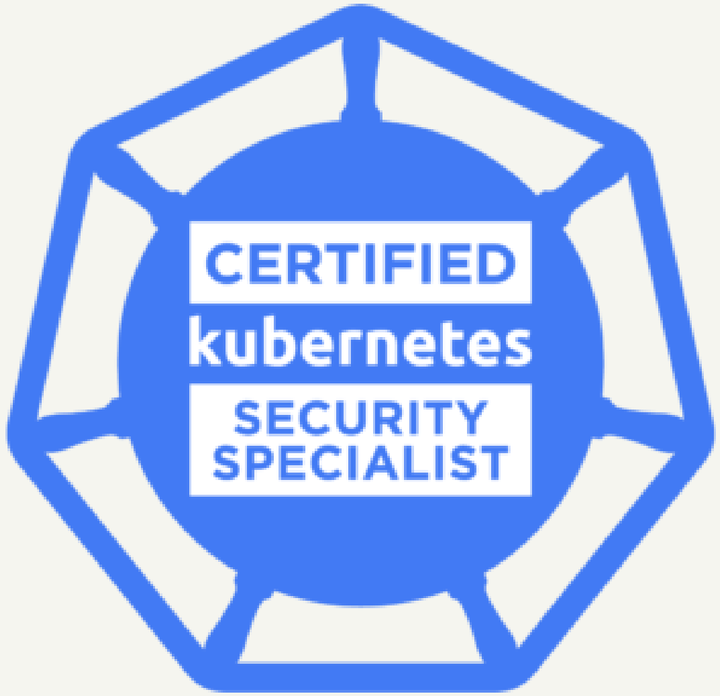


ch #4
Securing kube-apiserver [part three]

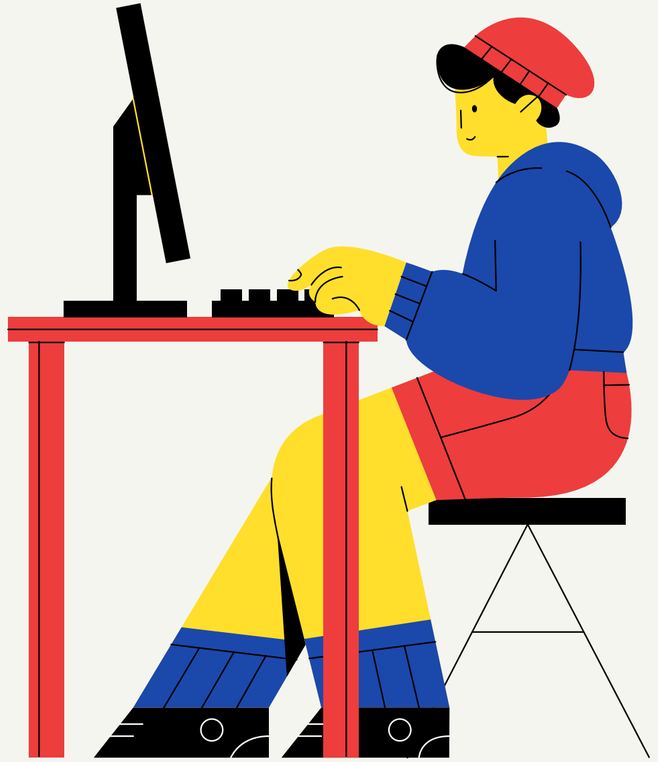
LFS 260

Identity & Access Management

Unlike many cluster solutions, Kubernetes does not have an ongoing session with an associated user or identity. Every API call is unique and must be independently validated.

Nor does Kubernetes have users, but relies on the operating system or some tool such as Keycloak or Active Directory to provide user information and more.

Some clusters may have add-on access control. Amazon Identity and Access Management (IAM) allows for granular authentication and authorization for multiple AWS services.



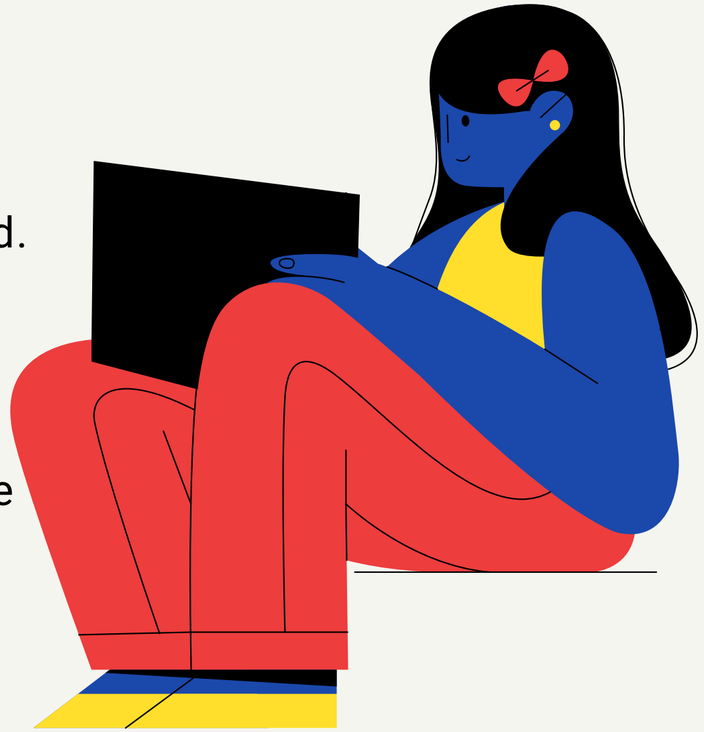
Persistent State from etcd

Special care must be taken to ensure the integrity and safety of etcd. The entire persistent state of the cluster is kept in etcd.

In addition, secrets (which, by nature, should be protected) and configMaps (which are used to configure much of the cluster) must not be compromised.

If an attacker were to update either of them, it would be ingested into the pod, potentially bypassing local security settings. All other infrastructure pods request the status and spec via kube-apiserver, and, as a result, can be replaced or rebuilt easily.

If your database is corrupted or becomes unavailable, orchestration no longer can function. The only agent communicating with etcd should be kube-apiserver. You may remember the settings for TLS communication between them mentioned earlier.



Encrypt Data at Rest

In order to encrypt secrets at rest, instead of just a symmetrical base64 encoding, the kube-apiserver must be updated to use a newly-configured encryption provider configuration. Once the kube-apiserver has been updated and restarted, every secret must be re-written as encryption happens during the write.

When updating keys or using a different type of encryption, add the new key or setting to the top of the configuration file. Leave the previous keys after. They will be used when the existing encrypted secret is read, and the first listed, newly-created key will be used to encrypt and write that secret to persistent storage.

When this is configured typically, the raw encryption key is kept as part of the *EncryptionConfig*. Use of a separate key management system (KMS) removes the local keys and instead keeps them in a distinct and, hopefully more secure, location. Several cloud providers and a couple of CNCF projects can provide KMS functionality.

Encryption Providers

identity

There is no encryption with this provider, resources are saved as passed. It is often kept in the list to allow for reading unencrypted resources. If the key is listed first, it will be used and, when any resource is created, it will not be encrypted.

aescbc

This is the suggested provider setting. One of the slower providers, it offers strong encryption using a 32-byte key. It represents a combination of the Advanced Encryption Standard (AES) Cipher Algorithm in Cipher Block Chaining (CBC) Mode Public- Key Cryptography Standards (PKCS). You can read more about it in RFC 3602.

Encryption Providers

secretbox

This is a newer provider, which may prevent its use by organizations which require well-known and trusted code. It uses XSalsa20 and Poly1305 encryption. Also 32-bytes, and faster than aescbc.

aesgcm

While this is the fastest encryption provider, it requires a key rotation every two hundred thousand writes. It offers 16 and 32-byte keys.

kms

This is the provider to use when using an external tool management system. It uses a new Data Encryption Key (DEK) for every encryption along with AES-CBC. A KMS must be enabled prior to using this provider. Using this prevents the keys from being stored in a raw manner locally.

Encrypting Policy File

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
name: configureKeys
resources:
  - resources:
    - secrets
    providers:
    - aescbc:
        keys:
        - name: encKey
          secret: Ezqai0SIGChGOwf0VdbkFtYPUM2EYf1TAAQbDrfizJQ=
    - identity: {}
```

You can encrypt many objects in the etcd cluster; this configuration narrows it down to only to secrets resources.You can have several providers listed, including the same provider more than once. The first key listed is used whenever a resource is persisted to etcd. All the following keys would be kept in case you want to read a previously encrypted resource. If you delete the old keys prior to rewriting the resource, you will not be able to access those resources again.

Encrypting Secrets

Lets create secret. Checking that secrets are only encoded, not encrypted.

```
root@master:~# kubectl create secret generic first -n default --from-literal=somekey=findme
secret/first created
root@master:~#
root@master:~# kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-l4d2j   kubernetes.io/service-account-token   3      10d
first                 Opaque                                1      4s
root@master:~# kubectl get secret firts -o yaml
Error from server (NotFound): secrets "firts" not found
root@master:~# kubectl get secret first -o yaml
apiVersion: v1
data:
  somekey: ZmluZG1l
kind: Secret
metadata:
```

see the value is encoded

Encrypting Secrets

Lets create secret. Checking that secrets are only encoded, not encrypted.

```
root@master:~#
root@master:~# echo 'ZmluZG1l' | base64 --decode
findmeroot@master:~#
root@master:~#
```

Encrypting Secrets

Get the values of pki files

```
root@master:~#
root@master:~# grep etcd /etc/kubernetes/manifests/kube-apiserver.yaml
    - --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
root@master:~#
```

Encrypting Secrets

Run a command inside the etcd container to view the secret. You may want to ensure the location of the pki files to copy and paste the long command.

```
root@master:~#
root@master:~# kubectl -n kube-system exec -it etcd-master.example.com -- sh -c \
> "ETCDCTL_API=3 ETCDCTL_CACERT=/etc/kubernetes/pki/etcd/ca.crt ETCDCTL_CERT=/etc/kubernetes/pki/
etcd/server.crt ETCDCTL_KEY=/etc/kubernetes/pki/etcd/server.key etcdctl --endpoints=https://127.0
.0.1:2379 get /registry/secrets/default/first"
/registry/secrets/default/first
k8s


v1Secret
first"default"*$37726874-7867-469f-b011-83af0d2356df2z
kubectl-createUpdatevFieldsV1:0
.{"f:data":{".":{},"f:somekey":{}},"f:type":{}}
somekeyfindmeOpaque'
root@master:~#
```

Encrypting Secrets

Generate a random key to use for encryption using base64

.

```
root@master:~#
root@master:~# head -c 32 /dev/urandom | base64
aLrFcJJqOKwea/hyruTTbHPIH9hayEjJ0ZBrqIFRlsM=
root@master:~#
root@master:~#
```

Encrypting Secrets

Create an encryption configuration yaml file. Add the encoded string created in the previous command

```
root@master:~#
root@master:~# cat encryptionconfig.yaml
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
name: newsetup
resources:
  - resources:
    - secrets
    providers:
    - aescbc:
        keys:
        - name: firstkey
          secret: aLrFcJJqOKwea/hyruTTbHPIH9hayEjJ0ZBrqIFRlsM=
    - identity: {}
root@master:~#
```

Encrypting Secrets

In order for the configuration file to be seen by the kube-apiserver container it must be in a directory mounted to the pod. While we could create a new volume, for now let's copy the file to a known and typically protected directory.

```
root@master:~#
root@master:~# cp encryptionconfig.yaml /etc/kubernetes/pki/
root@master:~#
root@master:~# ll /etc/kubernetes/pki/encryptionconfig.yaml
-rw-r--r-- 1 root root 274 Mar 12 18:30 /etc/kubernetes/pki/encryptionconfig.yaml
root@master:~#
root@master:~#
```

Encrypting Secrets

Edit the manifest file for kube-apiserver and add the **--encryption-provider-config=**

```
- --service-cluster-ip-range=10.96.0.0/12
- --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
- --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
- --encryption-provider-config=/etc/kubernetes/pki/encryptionconfig.yaml ## add this line
image: k8s.gcr.io/kube-apiserver:v1.20.4
imagePullPolicy: IfNotPresent
livenessProbe:
```

Encrypting Secrets

Create another secret named second and verify you can see it using kubectl

.

```
root@master:~#
root@master:~# kubectl create secret generic second -n default --from-literal=anotherkey=hidden
secret/second created
root@master:~#
root@master:~#
```

Encrypting Secrets

Create another secret named second and verify you can see it using kubectl

.

```
root@master:~#
root@master:~# kubectl -n kube-system exec -it etcd-master.example.com -- sh -c "ETCDCTL_API=3 ET
CDCTL_CACERT=/etc/kubernetes/pki/etcd/ca.crt ETCDCTL_CERT=/etc/kubernetes/pki/etcd/server.crt ETC
DCTL_KEY=/etc/kubernetes/pki/etcd/server.key etcdctl --endpoints=https://10.0.0.100:2379 get /reg
istry/secrets/default/second"
/registry/secrets/default/second
k8s


v1Secret
$second"default"*$d3bab6db-5983-4641-b92c-6e0213d92eb52z
kubectl-createUpdatevFieldsV1:3
1{"f:data":{".":{},"f:anotherkey":{}},"f:type":{}}

anotherkeyhiddenOpaque
root@master:~#
```

Encrypting Secrets

Re-write the all the secrets and verify first is encrypted, along with the rest.

```
root@master:~#
root@master:~# kubectl get secrets --all-namespaces -o json | kubectl replace -f -
secret/default-token-l4d2j replaced
secret/first replaced
secret/four replaced
secret/second replaced
secret/third replaced
secret/default-token-6559g replaced
secret/default-token-v4qj5 replaced
secret/attachdetach-controller-token-m8w9t replaced
secret/bootstrap-signer-token-f8dnk replaced
secret/calico-kube-controllers-token-httdp replaced
secret/calico-node-token-8jx6r replaced
secret/certificate-controller-token-crz84 replaced
```