabled
2024-09-10T15:53:31Z    INFO    Detected SBOM format    format="spdx-json"
2024-09-10T15:53:31Z    INFO    Detected OS     family="debian" version="11.8"
2024-09-10T15:53:31Z    INFO    [debian] Detecting vulnerabilities...   os_version="11" pkg_num=3
2024-09-10T15:53:31Z    INFO    Number of language-specific files       num=6
2024-09-10T15:53:31Z    INFO    [gobinary] Detecting vulnerabilities...
{
  "SchemaVersion": 2,
  "CreatedAt": "2024-09-10T15:53:32.036341847Z",
  "ArtifactName": "/opt/course/18/sbom_check.json",
  "ArtifactType": "spdx",
  "Metadata": {
    "OS": {
      "Family": "debian",
      "Name": "11.8"
    },
...
```

Above we can see the `ArtifactName` used for the report. Finally we export it to the required location:

```
➜ candidate@cks8930:~$ trivy sbom --format json --output /opt/course/18/sbom_check_result.json
/opt/course/18/sbom_check.json
2024-09-10T16:50:56Z    INFO    Need to update DB
2024-09-10T16:50:56Z    INFO    Downloading DB...       repository="ghcr.io/aquasecurity/trivy-db:2"
52.89 MiB / 52.89 MiB [-------------------------------------------------------------------------------------------------
---------------------] 100.00% 9.90 MiB p/s 5.5s
2024-09-10T16:51:02Z    INFO    Vulnerability scanning is enabled
2024-09-10T16:51:02Z    INFO    Detected SBOM format    format="spdx-json"
2024-09-10T16:51:03Z    INFO    Detected OS     family="debian" version="11.8"
2024-09-10T16:51:03Z    INFO    [debian] Detecting vulnerabilities...   os_version="11" pkg_num=3
2024-09-10T16:51:03Z    INFO    Number of language-specific files       num=6
2024-09-10T16:51:03Z    INFO    [gobinary] Detecting vulnerabilities...

➜ candidate@cks8930:~$ vim /opt/course/18/sbom_check_result.json
```

```
# cks8930:/opt/course/18/sbom_check_result.json
{
  "SchemaVersion": 2,
  "CreatedAt": "2024-09-10T16:51:03.311963768Z",
  "ArtifactName": "/opt/course/18/sbom_check.json",
  "ArtifactType": "spdx",
  "Metadata": {
    "OS": {
      "Family": "debian",
      "Name": "11.8"
    },
    "ImageConfig": {
      "architecture": "",
      "created": "0001-01-01T00:00:00Z",
      "os": "",
      "rootfs": {
        "type": "",
        "diff_ids": null
      },
      "config": {}
    }
  },
  "Results": [
    {
...
```

Done.

Question 19 | Immutable Root FileSystem

Solve this question on: `ssh cks7262`

The *Deployment* `immutable-deployment` in *Namespace* `team-purple` should run immutable, it's created from file `/opt/course/19/immutable-deployment.yaml` on `cks7262`. Even after a successful break-in, it shouldn't be possible for an attacker to modify the filesystem of the running container.

1. Modify the *Deployment* in a way that no processes inside the container can modify the local filesystem, only `/tmp` directory should be writeable. Don't modify the Docker image.
2. Save the updated YAML under `/opt/course/19/immutable-deployment-new.yaml` on `cks7262` and update the running *Deployment*.

Answer:

Processes in containers can write to the local filesystem by default. This increases the attack surface when a non-malicious process gets hijacked. Preventing applications to write to disk or only allowing to certain directories can mitigate the risk. If there is for example a bug in Nginx which allows an attacker to override any file inside the container, then this only works if the Nginx process itself can write to the filesystem in the first place.

Making the root filesystem readonly can be done in the Docker image itself or in a *Pod* declaration.

Let us first check the *Deployment* `immutable-deployment` in *Namespace* `team-purple`:

```
➜ ssh cks7262

➜ candidate@cks7262:~# k -n team-purple edit deploy -o yaml
```

```
# kubectl -n team-purple edit deploy -o yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: team-purple
  name: immutable-deployment
  labels:
    app: immutable-deployment
  ...
spec:
  replicas: 1
  selector:
    matchLabels:
      app: immutable-deployment
  template:
    metadata:
      labels:
        app: immutable-deployment
    spec:
      containers:
      - image: busybox:1.32.0
        command: ['sh', '-c', 'tail -f /dev/null']
        imagePullPolicy: IfNotPresent
        name: busybox
      restartPolicy: Always
...
```

The container has write access to the Root File System, as there are no restrictions defined for the *Pods* or containers by an existing SecurityContext. And based on the task we're not allowed to alter the Docker image.

So we modify the YAML manifest to include the required changes:

```
➜ candidate@cks7262:~# cp /opt/course/19/immutable-deployment.yaml /opt/course/19/immutable-deployment-new.yaml

➜ candidate@cks7262:~# vim /opt/course/19/immutable-deployment-new.yaml
```

```
# cks7262:/opt/course/19/immutable-deployment-new.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: team-purple
  name: immutable-deployment
  labels:
    app: immutable-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: immutable-deployment
  template:
    metadata:
      labels:
```

```yaml
        app: immutable-deployment
    spec:
      containers:
      - image: busybox:1.32.0
        command: ['sh', '-c', 'tail -f /dev/null']
        imagePullPolicy: IfNotPresent
        name: busybox
        securityContext:                  # add
          readOnlyRootFilesystem: true    # add
        volumeMounts:                     # add
        - mountPath: /tmp                 # add
          name: temp-vol                  # add
      volumes:                            # add
      - name: temp-vol                    # add
        emptyDir: {}                      # add
      restartPolicy: Always
```

SecurityContexts can be set on *Pod* or container level, here the latter was asked. Enforcing `readOnlyRootFilesystem: true` will render the root filesystem readonly. We can then allow some directories to be writable by using an emptyDir volume.

Once the changes are made, let us update the *Deployment*:

```
➜ candidate@cks7262:~# k delete -f /opt/course/19/immutable-deployment-new.yaml
deployment.apps "immutable-deployment" deleted

➜ candidate@cks7262:~# k create -f /opt/course/19/immutable-deployment-new.yaml
deployment.apps/immutable-deployment created
```

We can verify if the required changes are propagated:

```
➜ candidate@cks7262:~# k -n team-purple exec immutable-deployment-5f4865fbf-7ckkj -- touch /abc.txt
touch: /abc.txt: Read-only file system
command terminated with exit code 1

➜ candidate@cks7262:~# k -n team-purple exec immutable-deployment-5f4865fbf-7ckkj -- touch /var/abc.txt
touch: /var/abc.txt: Read-only file system
command terminated with exit code 1

➜ candidate@cks7262:~# k -n team-purple exec immutable-deployment-5f4865fbf-7ckkj -- touch /etc/abc.txt
touch: /etc/abc.txt: Read-only file system
command terminated with exit code 1

➜ candidate@cks7262:~# k -n team-purple exec immutable-deployment-5f4865fbf-7ckkj -- touch /tmp/abc.txt

➜ candidate@cks7262:~# k -n team-purple exec immutable-deployment-5f4865fbf-7ckkj -- ls /tmp
abc.txt
```

The *Deployment* has been updated so that the container's file system is read-only, and the updated YAML has been placed under the required location. Sweet!

Question 20 | Update Kubernetes

Solve this question on: `ssh cks8930`

The cluster is running Kubernetes `1.30.5`, update it to `1.31.1`.

Use `apt` package manager and `kubeadm` for this.

Use `ssh cks8930-node1` from `cks8930` to connect to the worker node.

> ℹ Use `sudo -i` to become root which may be required for this question

Answer:

Let's have a look at the current versions:

```
➜ ssh cks8930

➜ candidate@cks8930:~# k get node
NAME            STATUS   ROLES           AGE   VERSION
cks8930         Ready    control-plane   12h   v1.30.5
cks8930-node1   Ready    <none>          12h   v1.30.5
```

We're logged via ssh into the control plane.

Control Plane Components

First we should update the control plane components running on the controlplane node, so we drain it:

```
➜ candidate@cks8930:~# k drain cks8930 --ignore-daemonsets
node/cks8930 cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/kube-proxy-r4w4r, kube-system/weave-net-kg2nx
node/cks8930 drained
```

Next we check versions:

```
➜ candidate@cks8930:~# sudo -i

➜ root@cks8930:~# kubelet --version
Kubernetes v1.30.5

➜ root@cks8930:~# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"31", GitVersion:"v1.31.1",
GitCommit:"948afe5ca072329a73c8e79ed5938717a5cb3d21", GitTreeState:"clean", BuildDate:"2024-09-11T21:26:49Z",
GoVersion:"go1.22.6", Compiler:"gc", Platform:"linux/amd64"}
```

We see above that `kubeadm` is already installed in the required version. Otherwise we would need to install it:

```
# not necessary because here kubeadm is already installed in correct version
apt-mark unhold kubeadm
apt-mark hold kubectl kubelet
apt install kubeadm=1.31.1-1.1
apt-mark hold kubeadm
```

Check what kubeadm has available as an upgrade plan:

```
➜ root@cks8930:~# kubeadm upgrade plan
[preflight] Running pre-flight checks.
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[upgrade] Running cluster health checks
[upgrade] Fetching available versions to upgrade to
[upgrade/versions] Cluster version: 1.30.5
[upgrade/versions] kubeadm version: v1.31.1
[upgrade/versions] Target version: v1.31.1
[upgrade/versions] Latest version in the v1.30 series: v1.30.5

Components that must be upgraded manually after you have upgraded the control plane with 'kubeadm upgrade apply':
COMPONENT   NODE            CURRENT   TARGET
kubelet     cks8930         v1.30.5   v1.31.1
kubelet     cks8930-node1   v1.30.5   v1.31.1

Upgrade to the latest stable version:

COMPONENT                 NODE      CURRENT    TARGET
kube-apiserver            cks8930   v1.30.5    v1.31.1
kube-controller-manager   cks8930   v1.30.5    v1.31.1
kube-scheduler            cks8930   v1.30.5    v1.31.1
kube-proxy                          1.30.5     v1.31.1
CoreDNS                             v1.11.3    v1.11.3
etcd                      cks8930   3.5.15-0   3.5.15-0

You can now apply the upgrade by executing the following command:

        kubeadm upgrade apply v1.31.1

_____________________________________________________________________


The table below shows the current state of component configs as understood by this version of kubeadm.
Configs that have a "yes" mark in the "MANUAL UPGRADE REQUIRED" column require manual config upgrade or
resetting to kubeadm defaults before a successful upgrade can be performed. The version to manually
upgrade to is denoted in the "PREFERRED VERSION" column.

API GROUP                 CURRENT VERSION   PREFERRED VERSION   MANUAL UPGRADE REQUIRED
kubeproxy.config.k8s.io   v1alpha1          v1alpha1            no
kubelet.config.k8s.io     v1beta1           v1beta1             no
_____________________________________________________________________
```

And we apply to the required version:

```
➜ root@cks8930:~# kubeadm upgrade apply v1.31.1
[preflight] Running pre-flight checks.
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
```

```
[upgrade] Running cluster health checks
[upgrade/version] You have chosen to change the cluster version to "v1.31.1"
[upgrade/versions] Cluster version: v1.30.5
[upgrade/versions] kubeadm version: v1.31.1
[upgrade] Are you sure you want to proceed? [y/N]: y
[upgrade/prepull] Pulling images required for setting up a Kubernetes cluster
[upgrade/prepull] This might take a minute or two, depending on the speed of your internet connection
[upgrade/prepull] You can also perform this action beforehand using 'kubeadm config images pull'
...

[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.31.1". Enjoy!

[upgrade/kubelet] Now that your control plane is upgraded, please proceed with upgrading your kubelets if you haven't
already done so.
```

Next we can check if our required version was installed correctly:

```
➜ root@cks8930:~# kubeadm upgrade plan
[preflight] Running pre-flight checks.
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[upgrade] Running cluster health checks
[upgrade] Fetching available versions to upgrade to
[upgrade/versions] Cluster version: 1.31.1
[upgrade/versions] kubeadm version: v1.31.1
[upgrade/versions] Target version: v1.31.1
[upgrade/versions] Latest version in the v1.31 series: v1.31.1
```

Control Plane kubelet and kubectl

Now we have to upgrade `kubelet` and `kubectl`:

```
➜ root@cks8930:~# apt update
Hit:1 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.30/deb  InRelease
Hit:2 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
2 packages can be upgraded. Run 'apt list --upgradable' to see them.

➜ root@cks8930:~# apt show kubelet | grep 1.31.1
Version: 1.31.1-1.1

➜ root@cks8930:~# apt install kubelet=1.31.1-1.1 kubectl=1.31.1-1.1
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  squashfs-tools
Use 'apt autoremove' to remove it.
The following packages will be upgraded:
  kubectl kubelet
2 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 26.4 MB of archives.
After this operation, 18.3 MB disk space will be freed.
Get:1 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  kubectl 1.31.1-1.1 [11.2
MB]
Get:2 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  kubelet 1.31.1-1.1 [15.2
MB]
Fetched 26.4 MB in 1s (32.3 MB/s)
(Reading database ... 72952 files and directories currently installed.)
Preparing to unpack .../kubectl_1.31.1-1.1_amd64.deb ...
Unpacking kubectl (1.31.1-1.1) over (1.30.5-1.1) ...
Preparing to unpack .../kubelet_1.31.1-1.1_amd64.deb ...
Unpacking kubelet (1.31.1-1.1) over (1.30.5-1.1) ...
Setting up kubectl (1.31.1-1.1) ...
Setting up kubelet (1.31.1-1.1) ...
Scanning processes...
Scanning candidates...
Scanning linux images...

Running kernel seems to be up-to-date.

Restarting services...
 systemctl restart kubelet.service

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
```

```
➜ root@cks8930:~# apt-mark hold kubelet kubectl
kubelet set on hold.
kubectl set on hold.

➜ root@cks8930:~# service kubelet restart

➜ root@cks8930:~# service kubelet status
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; preset: enabled)
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Fri 2024-10-04 09:41:20 UTC; 3s ago
       Docs: https://kubernetes.io/docs/
   Main PID: 16130 (kubelet)
      Tasks: 11 (limit: 1317)
     Memory: 71.2M (peak: 71.5M)
        CPU: 1.038s
     CGroup: /system.slice/kubelet.service
             └─16130 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/>
...

➜ root@cks8930:~# k get node
NAME            STATUS                     ROLES           AGE   VERSION
cks8930         Ready,SchedulingDisabled   control-plane   12h   v1.31.1
cks8930-node1   Ready                      <none>          12h   v1.30.5
```

Done, only uncordon missing:

```
➜ root@cks8930:~# k uncordon cks8930
node/cks8930 uncordoned
```

Data Plane

```
➜ root@cks8930:~# k get node
NAME            STATUS   ROLES           AGE   VERSION
cks8930         Ready    control-plane   12h   v1.31.1
cks8930-node1   Ready    <none>          12h   v1.30.5
```

Our data plane consist of one single worker node, so let's update it. First thing is we should drain it:

```
➜ root@cks8930:~# k drain cks8930-node1 --ignore-daemonsets
node/cks8930-node1 cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/kube-proxy-8h79n, kube-system/weave-net-z9vhk
evicting pod team-blue/pto-webform-666f748759-nvbbd
evicting pod default/classification-bot-7d458d4559-lhsp8
evicting pod team-blue/pto-webform-666f748759-45hnl
pod/pto-webform-666f748759-45hnl evicted
pod/pto-webform-666f748759-nvbbd evicted
pod/classification-bot-7d458d4559-lhsp8 evicted
node/cks8930-node1 drained
```

Next we ssh into it and upgrade kubeadm to the wanted version, or check if already done:

```
➜ root@cks8930:~# ssh cks8930-node1

➜ root@cks8930-node1:~# apt update
Hit:1 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.30/deb  InRelease
Hit:2 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
3 packages can be upgraded. Run 'apt list --upgradable' to see them.

➜ root@cks8930-node1:~# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"30", GitVersion:"v1.30.5",
GitCommit:"74e84a90c725047b1328ff3d589fedb1cb7a120e", GitTreeState:"clean", BuildDate:"2024-09-12T00:17:07Z",
GoVersion:"go1.22.6", Compiler:"gc", Platform:"linux/amd64"}

➜ root@cks8930-node1:~# apt-mark unhold kubeadm
kubeadm was already not hold.

➜ root@cks8930-node1:~# apt-mark hold kubectl kubelet
kubectl set on hold.
kubelet set on hold.

➜ root@cks8930-node1:~# apt install kubeadm=1.31.1-1.1
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
```

```
  squashfs-tools
Use 'apt autoremove' to remove it.
The following packages will be upgraded:
  kubeadm
1 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
Need to get 11.4 MB of archives.
After this operation, 8032 kB of additional disk space will be used.
Get:1 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  kubeadm 1.31.1-1.1 [11.4
MB]
Fetched 11.4 MB in 0s (23.7 MB/s)
(Reading database ... 72622 files and directories currently installed.)
Preparing to unpack .../kubeadm_1.31.1-1.1_amd64.deb ...
Unpacking kubeadm (1.31.1-1.1) over (1.30.5-1.1) ...
Setting up kubeadm (1.31.1-1.1) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.

➜ root@cks8930-node1:~# apt-mark hold kubeadm
kubeadm set on hold.

➜ root@cks8930-node1:~# kubeadm upgrade node
[upgrade] Reading configuration from the cluster...
[upgrade] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[preflight] Running pre-flight checks
[preflight] Skipping prepull. Not a control plane node.
[upgrade] Skipping phase. Not a control plane node.
[upgrade] Skipping phase. Not a control plane node.
[upgrade] Backing up kubelet config file to /etc/kubernetes/tmp/kubeadm-kubelet-config68138050/config.yaml
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[upgrade] The configuration for this node was successfully updated!
[upgrade] Now you should go ahead and upgrade the kubelet package using your package manager.
```

Now we follow what kubeadm told us in the last line and upgrade kubelet (and kubectl):

```
➜ root@cks8930-node1:~# apt-mark unhold kubectl kubelet
Canceled hold on kubectl.
Canceled hold on kubelet.

➜ root@cks8930-node1:~# apt install kubelet=1.31.1-1.1 kubectl=1.31.1-1.1
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  squashfs-tools
Use 'apt autoremove' to remove it.
The following packages will be upgraded:
  kubectl kubelet
2 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 26.4 MB of archives.
After this operation, 18.3 MB disk space will be freed.
Get:1 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  kubectl 1.31.1-1.1 [11.2
MB]
Get:2 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  kubelet 1.31.1-1.1 [15.2
MB]
Fetched 26.4 MB in 1s (32.8 MB/s)
(Reading database ... 72622 files and directories currently installed.)
Preparing to unpack .../kubectl_1.31.1-1.1_amd64.deb ...
Unpacking kubectl (1.31.1-1.1) over (1.30.5-1.1) ...
Preparing to unpack .../kubelet_1.31.1-1.1_amd64.deb ...
Unpacking kubelet (1.31.1-1.1) over (1.30.5-1.1) ...
Setting up kubectl (1.31.1-1.1) ...
Setting up kubelet (1.31.1-1.1) ...
Scanning processes...
Scanning candidates...
Scanning linux images...

Running kernel seems to be up-to-date.

Restarting services...
 systemctl restart kubelet.service

No containers need to be restarted.

No user sessions are running outdated binaries.
```

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.

➜ root@cks8930-node1:~# service kubelet restart

➜ root@cks8930-node1:~# service kubelet status
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; preset: enabled)
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Fri 2024-10-04 09:45:40 UTC; 2s ago
       Docs: https://kubernetes.io/docs/
   Main PID: 13370 (kubelet)
      Tasks: 9 (limit: 1113)
     Memory: 20.2M (peak: 20.4M)
        CPU: 577ms
     CGroup: /system.slice/kubelet.service
             └─13370 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/>
```

Looking good, what does the node status say?

```
➜ root@cks8930:~# k get node
NAME            STATUS                      ROLES           AGE   VERSION
cks8930         Ready                       control-plane   12h   v1.31.1
cks8930-node1   Ready,SchedulingDisabled    <none>          12h   v1.31.1
```

Beautiful, let's make it schedulable again:

```
➜ root@cks8930:~# k uncordon cks8930-node1
node/cks8930-node1 uncordoned

➜ root@cks8930:~# k get node
NAME            STATUS   ROLES           AGE   VERSION
cks8930         Ready    control-plane   12h   v1.31.1
cks8930-node1   Ready    <none>          12h   v1.31.1
```

We're up to date.

Question 21 | Image Vulnerability Scanning

Solve this question on: `ssh cks8930`

The Vulnerability Scanner `trivy` is installed on your main terminal. Use it to scan the following images for known CVEs:

- `nginx:1.16.1-alpine`
- `k8s.gcr.io/kube-apiserver:v1.18.0`
- `k8s.gcr.io/kube-controller-manager:v1.18.0`
- `docker.io/weaveworks/weave-kube:2.7.0`

Write all images that don't contain the vulnerabilities `CVE-2020-10878` or `CVE-2020-1967` into `/opt/course/21/good-images` on `cks8930`.

Answer:

The tool `trivy` is very simple to use, it compares images against public databases.

```
➜ ssh cks8930

➜ candidate@cks8930:~# trivy image nginx:1.16.1-alpine
2024-09-08T13:50:52Z    INFO    [db] Need to update DB
2024-09-08T13:50:52Z    INFO    [db] Downloading DB...  repository="ghcr.io/aquasecurity/trivy-db:2"
52.98 MiB / 52.98 MiB [--------------------------------------------------------------------------------------------------------------------] 100.00% 3.96 MiB p/s 14s
2024-09-08T13:51:07Z    INFO    [vuln] Vulnerability scanning is enabled
2024-09-08T13:51:07Z    INFO    [secret] Secret scanning is enabled
2024-09-08T13:51:07Z    INFO    [secret] If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2024-09-08T13:51:07Z    INFO    [secret] Please see also https://aquasecurity.github.io/trivy/v0.55/docs/scanner/secret#recommendation for faster secret detection
2024-09-08T13:51:13Z    INFO    Detected OS     family="alpine" version="3.10.4"
2024-09-08T13:51:13Z    INFO    [alpine] Detecting vulnerabilities...   os_version="3.10" repository="3.10" pkg_num=37
2024-09-08T13:51:13Z    INFO    Number of language-specific files       num=0
2024-09-08T13:51:13Z    WARN    This OS version is no longer supported by the distribution      family="alpine" version="3.10.4"
```

```
2024-09-08T13:51:13Z    WARN    The vulnerability detection may be insufficient because security updates are not
provided

nginx:1.16.1-alpine (alpine 3.10.4)

Total: 31 (UNKNOWN: 0, LOW: 2, MEDIUM: 14, HIGH: 14, CRITICAL: 1)
...
```

To solve the task we can run:

```
➜ candidate@cks8930:~# trivy image nginx:1.16.1-alpine | grep -E 'CVE-2020-10878|CVE-2020-1967'
...
│ libcrypto1.1  │ CVE-2020-1967  │ HIGH
│ libssl1.1     │ CVE-2020-1967  │

➜ candidate@cks8930:~# trivy image k8s.gcr.io/kube-apiserver:v1.18.0 | grep -E 'CVE-2020-10878|CVE-2020-1967'
...
│                         │ CVE-2020-10878

➜ candidate@cks8930:~# trivy image k8s.gcr.io/kube-controller-manager:v1.18.0 | grep -E 'CVE-2020-10878|CVE-2020-1967'
...
│                         │ CVE-2020-10878

➜ candidate@cks8930:~# trivy image docker.io/weaveworks/weave-kube:2.7.0 | grep -E 'CVE-2020-10878|CVE-2020-1967'

➜ candidate@cks8930:~#
```

The only image without the any of the two CVEs is `docker.io/weaveworks/weave-kube:2.7.0`, hence our answer will be:

```
# cks8930:/opt/course/21/good-images
docker.io/weaveworks/weave-kube:2.7.0
```

Question 22 | Manual Static Security Analysis

Solve this question on: `ssh cks8930`

The Release Engineering Team has shared some YAML manifests and Dockerfiles with you to review. The files are located under `/opt/course/22/files`.

As a container security expert, you are asked to perform a manual static analysis and find out possible security issues with respect to unwanted credential exposure. Running processes as root is of no concern in this task.

Write the filenames which have issues into `/opt/course/22/security-issues` on `cks8930`.

> In the Dockerfiles and YAML manifests, assume that the referred files, folders, secrets and volume mounts are present. Disregard syntax or logic errors.

Answer:

We check location `/opt/course/22/files` and list the files.

```
➜ ssh cks8930

➜ candidate@cks8930:~# ls -la /opt/course/22/files
-rw-r--r-- 1 candidate candidate 384 Sep  8 14:05 Dockerfile-go
-rw-r--r-- 1 candidate candidate 441 Sep  8 14:05 Dockerfile-mysql
-rw-r--r-- 1 candidate candidate 390 Sep  8 14:05 Dockerfile-py
-rw-r--r-- 1 candidate candidate 341 Sep  8 14:05 deployment-nginx.yaml
-rw-r--r-- 1 candidate candidate 723 Sep  8 14:05 deployment-redis.yaml
-rw-r--r-- 1 candidate candidate 529 Sep  8 14:05 pod-nginx.yaml
-rw-r--r-- 1 candidate candidate 228 Sep  8 14:05 pv-manual.yaml
-rw-r--r-- 1 candidate candidate 188 Sep  8 14:05 pvc-manual.yaml
-rw-r--r-- 1 candidate candidate 211 Sep  8 14:05 sc-local.yaml
-rw-r--r-- 1 candidate candidate 902 Sep  8 14:05 statefulset-nginx.yaml
```

We have 3 Dockerfiles and 7 Kubernetes Resource YAML manifests. Next we should go over each to find security issues with the way credentials have been used.

> You should be comfortable with Docker Best Practices and the Kubernetes Configuration Best Practices.

While navigating through the files we might notice:

Number 1

File `Dockerfile-mysql` might look innocent on first look. It copies a file `secret-token` over, uses it and deletes it afterwards. But because of the way Docker works, every `RUN`, `COPY` and `ADD` command creates a new layer and every layer is persistet in the image.

This means even if the file `secret-token` get's deleted in layer Z, it's still included with the image in layer X and Y. In this case it would be better to use for example variables passed to Docker.

```
# cks8930:/opt/course/22/files/Dockerfile-mysql
FROM ubuntu

# Add MySQL configuration
COPY my.cnf /etc/mysql/conf.d/my.cnf
COPY mysqld_charset.cnf /etc/mysql/conf.d/mysqld_charset.cnf

RUN apt-get update && \
    apt-get -yq install mysql-server-5.6 &&

# Add MySQL scripts
COPY import_sql.sh /import_sql.sh
COPY run.sh /run.sh

# Configure credentials
COPY secret-token .                                       # LAYER X
RUN /etc/register.sh ./secret-token                       # LAYER Y
RUN rm ./secret-token # delete secret token again         # LATER Z

EXPOSE 3306
CMD ["/run.sh"]
```

So we do:

```
echo Dockerfile-mysql >> /opt/course/22/security-issues
```

Number 2

The file `deployment-redis.yaml` is fetching credentials from a *Secret* named `mysecret` and writes these into environment variables. So far so good, but in the command of the *container* it's echoing these which can be directly read by any user having access to the logs.

```
# cks8930:/opt/course/22/files/deployment-redis.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: mycontainer
        image: redis
        command: ["/bin/sh"]
        args:
        - "-c"
        - "echo $SECRET_USERNAME && echo $SECRET_PASSWORD && docker-entrypoint.sh" # NOT GOOD
        env:
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: username
        - name: SECRET_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: password
```

Credentials in logs is never a good idea, hence we do:

```
echo deployment-redis.yaml >> /opt/course/22/security-issues
```

Number 3

In file `statefulset-nginx.yaml`, the password is directly exposed in the environment variable definition of the *container*.

```
# cks8930:/opt/course/22/files/statefulset-nginx.yaml
...
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        env:
        - name: Username
          value: Administrator
        - name: Password
          value: MyDiReCtP@sSw0rd               # NOT GOOD
        ports:
        - containerPort: 80
          name: web
..
```

This should better be injected via a *Secret*. So we do:

```
echo statefulset-nginx.yaml >> /opt/course/22/security-issues
```

```
➜ candidate@cks8930:~# cat /opt/course/22/security-issues
Dockerfile-mysql
deployment-redis.yaml
statefulset-nginx.yaml
```

Question 23 | ImagePolicyWebhook

Solve this question on: `ssh cks4024`

Team White created an *ImagePolicyWebhook* solution at `/opt/course/23/webhook` on `cks4024` which needs to be enabled for the cluster. There is an existing and working `webhook-backend` *Service* in *Namespace* `team-white` which will be the *ImagePolicyWebhook* backend.

1. Create an *AdmissionConfiguration* at `/opt/course/23/webhook/admission-config.yaml` which contains the following *ImagePolicyWebhook* configuration in the same file:

   ```
   imagePolicy:
     kubeConfigFile: /etc/kubernetes/webhook/webhook.yaml
     allowTTL: 10
     denyTTL: 10
     retryBackoff: 20
     defaultAllow: true
   ```

2. Configure the apiserver to:
   - Mount `/opt/course/23/webhook` at `/etc/kubernetes/webhook`
   - Use the *AdmissionConfiguration* at path `/etc/kubernetes/webhook/admission-config.yaml`
   - Enable the *ImagePolicyWebhook* admission plugin

As result the *ImagePolicyWebhook* backend should prevent container images containing `danger-danger` from being used, any other image should still work.

> ℹ️ Create a backup of `/etc/kubernetes/manifests/kube-apiserver.yaml` outside of `/etc/kubernetes/manifests` so you can revert back in case of issues

Use `sudo -i` to become root which may be required for this question

Answer:

The *ImagePolicyWebhook* is a Kubernetes Admission Controller which allows a backend to make admission decisions. According to the question that backend exists already and is working, let's have a short look:

```
➜ ssh cks4024

➜ candidate@cks4024:~$ k -n team-white get pod,svc,secret
NAME                                   READY   STATUS    RESTARTS   AGE
pod/webhook-backend-669f74bf8d-2vgnd   1/1     Running   0          18s

NAME                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/webhook-backend   ClusterIP   10.111.10.111   <none>        443/TCP   67m

NAME                     TYPE                DATA   AGE
secret/webhook-backend   kubernetes.io/tls   2      59s
```

The idea is to let the apiserver know it should contact that `webhook-backend` before any *Pod* is created and only if it receives a success-response the *Pod* will be created. We can see the *Service* IP is `10.111.10.111` and somehow we need to tell that to the apiserver.

```
➜ candidate@cks4024:~$ cd /opt/course/23/webhook

➜ candidate@cks4024:/opt/course/23/webhook$ ls
webhook-backend.crt  webhook-backend.csr  webhook-backend.key  webhook.yaml

➜ candidate@cks4024:/opt/course/23/webhook$ vim webhook.yaml
```

```yaml
# cks4024:/opt/course/23/webhook/webhook.yaml
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /etc/kubernetes/webhook/webhook-backend.crt
    server: https://10.111.10.111
  name: webhook
contexts:
- context:
    cluster: webhook
    user: webhook-backend.team-white.svc
  name: webhook
current-context: webhook
kind: Config
users:
- name: webhook-backend.team-white.svc
  user:
    client-certificate: /etc/kubernetes/pki/apiserver.crt
    client-key: /etc/kubernetes/pki/apiserver.key
```

Here we see a KubeConfig formatted file which the apiserver will use to contact the `webhook-backend` via specified URL `server: https://10.111.10.111`, which is the *Service* IP we noticed earlier. In addition we have a certificate at path `certificate-authority: /etc/kubernetes/webhook/webhook-backend.crt` which is used by the apiserver to communicate with the backend.

Step 1

We create the *AdmissionConfiguration* which contains the provided *ImagePolicyWebhook* config in the same file:

```
➜ candidate@cks4024:~$ sudo -i

➜ root@cks4024:~# vim /opt/course/23/webhook/admission-config.yaml
```

```yaml
# cks4024:/opt/course/23/webhook/admission-config.yaml
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
  - name: ImagePolicyWebhook
    configuration:
      imagePolicy:
        kubeConfigFile: /etc/kubernetes/webhook/webhook.yaml
        allowTTL: 10
        denyTTL: 10
        retryBackoff: 20
        defaultAllow: true
```

This should already be the solution for that step. Note that it's also possible to specify a path inside the *AdmissionConfiguration* pointing to a different file containing the *ImagePolicyWebhook*:

```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
  - name: ImagePolicyWebhook
    path: imagepolicyconfig.yaml
```

Step 2

We now register the *AdmissionConfiguration* with the apiserver. And before we do so we should probably create a backup so we can revert back easy:

> ℹ️ Create a backup always outside of `/etc/kubernetes/manifests` so the kubelet won't try to create the backup file as a static *Pod*

```
➜ root@cks4024:~# cp /etc/kubernetes/manifests/kube-apiserver.yaml ~/s23_kube-apiserver.yaml

➜ root@cks4024:~# vim /etc/kubernetes/manifests/kube-apiserver.yaml
```

```
# cks4024:/etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
...
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.100.11
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction,ImagePolicyWebhook # CHANGE
    - --admission-control-config-file=/etc/kubernetes/webhook/admission-config.yaml # ADD
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
...
    image: registry.k8s.io/kube-apiserver:v1.30.1
    name: kube-apiserver
...
    volumeMounts:
    - mountPath: /etc/kubernetes/webhook  # ADD
      name: webhook                       # ADD
      readOnly: true                      # ADD
    - mountPath: /etc/ssl/certs
      name: ca-certs
      readOnly: true
    - mountPath: /etc/ca-certificates
      name: etc-ca-certificates
      readOnly: true
...
  volumes:
  - hostPath:                             # ADD
      path: /opt/course/23/webhook        # ADD
      type: DirectoryOrCreate             # ADD
    name: webhook                         # ADD
  - hostPath:
      path: /etc/ssl/certs
      type: DirectoryOrCreate
    name: ca-certs
  - hostPath:
      path: /etc/ca-certificates
      type: DirectoryOrCreate
    name: etc-ca-certificates
...
```

If there is no existing `--enable-admission-plugins` argument then we need to create it, otherwise we can expand it as done above.

We create a hostPath volume of `/opt/course/23/webhook` and mount it to `/etc/kubernetes/webhook` inside the apiserver container. This way we can then reference `/etc/kubernetes/webhook/admission-config.yaml` using the `--admission-control-config-file` argument. Also this means that the provided path `/etc/kubernetes/webhook/webhook.yaml` in `/opt/course/23/webhook/admission-config.yaml` will work.

After we saved the changes we need to wait for the apiserver container to be restarted, this can take a minute:

```
➜ root@cks4024:~# watch crictl ps
```

Errors

In case the apiserver doesn't restart, or gets restarted over and over again, we should check the errors logs in `/var/log/pods/` to investigate any misconfiguration.

If there are no logs available we could also check the kubelet logs in `/var/log/syslog` or `journalctl -u kubelet`.

If the apiserver comes back up and there are no errors but the webhook just doesn't work then it could be a connection issue. Because the *ImagePolicyWebhook* config has setting `defaultAllow: true`, a connection issue between apiserver and `webhook-backend` would allow all *Pods*. We should see information about this in the apiserver logs or `kubectl get events -A`.

Result

Now we can simply try to create a *Pod* with a forbidden image and one with a still allowed one:

```
➜ root@cks4024:~# k run test1 --image=something/danger-danger
Error from server (Forbidden): pods "test1" is forbidden: image policy webhook backend denied one or more images:
Images containing danger-danger are not allowed

➜ root@cks4024:~# k run test2 --image=nginx:alpine
pod/test2 created

➜ root@cks4024:~# k get pod
NAME                                  READY   STATUS    RESTARTS   AGE
test2                                 1/1     Running   0          7s
```

The `webhook-backend` used in this scenario also outputs some log messages every time it receives a request from the apiserver:

```
➜ root@cks4024:~# k -n team-white logs deploy/webhook-backend
POST request received with body: {"kind":"ImageReview","apiVersion":"imagepolicy.k8s.io/v1alpha1","metadata":
{"creationTimestamp":null},"spec":{"containers":[{"image":"registry.k8s.io/kube-apiserver:v1.30.1"}],"namespace":"kube-
system"},"status":{"allowed":false}}
POST request check image name: registry.k8s.io/kube-apiserver:v1.30.1

POST request received with body: {"kind":"ImageReview","apiVersion":"imagepolicy.k8s.io/v1alpha1","metadata":
{"creationTimestamp":null},"spec":{"containers":[{"image":"something/danger-danger"}],"namespace":"default"},"status":
{"allowed":false}}
POST request check image name: something/danger-danger
POST image name FORBIDDEN

POST request received with body: {"kind":"ImageReview","apiVersion":"imagepolicy.k8s.io/v1alpha1","metadata":
{"creationTimestamp":null},"spec":{"containers":[{"image":"nginx:alpine"}],"namespace":"default"},"status":
{"allowed":false}}
POST request check image name: nginx:alpine
```

In this case we see that the `webhook-backend` received three requests for *Pod* admissions:

1. `registry.k8s.io/kube-apiserver:v1.30.1`
2. `something/danger-danger`
3. `nginx:alpine`

Even before we created the two test *Pods*, the backend received a request to check the container image of the `kube-apiserver` itself. This is why misconfigurations can become quite dangerous for the whole cluster if even Kubernetes internal or CNI *Pods* are prevented from being created.

CKS Simulator Preview Kubernetes 1.31

https://killer.sh

This is a preview of the full CKS Simulator course content.

The full course contains 22 questions and scenarios which cover all the CKS areas. The course also provides a browser terminal which is a very close replica of the original one. This is great to get used and comfortable before the real exam. After the test session (120 minutes), or if you stop it early, you'll get access to all questions and their detailed solutions. You'll have 36 hours cluster access in total which means even after the session, once you have the solutions, you can still play around.

The following preview will give you an idea of what the full course will provide. These preview questions are not part of the 22 in the full course but in addition to it. But the preview questions are part of the same CKS simulation environment which we setup for you, so with access to the full course you can solve these too.

These questions can be solved in the test environment provided through the CKS Simulator

Preview Question 1

Solve this question on: `ssh cks3477`

You're asked to implement some RBAC for user `gianna`:

1. There are existing cluster-level RBAC resource in place to, among other things, ensure that user `gianna` can never read *Secret* contents cluster-wide. Confirm this is correct or restrict the existing RBAC resources to ensure this.
2. I addition, create more RBAC resources to allow user `gianna` to create *Pods* and *Deployments* in *Namespaces* `security`, `restricted` and `internal`. It's likely the user will receive these exact permissions as well for other *Namespaces* in the future.

To test your RBAC your can:

- Switch to the other context with:

```
k config use-context gianna@infra-prod
```

- And afterwards switch back to the default context with:

```
k config use-context kubernetes-admin@kubernetes
```

Answer:

Part 1 - check existing RBAC rules

We should probably first have a look at the existing RBAC resources for user `gianna`. We don't know the resource names but we know these are cluster-level so we can search for a *ClusterRoleBinding*:

```
➜ ssh cks3477

➜ candidate@cks3477:~# k get clusterrolebinding -oyaml | grep gianna -A10 -B20
```

From this we see the binding is also called `gianna`:

```
➜ candidate@cks3477:~# k edit clusterrolebinding gianna
```

```
# kubectl edit clusterrolebinding gianna
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  creationTimestamp: "2020-09-26T13:57:58Z"
  name: gianna
  resourceVersion: "3049"
  selfLink: /apis/rbac.authorization.k8s.io/v1/clusterrolebindings/gianna
  uidapiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
...
  name: gianna
  resourceVersion: "2337"
  uid: 13fac261-e523-4ea4-a2a3-ca2e6fc455a7
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: gianna
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: gianna: 72b64a3b-5958-4cf8-8078-e5be2c55b25d
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: gianna
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: gianna
```

It links user `gianna` to same named *ClusterRole*:

```
➜ candidate@cks3477:~# k edit clusterrole gianna
```

```
# kubectl edit clusterrole gianna
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
```

```
...
  name: gianna
  resourceVersion: "2334"
  uid: eaab42d8-817d-4922-98e2-eafb09c8bfbe
rules:
- apiGroups:
  - ""
  resources:
  - secrets
  - configmaps
  - pods
  - namespaces
  verbs:
  - list
```

According to the task the user should never be able to read *Secrets* content. They verb `list` might indicate on first look that this is correct. We can also check using K8s User Impersonation:

```
➜ candidate@cks3477:~# k auth can-i list secrets --as gianna
yes

➜ candidate@cks3477:~# k auth can-i get secrets --as gianna
no
```

But let's have a closer look:

```
➜ candidate@cks3477:~# k config use-context gianna@infra-prod
Switched to context "gianna@infra-prod".

➜ candidate@cks3477:~# k -n security get secrets
NAME                  TYPE                                  DATA   AGE
default-token-gn455   kubernetes.io/service-account-token   3      20m
kubeadmin-token       Opaque                                1      20m
mysql-admin           Opaque                                1      20m
postgres001           Opaque                                1      20m
postgres002           Opaque                                1      20m
vault-token           Opaque                                1      20m

➜ candidate@cks3477:~# k -n security get secret kubeadmin-token
Error from server (Forbidden): secrets "kubeadmin-token" is forbidden: User "gianna" cannot get resource "secrets" in
API group "" in the namespace "security"
```

Still all good. **But** being able to list resources also allows to specify the format:

```
➜ candidate@cks3477:~# k -n security get secrets -oyaml | grep password
    password: ekhHYW5lQUVTaVVxCg==
        {"apiVersion":"v1","data":{"password":"ekhHYW5lQUVTaVVxCg=="},"kind":"Secret","metadata":{"annotations":
{},"name":"kubeadmin-token","namespace":"security"},"type":"Opaque"}
          f:password: {}
    password: bWdFVlBSdEpEWHBFCg==
...
```

The user `gianna` is actually able to read *Secret* content. To prevent this we should remove the ability to list these:

```
➜ candidate@cks3477:~# k config use-context infra-prod # back to admin context

➜ candidate@cks3477:~# k edit clusterrole gianna
```

```
# kubectl edit clusterrole gianna
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
...
  creationTimestamp: "2024-09-08T11:03:26Z"
  name: gianna
  resourceVersion: "2334"
  uid: eaab42d8-817d-4922-98e2-eafb09c8bfbe
rules:
- apiGroups:
  - ""
  resources:
  #- secrets          # REMOVE
  - configmaps
  - pods
  - namespaces
  verbs:
  - list
```

And the result:

```
➜ candidate@cks3477:~# k auth can-i list secrets --as gianna
no

➜ candidate@cks3477:~# k auth can-i get secrets --as gianna
no
```

Better.

Part 2 - create additional RBAC rules

Let's talk a little about RBAC resources:

A *ClusterRole*|*Role* defines a set of permissions and **where it is available**, in the whole cluster or just a single *Namespace*.

A *ClusterRoleBinding*|*RoleBinding* connects a set of permissions with an account and defines **where it is applied**, in the whole cluster or just a single *Namespace*.

Because of this there are 4 different RBAC combinations and 3 valid ones:

1. *Role* + *RoleBinding* (available in single *Namespace*, applied in single *Namespace*)
2. *ClusterRole* + *ClusterRoleBinding* (available cluster-wide, applied cluster-wide)
3. *ClusterRole* + *RoleBinding* (available cluster-wide, applied in single *Namespace*)
4. *Role* + *ClusterRoleBinding* (**NOT POSSIBLE:** available in single *Namespace*, applied cluster-wide)

The user `gianna` should be able to create *Pods* and *Deployments* in three *Namespaces*. We can use number 1 or 3 from the list above. But because the task says: "The user might receive these exact permissions as well for other *Namespaces* in the future", we choose number 3 as it requires to only create one *ClusterRole* instead of three *Roles*.

```
➜ candidate@cks3477:~# k create clusterrole gianna-additional --verb=create --resource=pods --resource=deployments

clusterrole.rbac.authorization.k8s.io/gianna-additional created
```

This will create a *ClusterRole* like:

```
# kubectl create clusterrole gianna-additional --verb=create --resource=pods --resource=deployments
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: gianna-additional
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - create
- apiGroups:
  - apps
  resources:
  - deployments
  verbs:
  - create
```

Next the three bindings:

```
➜ candidate@cks3477:~# k -n security create rolebinding gianna-additional \
--clusterrole=gianna-additional --user=gianna

rolebinding.rbac.authorization.k8s.io/gianna-additional created


➜ candidate@cks3477:~# k -n restricted create rolebinding gianna-additional \
--clusterrole=gianna-additional --user=gianna

rolebinding.rbac.authorization.k8s.io/gianna-additional created


➜ candidate@cks3477:~# k -n internal create rolebinding gianna-additional \
--clusterrole=gianna-additional --user=gianna

rolebinding.rbac.authorization.k8s.io/gianna-additional created
```

Which will create *RoleBindings* like:

```
# k -n security create rolebinding gianna-additional --clusterrole=gianna-additional --user=gianna
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: null
  name: gianna-additional
```

```yaml
  namespace: security
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: gianna-additional
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: gianna
```

And we test:

```
➜ candidate@cks3477:~# k -n default auth can-i create pods --as gianna
no

➜ candidate@cks3477:~# k -n security auth can-i create pods --as gianna
yes

➜ candidate@cks3477:~# k -n restricted auth can-i create pods --as gianna
yes

➜ candidate@cks3477:~# k -n internal auth can-i create pods --as gianna
yes
```

Feel free to verify this as well by actually creating *Pods* and *Deployments* as user `gianna` through context `gianna@infra-prod`.

Preview Question 2

Solve this question on: `ssh cks3477`

Namespace `security` contains five *Secrets* of type Opaque which can be considered highly confidential. The latest Incident-Prevention-Investigation revealed that *ServiceAccount* `p.auster` had too broad access to the cluster for some time. This *SA* should've never had access to any *Secrets* in that *Namespace*.

Find out which *Secrets* in *Namespace* `security` this *SA* did access by looking at the Audit Logs under `/opt/course/p2/audit.log`.

Change the password to any new string of only those *Secrets* that were accessed by this *SA*.

> ***NOTE:*** You can use `jq` to render json more readable, like `cat data.json | jq`

Answer:

First we look at the *Secrets* this is about:

```
➜ ssh cks3477

➜ candidate@cks3477:~# k -n security get secret | grep Opaque
kubeadmin-token       Opaque                                1      37m
mysql-admin           Opaque                                1      37m
postgres001           Opaque                                1      37m
postgres002           Opaque                                1      37m
vault-token           Opaque                                1      37m
```

Next we investigate the Audit Log file:

```
➜ candidate@cks3477:~# cd /opt/course/p2

➜ candidate@cks3477:/opt/course/p2$ ls -lh
audit.log

➜ candidate@cks3477:/opt/course/p2$ cat audit.log | wc -l
4448
```

Audit Logs can be huge and it's common to limit the amount by creating an Audit *Policy* and to transfer the data in systems like Elasticsearch. In this case we have a simple JSON export, but it already contains 4448 lines.

We should try to filter the file down to relevant information:

```
➜ candidate@cks3477:/opt/course/p2$ cat audit.log | grep "p.auster" | wc -l
28
```

Not too bad, only 28 logs for *ServiceAccount* `p.auster`.

```
➜ candidate@cks3477:/opt/course/p2$ cat audit.log | grep "p.auster" | grep Secret | wc -l
2
```

And only 2 logs related to *Secrets*...

```
➜ candidate@cks3477:/opt/course/p2$ cat audit.log | grep "p.auster" | grep Secret | grep list | wc -l

0

➜ candidate@cks3477:/opt/course/p2$ cat audit.log | grep "p.auster" | grep Secret | grep get | wc -l

2
```

No list actions, which is good, but 2 get actions, so we check these out:

```
cat audit.log | grep "p.auster" | grep Secret | grep get | jq
```

```
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "level": "RequestResponse",
  "auditID": "74fd9e03-abea-4df1-b3d0-9cfeff9ad97a",
  "stage": "ResponseComplete",
  "requestURI": "/api/v1/namespaces/security/secrets/vault-token",
  "verb": "get",
  "user": {
    "username": "system:serviceaccount:security:p.auster",
    "uid": "29ecb107-c0e8-4f2d-816a-b16f4391999c",
    "groups": [
      "system:serviceaccounts",
      "system:serviceaccounts:security",
      "system:authenticated"
    ]
  },
...
  "userAgent": "curl/7.64.0",
  "objectRef": {
    "resource": "secrets",
    "namespace": "security",
    "name": "vault-token",
    "apiVersion": "v1"
  },
 ...
}
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "level": "RequestResponse",
  "auditID": "aed6caf9-5af0-4872-8f09-ad55974bb5e0",
  "stage": "ResponseComplete",
  "requestURI": "/api/v1/namespaces/security/secrets/mysql-admin",
  "verb": "get",
  "user": {
    "username": "system:serviceaccount:security:p.auster",
    "uid": "29ecb107-c0e8-4f2d-816a-b16f4391999c",
    "groups": [
      "system:serviceaccounts",
      "system:serviceaccounts:security",
      "system:authenticated"
    ]
  },
...
  "userAgent": "curl/7.64.0",
  "objectRef": {
    "resource": "secrets",
    "namespace": "security",
    "name": "mysql-admin",
    "apiVersion": "v1"
  },
...
}
```

There we see that *Secrets* `vault-token` and `mysql-admin` were accessed by `p.auster`. Hence we change the passwords for those.

```
➜ candidate@cks3477:/opt/course/p2$ echo new-vault-pass | base64
bmV3LXZhdWx0LXBhc3MK

➜ candidate@cks3477:/opt/course/p2$ k -n security edit secret vault-token

➜ candidate@cks3477:/opt/course/p2$ echo new-mysql-pass | base64
bmV3LW15c3FsLXBhc3MK

➜ candidate@cks3477:/opt/course/p2$ k -n security edit secret mysql-admin
```

Audit Logs ftw.

By running `cat audit.log | grep "p.auster" | grep Secret | grep password` we can see that passwords are stored in the Audit Logs, because they store the complete content of *Secrets*. It's never a good idea to reveal passwords in logs. In this case it would probably be sufficient to only store Metadata level information of *Secrets* which can be controlled via a Audit *Policy*.

Preview Question 3

Solve this question on: `ssh cks8930`

A security scan result shows that there is an unknown miner process running on one of the *Nodes* in this cluster.

The report states that the process is listening on port 6666.

Kill the process and delete the binary.

Answer:

We have a look at existing *Nodes*:

```
➜ ssh cks8930

➜ candidate@cks8930:~# k get node
NAME            STATUS   ROLES           AGE   VERSION
cks8930         Ready    control-plane   26m   v1.29.5
cks8930-node1   Ready    <none>          26m   v1.29.5
```

First we check the master:

```
➜ candidate@cks8930:~# sudo -i

➜ root@cks8930:~# netstat -plnt | grep 6666

➜ root@cks8930:~#
```

Doesn't look like any process listening on this port. So we check the worker:

```
➜ root@cks8930:~# ssh cks8930-node1

➜ root@cks8930-node1:~# netstat -plnt | grep 6666
tcp6       0      0 :::6666                 :::*                    LISTEN      8198/system-atm
```

There we go! We could also use `lsof`:

```
➜ root@cks8930-node1:~# lsof -i :6666
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
system-at 8198 root    3u  IPv6  39776      0t0  TCP *:6666 (LISTEN)
```

Before we kill the process we can check the magic `/proc` directory for the full process path:

```
➜ root@cks8930-node1:~# ls -lh /proc/8198/exe
lrwxrwxrwx 1 root root 0 Sep  8 14:44 /proc/8198/exe -> /usr/bin/system-atm
```

So we finish it:

```
➜ root@cks8930-node1:~# kill -9 8198

➜ root@cks8930-node1:~# rm /usr/bin/system-atm
```

Done.

CKS Tips Kubernetes 1.31

In this section we'll provide some tips on how to handle the CKS exam and browser terminal.

Knowledge

Pre-Knowledge

You should have your CKA knowledge up to date and be fast with kubectl, so we suggest to do:

- Study all scenarios on https://killercoda.com/killer-shell-cka

Knowledge

- Study all topics as proposed in the curriculum till you feel comfortable with all.
- Study all scenarios on https://killercoda.com/killer-shell-cks
- Read the free Sysdig Kubernetes Security Guide
- Also a nice read (though based on outdated k8s version) is the Kubernetes Security book by Liz Rice
- Check out the Cloud Native Security Whitepaper
- Great repository with many tips and sources: walidshari

Approach

- Do 1 or 2 test session with this CKS Simulator. Understand the solutions and maybe try out other ways to achieve the same thing.
- Be fast and breath `kubectl`

Content

- Be comfortable with changing the kube-apiserver in a kubeadm setup
- Be able to work with AdmissionControllers
- Know how to create and use the ImagePolicyWebhook
- Know how to use opensource tools Falco, Sysdig, Tracee, Trivy

CKS Exam Info

Read the Curriculum

https://github.com/cncf/curriculum

Read the Handbook

https://docs.linuxfoundation.org/tc-docs/certification/lf-handbook2

Read the important tips

https://docs.linuxfoundation.org/tc-docs/certification/important-instructions-cks

Read the FAQ

https://docs.linuxfoundation.org/tc-docs/certification/faq-cka-ckad-cks

Kubernetes documentation

Get familiar with the Kubernetes documentation and be able to use the search. Allowed resources are:

- https://kubernetes.io/docs
- https://kubernetes.io/blog
- https://falco.org/docs
- https://kubernetes-sigs.github.io/bom/cli-reference
- https://etcd.io/docs
- https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration
- https://docs.cilium.io/en/stable

> Verify the list here

CKS clusters

In the CKS exam you'll get access to as many clusters as you have questions, each will be solved in its own cluster. This is great because you cannot interfere with other tasks by breaking one. Every cluster will have one controlplane and one worker node.

The Test Environment / Browser Terminal

You'll be provided with a browser terminal which uses Ubuntu/Debian. The standard shells included with a minimal install will be available, including bash.

Laggin

There could be some lagging, definitely make sure you are using a good internet connection because your webcam and screen are uploading all the time.

Kubectl autocompletion and commands

Autocompletion is configured by default, as well as the `k` alias source and others:

`kubectl` with `k` alias and Bash autocompletion

`yq` and `jq` for YAML/JSON processing

`tmux` for terminal multiplexing

`curl` and `wget` for testing web services

`man` and man pages for further documentation

Copy & Paste

There could be issues copying text (like pod names) from the left task information into the terminal. Some suggested to "hard" hit or long hold `Cmd/Ctrl+C` a few times to take action. Apart from that copy and paste should just work like in normal terminals.

Score

There are 15-20 questions in the exam. Your results will be automatically checked according to the handbook. If you don't agree with the results you can request a review by contacting the Linux Foundation Support.

Notepad & Skipping Questions

You have access to a simple notepad in the browser which can be used for storing any kind of plain text. It might makes sense to use this for saving skipped question numbers. This way it's possible to move some questions to the end.

Servers

Each question needs to be solved on a specific instance other than your main terminal. You'll need to connect to the correct instance via ssh, the command is provided before each question.

PSI Bridge

Starting with PSI Bridge:

- The exam will now be taken using the PSI Secure Browser, which can be downloaded using the newest versions of Microsoft Edge, Safari, Chrome, or Firefox
- Multiple monitors will no longer be permitted
- Use of personal bookmarks will no longer be permitted

The new ExamUI includes improved features such as:

- A remote desktop configured with the tools and software needed to complete the tasks
- A timer that displays the actual time remaining (in minutes) and provides an alert with 30, 15, or 5 minute remaining
- The content panel remains the same (presented on the Left Hand Side of the ExamUI)

Read more here.

Terminal Handling

Bash Aliases

In the real exam, each question has to be solved on a different instance to which you connect via ssh. This means it's not advised to configure bash aliases because they wouldn't be available on the instances accessed by ssh.

Be fast

Use the `history` command to reuse already entered commands or use even faster history search through **Ctrl r** .

If a command takes some time to execute, like sometimes `kubectl delete pod x`. You can put a task in the background using **Ctrl z** and pull it back into foreground running command `fg`.

You can delete *pods* fast with:

```
k delete pod x --grace-period 0 --force
```

Vim

Be great with vim.

Settings

In case you face a situation where vim is not configured properly and you face for example issues with pasting copied content you should be able to configure via `~/.vimrc` or by entering manually in vim settings mode:

```
set tabstop=2
set expandtab
set shiftwidth=2
```

The `expandtab` make sure to use spaces for tabs.

Note that changes in `~/.vimrc` will not be transferred when connecting to other instances via ssh.

Toggle vim line numbers

When in `vim` you can press **Esc** and type `:set number` or `:set nonumber` followed by **Enter** to toggle line numbers. This can be useful when finding syntax errors based on line - but can be bad when wanting to mark&copy by mouse. You can also just jump to a line number with **Esc** `:22` + **Enter**.

Copy&Paste

Get used to copy/paste/cut with vim:

```
Mark lines: Esc+V (then arrow keys)
Copy marked lines: y
Cut marked lines: d
Past lines: p or P
```

Indent multiple lines

To indent multiple lines press **Esc** and type `:set shiftwidth=2`. First mark multiple lines using `Shift v` and the up/down keys. Then to indent the marked lines press `>` or `<`. You can then press `.` to repeat the action.