Lesson Plan

Encrypting Data in Transit

Summary

1. Introduction to Encryption [video]
2. Lab Setup
3. Enabling Encryption
4. Disabling Encryption

Lab Setup

This lab can be run on your cluster as is with Calico installed.

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

```
multipass shell host1
```

Enabling Encryption

Calico makes it easy to encrypt on the wire in-cluster pod traffic in a Calico cluster using WireGuard. WireGuard utilizes state-of-the-art cryptography and aims to be faster, simpler, leaner, than alternative encryption techniques such as IPsec.

Calico handles all the configuration of WireGuard for you to provide full mesh encryption across all the nodes in your cluster. WireGuard is included in the latest Linux kernel versions by default, and if running older Linux versions you can easily load it as a kernel module. (Note that if you have some nodes that don't have WireGuard support, then traffic to/from those specific nodes will be unencrypted.)

While WireGuard performs well, there is still an overhead associated with encryption. As-such, at the time of this writing: this is an optional feature that is not enabled by default.

Let's start by enabling encryption:

```
calicoctl patch felixconfiguration default --type='merge' -p
```

```
'{"spec":{"wireguardEnabled":true}}'
```

It's as simple as that! Within a few moments WireGuard encryption will be in place on all the nodes in the cluster.

Inspecting WireGuard status

Every node that is using WireGuard encryption generates its own public key. You check the node status using calicoctl. If WireGuard is active on the node you will see the public key it is using in the status section.

```
calicoctl get node node1 -o yaml
```

Example output:

```
apiVersion: projectcalico.org/v3
kind: Node
metadata:
  annotations:
    projectcalico.org/kube-labels:
'{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/instance-type":"k3s
","beta.kubernetes.io/os":"linux","k3s.io/hostname":"node1","k3s.io/interna
l-ip":"198.19.0.2","kubernetes.io/arch":"amd64","kubernetes.io/hostname":"n
ode1","kubernetes.io/os":"linux","node.kubernetes.io/instance-type":"k3s"}'
  creationTimestamp: "2020-10-20T23:33:09Z"
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/instance-type: k3s
    beta.kubernetes.io/os: linux
    k3s.io/hostname: node1
    k3s.io/internal-ip: 198.19.0.2
    kubernetes.io/arch: amd64
    kubernetes.io/hostname: node1
    kubernetes.io/os: linux
    node.kubernetes.io/instance-type: k3s
  name: node1
  resourceVersion: "8760"
  uid: 66f16def-8f76-46dc-90e3-491bfc75dc9b
spec:
  bgp:
    ipv4Address: 198.19.0.2/20
    ipv4IPIPTunnelAddr: 198.19.22.128
  orchRefs:
```

```
  - nodeName: node1
    orchestrator: k8s
  wireguard:
    interfaceIPv4Address: 198.19.22.131
status:
  podCIDRs:
  - 198.19.17.0/24
  wireguardPublicKey: An4UT4PR9XzGgJ5df452Dw034q1SfXZOI1Dp2ebUUWQ=
```

Let's ssh to a node directly and inspect the interfaces present.

```
ssh node1
```

Once we've entered the node, we can inspect the WireGuard interface:

```
ip addr | grep wireguard
```

Example output:

```
10: wireguard.cali: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1400 qdisc noqueue
state UNKNOWN group default qlen 1000
    inet 198.19.22.131/32 brd 198.19.22.131 scope global wireguard.cali
```

Note that the WireGuard interface needs an IP address. Calico automatically allocates this from the default IP Pool.

Finally let's exit the node returning to host1.

```
exit
```

Disabling Encryption

Switching off encryption is as simple as switching it on. Let's try that now.

```
calicoctl patch felixconfiguration default --type='merge' -p
'{"spec":{"wireguardEnabled":false}}'
```

Within a few moments encryption will be disabled. This can be validated by looking at the node once again and seeing that the wireguard public key has been removed from the specification.

```
calicoctl get node node1 -o yaml
```

Example output:

```
apiVersion: projectcalico.org/v3
```

```
kind: Node
metadata:
  annotations:
    projectcalico.org/kube-labels:
'{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/instance-type":"k3s
","beta.kubern
etes.io/os":"linux","k3s.io/hostname":"node1","k3s.io/internal-ip":"198.19.
0.2","kubernetes.io/arch":"amd64","kubernetes.io/
hostname":"node1","kubernetes.io/os":"linux","node.kubernetes.io/instance-t
ype":"k3s"}'
  creationTimestamp: "2020-10-20T23:33:09Z"
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/instance-type: k3s
    beta.kubernetes.io/os: linux
    k3s.io/hostname: node1
    k3s.io/internal-ip: 198.19.0.2
    kubernetes.io/arch: amd64
    kubernetes.io/hostname: node1
    kubernetes.io/os: linux
    node.kubernetes.io/instance-type: k3s
  name: node1
  resourceVersion: "9067"
  uid: 66f16def-8f76-46dc-90e3-491bfc75dc9b
spec:
  bgp:
    ipv4Address: 198.19.0.2/20
    ipv4IPIPTunnelAddr: 198.19.22.128
  orchRefs:
  - nodeName: node1
    orchestrator: k8s
status:
  podCIDRs:
  - 198.19.17.0/24
```

Now that we've explored how easy it is to enable full cluster encryption with Calico we can proceed to the next module.