Turtles all the way down: securely managing Kubernetes secrets with secrets

Alexandr Tcherniakhovski, Google Cloud
Maya Kaczorowski, Google Cloud
Nov 14 2018
Turtles all the way down
Turtles all the way down
Alex Tcherniakhovski
Security Engineer, Google Cloud

Maya Kaczorowsk
Security PM, Google Cloud
@MayaKaczorowski
Protecting secrets
What’s a secret?

Credentials, configurations, API keys, and other small bits of information needed by applications at build or run time.
Why protect secrets?

- Attractive target
  - Controls access or use of sensitive resources
- Common attack vector
  - Checked into Github
  - Accessible by users who shouldn’t have access, e.g., CEO
  - Stored in public storage buckets
Secret management requirements

- **Identity**: Require strong identities and least privilege
- **Auditing**: Verify the use of individual secrets
- **Encryption**: Always encrypt before writing to disk
- **Rotation**: Change a secret regularly in case of compromise
- **Isolation**: Separate where secrets are used vs managed
Encryption at different layers (or turtles)

Recommendation: Use two-layers of encryption, e.g., full-disk & application-layer encryption.
... then tries to decrypt it

Key rotation
“Keys are analogous to the combination of a safe. If a safe combination is known to an adversary, the strongest safe provides no security against penetration. Similarly, poor key management may easily compromise strong algorithms.”

NIST SP 800-57, Recommendation for Key Management
Keys get old
Key rotation

- **Key rotation** is meant to limit the
  - ‘Blast radius’ if a single key is compromised
  - Time available for attempts to penetrate physical, procedural, and logical access
  - Time available for computationally intensive cryptanalytic attacks

- A **cryptoperiod** is the time during which a key is used to encrypt data
Key rotation: cryptoperiod

There are lots of factors that influence the choice of cryptoperiod

From NIST SP 800-57:

- Strength of cryptographic algorithms used
- Implementation
- Operating environment
- Volume of data
- Re-keying method
- Number of key copies
- Personnel turnover
- Threat model
- New and disruptive technologies, e.g., quantum computers
Key rotation: compliance

PCI DSS v3.2.1

3.5 Document and implement procedures to protect keys used to secure stored cardholder data against disclosure and misuse.

3.6 Fully document and implement all key-management processes and procedures for cryptographic keys used for encryption of cardholder data, including the following:

3.6.4 Cryptographic key changes for keys that have reached the end of their cryptoperiod (for example, after a defined period of time has passed and/or after a certain amount of cipher-text has been produced by a given key)
Re-encrypting data is hard.
Envelope encryption
Envelope encryption

Data
Envelope encryption

Data

Data encryption key (DEK)
Envelope encryption

Data

Data encryption key (DEK)

Key encryption key (KEK)
Envelope encryption: benefits

Easier to manage
Envelope encryption: best practices

Managing DEKs:

● Generate DEKs locally
● Use a strong cryptographic algorithm
● For easy access, store the DEK near the data that it encrypts
● Ensure DEKs are encrypted at rest
● Don’t use the same DEK to encrypt data from two different apps/users
● Generate a new DEK every time you write the data. This means you don't need to rotate the DEKs

Managing KEKs:

● Store KEKs centrally
● Set the granularity of the DEKs encrypted based on use case
● Rotate keys regularly, and also after a suspected incident
Kubernetes secrets
Kubernetes secrets

- Secrets are stored in etcd
  - base64 encoded
- A pod can access secrets via the filesystem, as an environment variable, or via Kubernetes API call
- Operations with secrets are audit logged
Kubernetes secrets: 1.7 EncryptionConfig

- Encrypt secrets with a locally managed key
- EncryptionConfig for secrets
- Multiple provider options
  - aesgcm
  - aescbc
  - secretbox
Kubernetes secrets: 1.7 EncryptionConfig

```yaml
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
      - secrets
providers:
  - identity: {}
    - aesgcm:
      keys:
        - name: key1
          secret: c2VjcmV0IGlzIHJnY3VyZQ==
      - aescbc: {}
      - secretbox: {}
```
Kubernetes secrets: **1.10 KMS plugins**

- Encrypt secrets with a locally managed key, which is then encrypted with a centrally managed key
- EncryptionConfig uses aescbc with a KMS provider
- Sidecar pod for the KMS plugin
Terminology and Notation

**DEK**
Data encryption key

**KEK**
Key encryption key

\{SECRET\}_DEK
Secret is encrypted with DEK

\{DEK\}_KEK
DEK is encrypted with KEK

\{SECRET\}_DEK + \{DEK\}_KEK
Envelope

Source for crypto notation:
1.10 Envelope Encryption Sequence

Master

- etcd
- kube-apiserver
- kms-plugin

KMS
1.10 Kube-ApiServer Generates a DEK

Master
- etcd
- kube-apiserver
- kms-plugin

SECRET

KMS

Google Cloud
1.10 Kube-ApiServer Sends DEK to Plugin

Master

etcd

kube-apiserver

Encrypt(DEK)

kms-plugin

SECRET

KMS
1.10 Plugin Forwards to KMS

SECRET

Encrypt(DEK)

kube-apiserver

kms-plugin

Encrypt(DEK)

KMS
1.10 KMS Encrypts a DEK

Master

etcd

kube-apiserver

Encrypt(DEK)

KMS

kms-plugin

SECRET

Encrypt(DEK)

{DEK}^KEK
1.10 Kube-ApiServer Constructs an Envelope
1.10 Enveloped Secret is saved to ETCD

The Secret is encrypted with a DEK and then encrypted with a KEK. The encrypted DEK is stored in the KMS, the KEK is stored in etcd, and the Secret is stored in the KMS.
1.10 KMS plugin version management

etcd

KMS
1.10 KMS plugin version management

etcd

KMS

KEKv1

Nov 12-Dec 12
1.10 KMS plugin version management

etcd

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

KMS

KEKv1

Nov 12-Dec 12
1.10 KMS plugin version management

```
{SECRET1}^{DEK1} + {DEK1}^{KEKv1}
{SECRET2}^{DEK2} + {DEK2}^{KEKv1}
```

Nov 12-Dec 12
1.10 KMS plugin version management

etcd

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

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

KMS

KEKv1

KEKv2

Nov 12-Dec 12

Dec 12 - Jan 11
1.10 KMS plugin version management

etcd

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

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

KMS

- KEKv1
- KEKv2
- KEKv3

- Nov 12-Dec 12
- Dec 12 - Jan 11
- Jan 11 - Feb 10
1.10 KMS plugin version management

etcd

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

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

\{SECRET3\}_{DEK3} + \{DEK3\}_{KEKv3}

KMS

KEKv1

KEKv2

KEKv3

Nov 12-Dec 12

Dec 12 - Jan 11

Jan 11 - Feb 10
1.10 KMS plugin version management

etcd

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

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

{SECRET3}_{DEK3} + {DEK3}_{KEKv3}

KMS

KEKv1

KEKv2

KEKv3

Nov 12-Dec 12

Dec 12 - Jan 11

Jan 11 - Feb 10
KMS plugin: threat model and concerns

- KMS server is compromised
- KMS plugin is compromised
- Auth token for KMS - offline attack against K8S with plugin
Demo
Kubernetes secrets: external secrets

Diagram showing the connection between Kubernetes and external secret managers.
Kubernetes secrets: HashiCorp Vault

HashiCorp Vault KMS plugin for Kubernetes
- Secrets are in etcd, with root of trust in Vault

Kubernetes auth backend for HashiCorp Vault
- Authenticate to Vault using a K8s service account

Watch: https://www.youtube.com/watch?v=B16YTeSs1hI
### Kubernetes secrets: requirements

<table>
<thead>
<tr>
<th></th>
<th>Identity</th>
<th>Auditing</th>
<th>Encryption</th>
<th>Rotation</th>
<th>Isolation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kubernetes default</strong></td>
<td>□ Node authorizer</td>
<td>□ K8s audit logging</td>
<td>aescbc, aesgcm, or secretbox</td>
<td></td>
<td>□ In etcd, not in applications</td>
</tr>
<tr>
<td><strong>1.7 EncryptionConfig</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>1.10 KMS plugin</strong></td>
<td></td>
<td></td>
<td></td>
<td>KEK only, depending on KMS</td>
<td></td>
</tr>
<tr>
<td><strong>External secrets provider</strong></td>
<td></td>
<td>Additional KMS logs</td>
<td></td>
<td>Depending on secret manager</td>
<td></td>
</tr>
<tr>
<td><strong>Google Cloud</strong></td>
<td>□ May be more tightly scoped</td>
<td>□ Additional secret manager logs</td>
<td>□ Depending on secret manager</td>
<td>□ Depending on secret manager</td>
<td>□ In external secret store</td>
</tr>
</tbody>
</table>
Kubernetes secrets: summary

- Use encryption based on your threat model, e.g., two layers, like full-disk + application-layer
- Rotate keys regularly to limit the impact of a potential key compromise
- Use envelope encryption to separate key management from secret management, and maintain a root of trust
- In Kubernetes, protect secrets using either the KMS plugin or if you already have one, use an external secret store
Learn more

**Kubernetes secrets:** [https://kubernetes.io/docs/concepts/configuration/secret/](https://kubernetes.io/docs/concepts/configuration/secret/)

- Secret encryption: [https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/](https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/)
- Using a KMS provider: [https://kubernetes.io/docs/tasks/administer-cluster/kms-provider/](https://kubernetes.io/docs/tasks/administer-cluster/kms-provider/)

**KMS plugins:**

- Microsoft Azure Key Vault: [https://github.com/Azure/kubernetes-kms](https://github.com/Azure/kubernetes-kms)
- AWS KMS: [https://github.com/kubernetes-sigs/aws-encryption-provider](https://github.com/kubernetes-sigs/aws-encryption-provider)
- HashiCorp Vault: [https://github.com/oracle/kubernetes-vault-kms-plugin](https://github.com/oracle/kubernetes-vault-kms-plugin)

**Container security overview:** [https://cloud.google.com/containers/security/](https://cloud.google.com/containers/security/)
Q&A