Table of Contents

Acceuil

Introduction

Ceci est l'acceuil pour le WriteUp éffectué dans le cadre du cours Sécurité Cloud. Le WriteUp est présent dans le menu déroulant sur la gauche. Celui-ci évoquera la sécuration de Kubernetes dans un environnement cloud AWS grâce à l'usage d'OPA Gatekeeper et du language Rego. Le WriteUp est pensé comme une forme de tutoriel explicatif des grands concepts et de la mise en production.

Sécuriser Kubernetes avec OPA Gatekeeper

Contexte

Votre entreprise a été engagée par SuperTech, qui a récemment rencontré des problèmes de sécurité avec ses clusters Kubernetes. L'organisation a eu des difficultés avec des déploiements mal configurés et des paramètres de ressources non conformes, ce qui a entraîné plusieurs incidents de sécurité. Pour résoudre ces problèmes, vous êtes chargé d'aider à mettre en œuvre une solution robuste d'application des politiques à l'aide d'OPA Gatekeeper afin de garantir les points suivants :

Système de fichiers en lecture seule

S'assurer que tous les pods utilisent un système de fichiers en lecture seule pour empêcher les modifications non autorisées.

Escalade de privilèges

Bloquer les pods où l'option AllowPrivilegeEscalation dans le contexte de sécurité est définie sur true, ce qui pourrait permettre aux processus d'acquérir des privilèges supplémentaires.

Utilisation du réseau de l'hôte

Interdire l'utilisation de hostNetwork: true dans les spécifications des pods pour éviter d'exposer directement le pod au réseau de l'hôte, réduisant ainsi le risque d'attaques au niveau réseau.

Restriction des volumes HostPath

Restreindre l'utilisation des volumes hostPath sur un pod Kubernetes.

Technologie

Kubernetes (ou K8s)

Kubernetes (souvent abrégé en K8s) est une plateforme open source d'orchestration de conteneurs. Elle automatise le déploiement, la mise à l'échelle, la gestion et l'exploitation des applications conteneurisées. Tout ceci permet donc de gérer des clusters de machines physiques ou virtuelles, assure une haute disponibilité des applications et simplifie leur maintenance grâce à des concepts tels que les pods, les services et les déploiements.

OPA

OPA (Open Policy Agent) est un moteur de règles généraliste qui utilise le langage Rego pour définir et appliquer des politiques dans divers systèmes tels que les microservices ou les pipelines CI/CD. OPA Gatekeeper, quant à lui, est une extension spécifique à Kubernetes d'OPA, fournissant une application des politiques et un contrôle pour les clusters Kubernetes. Bien qu'OPA puisse être intégré dans divers environnements, Gatekeeper est spécifiquement conçu pour gérer les ressources Kubernetes en utilisant des modèles de contraintes et des contraintes pour appliquer des politiques directement dans l'écosystème K8s.

OPA Gatekeeper

OPA Gatekeeper fonctionne comme un contrôleur d'admission dans Kubernetes :

Définition des politiques

Les politiques sont écrites en Rego et spécifient les règles auxquelles les ressources doivent se conformer.

Modèles de contraintes

Modèles qui définissent le schéma des contraintes. Les contraintes précisent quelles politiques appliquer et à quelles ressources.

Contraintes

Instances des modèles de contraintes qui appliquent des politiques spécifiques.

Contrôleur d'admission

Intercepte les requêtes envoyées au serveur API Kubernetes et les évalue par rapport aux contraintes définies.

Audit

Vérifie périodiquement les ressources existantes pour s'assurer qu'elles sont conformes aux contraintes définies.

Introduction sur Rego

Rego est le langage de requête utilisé par Open Policy Agent (OPA) pour définir et appliquer des politiques. Il est conçu pour être déclaratif, ce qui signifie qu'il faut décrire ce qui doit être fait plutôt que comment le faire. Rego permet d'écrire des règles logiques qui s'exécutent sur des données JSON.

Exemple Rego

```
package example.policy

# La règle 'deny' renvoie un message si une condition est violée
deny[msg] {
    input.kind == "Pod"
    input.spec.containers[_].securityContext.readOnlyRootFilesystem
== false
    msg = "Les pods doivent utiliser un système de fichiers en
lecture seule."
}
```

package example.policy

Définit le package dans lequel cette règle est incluse. Cela permet d'organiser les politiques.

deny[msg]

La règle deny s'exécute si toutes les conditions définies dans le bloc sont vraies. Si la règle est déclenchée, elle renvoie un message d'erreur.

input.kind == "Pod"

Vérifie que l'objet envoyé en entrée est un pod Kubernetes.

input.spec.containers[_].securityContext.readOnlyRootFilesystem == false

Parcourt tous les conteneurs spécifiés dans le pod (containers[_]) et vérifie si l'attribut readOnlyRootFilesystem dans le contexte de sécurité est défini sur false.

Si les conditions sont satisfaites (par exemple, un conteneur ne respecte pas la configuration), un message est renvoyé pour expliquer le problème : "Les pods doivent utiliser un système de fichiers en lecture seule."

Introduction sur K8s (en yaml)

Les fichiers YAML sont utilisés pour définir l'état souhaité d'objets dans un cluster. Ces fichiers permettent de spécifier les configurations nécessaires pour créer et gérer divers objets Kubernetes tels que pods, services, déploiements, etc. Dans le contexte d'OPA Gatekeeper, un fichier YAML est utilisé pour définir des modèles de contraintes (ConstraintTemplate) et des contraintes (Constraint).

Exemple K8s

```yaml
apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
metadata:
  name: UNIQUE_CONSTRAINT_Template_NAME
spec:
  crd:
    spec:
      names:
        kind: sample
  targets:
```

```
    - target: admission.k8s.gatekeeper.sh
      rego: |
```

apiVersion: templates.gatekeeper.sh/v1beta1

Spécifie la version de l'API utilisée pour définir l'objet. Ici, l'API templates.gatekeeper.sh/v1beta1 est spécifique aux modèles de contraintes (ConstraintTemplate) dans OPA Gatekeeper.

kind: ConstraintTemplate

Définit le type d'objet Kubernetes. ConstraintTemplate est un modèle qui sert de base pour créer des contraintes personnalisées dans Gatekeeper.

metadata

Contient des informations standard sur l'objet, telles que :

- name : Nom unique de la ressource. Ce nom est utilisé pour référencer le modèle dans d'autres ressources, comme les contraintes.

spec

Définit les spécifications de la ressource. Pour un ConstraintTemplate, cela inclut deux parties principales :

crd

Permet à Kubernetes de reconnaître et gérer les contraintes définies par le modèle.

- spec.names.kind: sample: Définit le type de contrainte que ce modèle prendra en charge. Par exemple, le type personnalisé ici est appelé sample.

targets

Spécifie la cible pour l'application de la politique.

- target: admission.k8s.gatekeeper.sh: Indique que cette politique s'applique aux requêtes interceptées par le contrôleur d'admission de Kubernetes.
- rego: Contient le code Rego qui définit la logique de la contrainte. Le contenu de ce bloc sera la règle écrite en langage Rego pour évaluer les ressources Kubernetes.

Application d'une règle Rego sur un YAML K8s

Dans l'exemple proposé, la contrainte Rego vérifie que les pods n'utilisent pas le champ `hostNetwork: true`, une configuration capable d'exposer des pods directement au réseau de la machine hôte.

ContraintTemplate

Création du modèle de contrainte.

```yaml
apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
metadata:
  name: k8shostnetworkdisallowed
spec:
  crd:
    spec:
      names:
        kind: K8sHostNetworkDisallowed
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8shostnetworkdisallowed

        # Détecte les violations lorsque hostNetwork est défini sur true
        violation[{"msg": msg}] {
          input.review.object.spec.hostNetwork == true
```

```
          msg := "The use of hostNetwork: true is not allowed for
security reasons."
        }
```

metadata

name: k8shostnetworkdisallowed

- Nom unique pour le modèle de contrainte. Il identifie clairement la politique appliquée (ici, interdire hostNetwork: true).
- Définit la version de l'API utilisée pour créer cette contrainte.

spec.crd.spec.names.kind

kind: K8sHostNetworkDisallowed

- Nom du type de contrainte qui sera défini. Les utilisateurs de Gatekeeper utiliseront ce nom pour créer des contraintes spécifiques basées sur ce modèle.

targets

target: admission.k8s.gatekeeper.sh

- Spécifie que cette politique s'applique au contrôleur d'admission Kubernetes interceptant les requêtes API.

rego

La logique

- violation[{"msg": msg}]
  - Définit la règle qui détecte les violations. Si une violation est trouvée, un message

explicatif est renvoyé.

- Condition : input.review.object.spec.hostNetwork == true
  - Vérifie si le champ hostNetwork de la spécification du pod est défini sur true.
- Message : msg := "The use of hostNetwork: true is not allowed for security reasons."
  - Si une violation est détectée, ce message est retourné pour expliquer pourquoi la requête est bloquée.

Constraint

La Constraint applique le modèle défini dans le ContraintTemplate à des ressources spécifiques.

```yaml
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sHostNetworkDisallowed
metadata:
  name: restrict-hostnetwork
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    # Facultatif : Filtre par namespace
    namespaces: ["default", "production"]
```

apiversion

apiVersion: constraints.gatekeeper.sh/v1beta1

- Définit la version de l'API utilisée pour créer cette contrainte.
- Le préfixe constraints.gatekeeper.sh est spécifique à Gatekeeper et permet d'appliquer des règles de politique définies à l'aide d'un ConstraintTemplate.

kind

kind: K8sHostNetworkDisallowed

- Indique le type de contrainte.
- Ce type doit correspondre à celui défini dans le ConstraintTemplate.
- Ce lien garantit que cette contrainte applique les règles définies dans le modèle associé.

metadata

name: restrict-hostnetwork

- Nom unique pour identifier cette contrainte dans le cluster. Cela permet d'appliquer plusieurs contraintes basées sur le même modèle avec des noms différents.

spec

spec.match.kinds

- apiGroups: [""]
  - Spécifie l'API groupée par laquelle les ressources sont exposées. Une valeur vide ("") signifie que l'API correspond au groupe de base (ressources standards comme Pod, Service, etc.).
- kinds: ["Pod"]
  - Indique que cette contrainte s'applique aux objets de type Pod.

Tester la contrainte

Création d'un Pod avec `hostNetwork: true` pour vérifier que la contrainte est appliquée.

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name: non-compliant-pod
spec:
  hostNetwork: true
  containers:
    - name: nginx
      image: nginx
```

Résultat

```
Error from server (Forbidden): admission webhook
"validation.gatekeeper.sh" denied the request:
The use of hostNetwork: true is not allowed for security reasons.
```

Application sur l'instance

Pour respecter la consigne de mise en oeuvre nous procéderons comme ceci en utilisant la documentation de K8s et ses capabilities. Sur la machine hôte du cluster K8s, il faudra créer les différents fichiers `.yaml` et les positionner dans un dossier quelconque. On pourra imaginer ceci :

```
---home
   |
   ---debian
      |
      ---k8s_yaml
         |
         ---Constraintz
            |
            ---spk8s
               |
               --CT_spk8s.yaml
               --CA_spk8s.yaml
```

Présence de K8s et de Gatekeeper

Check

Commande: `minikube` `start`

Commande: `kubectl` `get` `pods` `-n` `gatekeeper-system`

Installation

Commande: `curl` `-LO` `https://github.com/kubernetes/minikube/releases/latest/download/minikube-linux-amd64`

Commande: `sudo` `install` `minikube-linux-amd64` `/usr/local/bin/minikube` `&&` `rm` `minikube-linux-amd64`

Commande: `kubectl` `apply` `-f` `https://raw.githubusercontent.com/open-policy-agent/gatekeeper/v3.18.2/deploy/gatekeeper.yaml`

Pour s'assurer du bon déroulement de l'explication, il faut d'abord vérifier la présence d'un K8s (dans ce cas là, mis en place par `minikube`) et dans un autre, la présence de `Gatekeeper`. Vous pouvez installer K8s avec `minikube` (https://minikube.sigs.k8s.io/docs/start/?arch=/linux/x86-64/stable/binary+download) en obtenant les binaires d'installation sur le site officiel. Après l'installation de `minikube`, et l'avoir démarré, il faut maintenant installer `Gatekeeper` (https://open-policy-agent.github.io/gatekeeper/website/docs/install/) en appliquant la configuration sur `K8s`. Il faut ensuite vérifier son installation et obtenir la même sortie que celle-ci en executant : `kubectl get pods -n gatekeeper-system`.

```
NAME                                             READY   STATUS
RESTARTS       AGE
gatekeeper-audit-999899c4-4w2gg                  1/1     Running   2
(8m4s ago)   16h
gatekeeper-controller-manager-68488c48c5-66nb5   1/1     Running   1
(8m4s ago)   16h
gatekeeper-controller-manager-68488c48c5-kl9n5   1/1     Running   1
(8m4s ago)   16h
gatekeeper-controller-manager-68488c48c5-m9lfg   1/1     Running   1
(8m4s ago)   16h
```

ConstraintTemplate Application

Commande: `kubectl` `apply` `-f` `CT_spk8s.yaml`

Définition du fichier `CT_spk8s.yaml`.

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: spk8s
spec:
  crd:
    spec:
      names:
        kind: SPK8s
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package spk8s

        violation[{"msg": msg}] {

input.review.object.spec.containers[_].securityContext.readOnlyRootF
ilesystem != true
          msg := "Le systeme de fichier n'est pas en lecture
seulement."
        }

        violation[{"msg": msg}] {

input.review.object.spec.containers[_].securityContext.allowPrivileg
eEscalation == true
          msg := "Attention, l'escalation de privilege est actif."
        }

        violation[{"msg": msg}] {
          input.review.object.spec.hostNetwork == true
          msg := "hostNetwork est actuellement actif."
        }

        violation[{"msg": msg}] {
          input.review.object.spec.volumes[_].hostPath
```

```
        msg := "Les volumes HostPath ne sont pas autorises."
    }
```

Violation	Type	Rego
readOnlyRootFilesystem != true	Système de fichiers en lecture seule	violation[{"msg": msg}] { input.review.object.spec.containers[_].securityContext.readOnlyRootFilesystem != true msg := "Le systeme de fichier n'est pas en lecture seulement." }
allowPrivilegeEscalation == true	Escalade de privilèges	violation[{"msg": msg}] { input.review.object.spec.containers[_].securityContext.allowPrivilegeEscalation == true msg := "Attention, l'escalation de privilege est actif." }
hostNetwork == true	Utilisation du réseau de l'hôte	violation[{"msg": msg}] { input.review.object.spec.hostNetwork == true msg := "hostNetwork est actuellement actif." }
volumes[_].hostPath	Restriction des volumes HostPath	violation[{"msg": msg}] { input.review.object.spec.volumes[_].hostPath msg := "Les volumes HostPath ne sont pas autorises." }

Constraint Application

Commande: `kubectl` + `apply` + `-f` + `CA_spk8s.yaml`

Définition du fichier `CA_spk8s.yaml`.

```yaml
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: SPK8s
metadata:
  name: k8s-security-policies
spec:
  match:
    kinds:
```

```
    - apiGroups: [""]
      kinds: ["Pod"]
```

> Il est possible de filtrer une règle Constraint par namespace en ajoutant dans la section **"match"** la ligne **"namespaces: ["default", "production"]"** présente dans l'exemple ("Constraint" in "Sécuriser Kubernetes avec OPA Gatekeeper").

Debug

error: resource mapping not found

Commande: `kubectl` `describe` `ConstraintTemplate` `-o` `wide`

Lors de l'installation d'une `ConstraintTemplate`, celle-ci peut présenter des erreurs. K8s, dans son application, n'empêchera pas l'éxecution de la création, pour autant, lors de l'application de la `Constraint`, celle-ci déclarera que le template n'existe pas.

```
error: resource mapping not found for name: "disallow-host-network-
rule" namespace: "" from "ee.yaml": no matches for kind
"HostNetworkUsagea" in version "constraints.gatekeeper.sh/v1beta1"
ensure CRDs are installed first
```

Il est alors possible d'utiliser la commande `kubectl describe ConstraintTemplate -o wide` pour obtenir des détails lors d'erreurs dans la création des Constraints et corriger celle-ci.