Lesson Plan

Managing Trust Across Teams

Summary

1. Lab Setup
2. Lazy Policy
3. Lockdown Cluster Egress
4. Grant Selective Cluster Egress
5. Managing Trust Across Teams

Lab Setup

This lab assumes you already have the "Yet Another Online Bank" (yaobank) sample application and the network policies from the previous Policy Fundamentals module installed. If you don't already have these in place, you'll need to go back and install them now.

For reference, this is architecture of YAOBank:

If you are not already on host1, you can enter host1 by using the multipass shell command.

```
multipass shell host1
```

Lazy Policy

In the K8s Network Policy deployed in the Policy fundamentals lab, the developer's policy allows all pod egress, even to the Internet.

If you've not already done so in this shell, find the customer pod name and store it in an environment variable to simplify future commands.

```
CUSTOMER_POD=$(kubectl get pods -n yaobank -l app=customer -o name)
```

Note that the CUSTOMER_POD environment variable only exists within your current shell, so if you exit that shell you must set it again in your new shell using the same command as above.

Now exec into the customer pod running bash to give us a command prompt within the pod:

```
kubectl exec -it $CUSTOMER_POD -n yaobank -c customer -- /bin/bash
```

Try to access the internet:

```
ping -c 3 8.8.8.8
curl --connect-timeout 3 -I www.google.com
```

This succeeds, i.e., your pods have unfettered Internet access because the developer hasn't properly defined their policy. This is not a good security posture!

Return to the host shell command line:

```
exit
```

Lockdown Cluster Egress

We'll create a Calico GlobalNetworkPolicy to restrict egress to the Internet to only pods that have a ServiceAccount that is labeled "internet-egress = allowed".

Examine the network policy

Examine the policy. While Kubernetes network policies only have Allow rules, Calico network policies also support Deny rules. As this policy has Deny rules in it, it is important that we set its precedence higher than the lazy developer's Allow rules in their Kubernetes policy. To do this we specify order value of 600 in this policy, which gives this higher precedence than Kubernetes Network Policy (which does not have the concept of setting policy precedence, and is assigned a

fixed order value of 1000 by Calico - i.e, policy order 600 gets precedence over policy order 1000).

Apply the network policy

```
cat <<EOF | calicoctl apply -f -
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: egress-lockdown
spec:
  order: 600
  namespaceSelector: has(projectcalico.org/name) && projectcalico.org/name
not in {"kube-system", "calico-system"}
  serviceAccountSelector: internet-egress not in {"allowed"}
  types:
  - Egress
  egress:
    - action: Deny
      destination:
        notNets:
          - 10.0.0.0/8
          - 172.16.0.0/12
          - 192.168.0.0/16
          - 198.18.0.0/15
EOF
```

Now let's try to access the internet again from the customer pod.

```
kubectl exec -ti $CUSTOMER_POD -n yaobank -c customer -- /bin/bash
ping -c 3 8.8.8.8
curl --connect-timeout 3 -I www.google.com
```

These commands should fail - pods are now restricted from accessing the internet, even if a developer included lazy egress rules to their Kubernetes network policies. You may need to terminate the command with CTRL-C. Then don't forget to exit from the pod to get back to your host terminal.

```
exit
```

Granting selective Cluster Egress

Now imagine there was a legitimate reason to allow connections from the customer pod to the internet. As we used a Service Account label selector in our egress policy rules, we can enable this by adding the appropriate label to the pod's Service Account.

Add "internet-egress=allowed" to the Service Account

```
kubectl label serviceaccount -n yaobank customer internet-egress=allowed
```

Verify the pod can now access the internet

```
kubectl exec -ti $CUSTOMER_POD -n yaobank -c customer -- /bin/bash
ping -c 3 8.8.8.8
curl --connect-timeout 3 -I www.google.com
exit
```

Now you should find that the customer pod is allowed Internet Egress, but other pods (like Summary and Database) are not.

Managing trust across teams

There are many ways of dividing responsibilities across teams using Kubernetes RBAC.

Imagine the following:

The secops team is responsible for creating Namespaces and Services accounts for dev teams. They are also responsible for writing Calico Global Network Policies to define the cluster's overall security posture. Kubernetes RBAC is set up so that only they can do this.

Dev teams are given Kubernetes RBAC permissions to create pods and Kubernetes Network Policies in their Namespaces, and they can use, but not modify any Service Account in their Namespaces.

In this scenario, the secops team can control which teams should be allowed to have pods that access the internet. If a dev team is allowed to have pods that access the internet then the dev team can choose which of their pods access the internet by using the appropriate Service Account.

This is just one way of dividing responsibilities across teams. Pods, Namespaces, and Service Accounts, Kubernetes Network Policies, Calico Network Policies, and Calico Global Network Policies, all have separate RBAC controls. You can choose which teams have access to each, and construct a range of different trust boundaries and to support your preferred level of shift-left security practices (i.e. delegating different levels of security trust to dev or other teams).