Extending Kubernetes with Storage Transformers

Andrew Lytvynov awly@google.com
KubeCon China 2019
Agenda

1. What are Transformers
2. Why do we need them
3. How do you implement them
4. How do we secure them
RPC vs. Storage layers of kube-apiserver
Extensibility at the RPC layer

RPC
Webhook
Storage
kube-apiserver

Admission Webhook Server
Extensibility at the Storage layer

RPC

Storage
Transformer

kube-apiserver

KMS Plugin
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: MWYyZDFIM2Rm
Convention

Transformers

apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: MWYyZDFIM2Rm

Webhooks

apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: MWYyZDFIM2Rm
Why a Kubecon talk about Transformers

- Explain the feature
- Increase contribution
- Share lessons learned
- Spark new ideas
Motivating Problem - Encrypting Secrets at Rest

A default OSS Kubernetes setup is not encrypted by default. Secrets are stored in plaintext.
kube-apiserver

master

etcd
kube-apiserver

etcd

master
Offline attacks

Master

- etcd
- kube-apiserver

Sensitive Data
Offline attacks

Master

etcd

kube-apiserver

Sensitive Data

attacker
Offline attacks

Master
- etcd
- kube-apiserver

Backup Server
- SECRET

Sensitive Data

attacker
Offline attacks

Master
- kube-apiserver
- etcd

Backup Server
- SECRET

Sensitive Data

attacker
Demo: fancy tools?
Implementing Storage Transformers

https://github.com/awly/kubernetes/commits/kubecon-china-transformers
Step #1: Implement Transformer Interface

[k8s.io/apiserver/pkg/storage/value/encrypt/mytransformer/](https://github.com/awly/kubernetes/commit/214b616b152aa22a851b37923a7314de834dc104)

type Transformer interface {
  TransformFromStorage(data []byte, context Context) (out []byte, stale bool, err error)
  TransformToStorage(data []byte, context Context) (out []byte, err error)
}
Step #2: Create your YAML config structure

```yaml
kind: EncryptionConfiguration
apiVersion: apiserver.config.k8s.io/v1
resources:
  - resources:
    - secrets
  providers:
    - myProvider:
      key: key1
      field2: value2
```

https://github.com/awly/kubernetes/commit/d1c32f5766931f08567dda318b0b30eb8390fd60
Step #2: Create your YAML config structure

k8s.io/apiserver/pkg/apis/config/types.go
k8s.io/apiserver/pkg/apis/config/v1/types.go

type MyConfiguration struct {
    Key Key `json:"key"
    Field2 Type2 `json:"field2"
    ...
}

https://github.com/awly/kubernetes/commit/d1c32f5766931f08567dda318b0b30eb8390fd60
Step #3: Add your type to ProviderConfiguration

k8s.io/apiserver/pkg/apis/config/types.go
k8s.io/apiserver/pkg/apis/config/v1/types.go

```go
type ProviderConfiguration struct {
    AESGCM  *AESConfiguration
    AESCBC  *AESConfiguration
    Secretbox *SecretboxConfiguration
    Identity *IdentityConfiguration
    KMS     *KMSConfiguration
    MyProvider *MyConfiguration
}
```

https://github.com/awly/kubernetes/commit/39397c8bb87e4b17904d64820355ebe9bb6e6c45
Prefix Transformer

$ cat ${ETCD_DATA} | grep -A 2 -a db-password
...
  {2e+1^f)'=\[lXr;%v'}efault/db-password33
* k8s:enc:my:v1:key1:v;c[yb:5;;PzV|$\!x\@\w5;Q&eXTHIrQ\_V@
J`(4_
...
...
Prefix Transformer

type PrefixTransformer struct {
    Prefix []byte
    Transformer Transformer
}
Step #4: Define your prefix

k8s.io/apiserver/pkg/server/options/encryptionconfig/config.go

const (
    aesCBCTransformerPrefixV1 = "k8s:enc:aescbc:v1:"
    aesGCMTransformerPrefixV1 = "k8s:enc:aesgcm:v1:"
    secretboxTransformerPrefixV1 = "k8s:enc:secretbox:v1:"
    kmsTransformerPrefixV1 = "k8s:enc:kms:v1:"
    myTransformerPrefixV1 = "k8s:enc:my:v1:"
)

https://github.com/awly/kubernetes/commit/cf128fc92e5e66582bc02c6992df065b91e6c582
Step #5: Add Init logic for your transformer

k8s.io/apiserver/pkg/server/options/encryptionconfig/config.go

```go
func GetMyPrefixTransformer(config *apiserverconfig.MyConfiguration, prefix string) (value.PrefixTransformer, error) {

    // 1. Validate and parse fields of config.
    // 2. Create an instance of MyTransformer.

    return value.PrefixTransformer{
        Transformer: myTransformer,
        Prefix:     []byte(prefix),
    }, nil
}
```

https://github.com/awly/kubernetes/commit/a82c659f7f13450b16bf3f47c17f6566495c43c5
Demo: SM4 transformer
It works, but...

1. Key rotation is manual and requires kube-apiserver restart
2. Key is in plaintext on disk
Envelope Transformers
Envelope encryption

Data

Data encryption key
DEK

Key encryption key
KEK
Envelope encryption

Data: $\{\text{SECRET}\}_{\text{DEK}}$

Data encryption key: $\{\text{DEK}\}_{\text{KEK}}$

Key encryption key: $\{\text{DEK}\}_{\text{KEK}}$

Envelope: $\{\text{SECRET}\}_{\text{DEK}} + \{\text{DEK}\}_{\text{KEK}}$
Envelope encryption

Data
\{SECRET\}_{DEK}

Data encryption key
\{DEK\}_{KEK}

Envelope
\{SECRET\}_{DEK} + \{DEK\}_{KEK}

KMS

Key encryption key
Version management

Data store

KMS
Version management

Data store

KMS

KEKv1

Nov 1 - Nov 31
Version management

Data store

\[\{\text{SECRET1}\}_{\text{DEK1}} + \{\text{DEK1}\}_{\text{KEKv1}}\]

KMS

KEKv1

Nov 1 - Nov 31
Version management

Data store

{SECRET1}_{DEK1} + {DEK1}_{KEKv1}

{SECRET2}_{DEK2} + {DEK2}_{KEKv1}

KMS

KEKv1

Nov 1 - Nov 31
Version management

Data store

\[
\{\text{SECRET1}\}_{\text{DEK1}} + \{\text{DEK1}\}_{\text{KEKv1}} \\
\{\text{SECRET2}\}_{\text{DEK2}} + \{\text{DEK2}\}_{\text{KEKv1}}
\]

KMS

- KEKv1
- KEKv2

- Nov 1 - Nov 30
- Dec 1 - Jan 31
Version management

Data store

\{\text{SECRET1}\}_{\text{DEK1}} + \{\text{DEK1}\}_{\text{KEKv1}}
\{\text{SECRET2}\}_{\text{DEK2}} + \{\text{DEK2}\}_{\text{KEKv1}}

KMS

- KEKv1
- KEKv2
- KEKv3

Timeline:
- Nov 1 - Nov 30
- Dec 1 - Dec 21
- Jan 1 - Jan 30
Version management

Data store

\[
\{\text{SECRET1}\}_{\text{DEK1}} + \{\text{DEK1}\}_{\text{KEKv1}} \\
\{\text{SECRET2}\}_{\text{DEK2}} + \{\text{DEK2}\}_{\text{KEKv1}} \\
\{\text{SECRET3}\}_{\text{DEK3}} + \{\text{DEK3}\}_{\text{KEKv3}}
\]

KMS

- KEKv1
- KEKv2
- KEKv3

\[
\text{Nov 1 - Nov 30} \quad \text{Dec 1 - Dec 31} \quad \text{Jan 1 - Jan 31}
\]
Version management

Data store

\[ \text{SECRET1}_{DEK1} + \text{DEK1}_{KEKv1} \]
\[ \text{SECRET2}_{DEK4} + \text{DEK4}_{KEKv3} \]
\[ \text{SECRET3}_{DEK3} + \text{DEK3}_{KEKv3} \]

KMS

- KEKv1
- KEKv2
- KEKv3

Timeline:
- Nov 1 - Nov 30
- Dec 1 - Dec 31
- Jan 1 - Jan 31
Implementing Envelope Transformer
func NewEnvelopeTransformer(
envelopeService Service,
cacheSize int,
baseTransformerFunc func(cipher.Block) value.Transformer,
) (value.Transformer, error)
type Service interface {
    Decrypt(data []byte) ([]byte, error)
    Encrypt(data []byte) ([]byte, error)
}
KMS Plugins
KMS encryption configuration

```yaml
kind: EncryptionConfiguration
apiVersion: apiserver.config.k8s.io/v1
resources:
  - resources:
    - secrets
providers:
  - kms:
    name: myKmsPlugin
    endpoint: unix:///var/kms-plugin/kms-socket.sock
    cachesize: 100
```
service KeyManagementService {
  // Version returns the runtime name and runtime version of the KMS provider.
  rpc Version(VersionRequest) returns (VersionResponse) {} 
  // Execute decryption operation in KMS provider.
  rpc Decrypt(DecryptRequest) returns (DecryptResponse) {} 
  // Execute encryption operation in KMS provider.
  rpc Encrypt(EncryptRequest) returns (EncryptResponse) {} 
}
Envelope encryption sequence

- Master
  - etcd
  - kube-apiserver
  - kms-plugin
- KMS

SECRET
kube-apiserver generates a DEK
kube-apiserver sends DEK to kms-plugin
kms-plugin forwards to KMS

Master

etcd

kube-apiserver

kms-plugin

SECRET

Encrypt(KEK)

KMS

Encrypt(KEK)
kube-apiserver constructs an envelope

Master

etcd

kms-plugin

Encrypt(DEK) → {DEK}_{KEK} → kube-apiserver

SECRET

Encrypt(DEK) → {DEK}_{KEK} → KMS

KMS

{DEK}_{KEK}
enveloped Secret is saved to etcd

SECRET

Encrypt(KEK)

kms-plugin

DEK

Encrypt(DEK)

kube-apiserver

{DEK}_KEK

{SECRET}_DEK + {DEK}_KEK

Master

etcd

KMS

{DEK}_KEK
Step #6: add configurable DEK type to KMS plugin

Code examples:

```
package types

type KMSConfiguration struct {
    Name string
    CacheSize int32
    Endpoint string
    Timeout *metav1.Duration
    DEKType string
}
```

GitHub commit: https://github.com/awly/kubernetes/commit/1b7fb37bad4ed4e717b14f16e0e26e52d3267662
Step #7: teach KMS plugin about your new DEK type

k8s.io/apiserver/pkg/server/options/encryptionconfig/config.go

```go
func GetPrefixTransformers(config *apiserverconfig.ResourceConfiguration) ([]value.PrefixTransformer, error) {
    ...
    if provider.KMS != nil {
        switch provider.KMS.DEKType {
        case "myType":
            newDEKTransformer = mytransformer.New
dekSize = myKeySize
        }
    }
}
```

https://github.com/awly/kubernetes/commit/93cdb73c430795de4886648a97bd34e26d0c677b
Step #8: choose your KMS provider and plugin

Google Cloud KMS: https://github.com/GoogleCloudPlatform/k8s-cloudkms-plugin/
Microsoft Azure Key Vault: https://github.com/Azure/kubernetes-kms
AWS KMS: https://github.com/kubernetes-sigs/aws-encryption-provider
HashiCorp Vault: https://github.com/oracle/kubernetes-vault-kms-plugin
Threat Model of KMS Plugin
The diagram illustrates the setup of a Kubernetes system with a focus on the last mile security. The components include:

- **Host**: The physical machine where the system runs.
- **Persistent Disk**: Stores the etcd data file and tokens.
- **Master VM**: The virtual machine that runs the kube-api and kms-plugin services.
- **KMS Plugin**: Handles encryption and decryption.
- **KMS**: The key management service that secures the tokens.
- **Etcd Data File**: Contains critical system configurations.
- **Token**: Key material that is used for authentication.

The diagram shows the flow of data and the authentication process, emphasizing the importance of secure token management in the last mile of the Kubernetes deployment.
The diagram illustrates the flow of data and keys within a Kubernetes cluster. The `kms-plugin` interacts with the `kube-api` on the `master VM`, which accesses data from the `persistent disk` through the `etcd` service. The `token` is passed between components to authenticate and secure the communication. The `attacker` is shown as a potential threat, attempting to intercept or manipulate the communication, particularly at the 'the last mile' where the keys are exchanged. The `/offline disk` contains a backup or a version of the `etcd data file` and `token`, suggesting strategies for securing critical data even when the primary system is compromised.
Protecting KMS plugin with Trusted Platform Modules
What’s a Trusted Platform Module (TPM)?

- Crypto coprocessor
- **Protected memory boundary**, outside of kernel reach
- Bound to the host machine

Image source: https://www.gigaparts.com/gigabyte-trusted-platform-module-2-0-gc-tpm2-0-s.html
Seal the credential for KMS in TPM
Seal the credential for KMS in TPM

host

kube-api

kms-plugin

token

e etcd

persistent disk

master vm

e etcd data file

token

TPM

Key

the last mile

kms

attacker

offline image

e etcd data file

token

Seal the credential for KMS in TPM
Seal the credential for KMS in TPM

host
kube-api
kms-plugin
etcd
master vm
token
etcd data file
persistent disk
Key
TPM

attacker

kms

the last mile

offline image
etcd data file
token

Seal the credential for KMS in TPM
Summary

1. Transformers mutate data at etcd boundary
2. Layers of Transformers
   a. built-in storage transformer
   b. envelope transformer
   c. KMS plugin
3. TPMs for last-mile credential protection

Call to action

1. Encrypt your Secrets at rest!
2. Contribute to OSS!
References

1. Turtles All the Way Down, KubeCon China 2019
2. Securing Kubernetes with TPMs, KubeCon EU 2019
3. KMS plugin talk from Next 2019
4. Best practices for writing gRPC services
Backup slides