ATTACKING POLICY

OPEN POLICY AGENT FOR DEVOPS ENVIRONMENTS

WWW.DEVSECOPSGUIDES.COM

Attacking Policy

• Jul 15, 2024 • 22 min read

Table of contents

Show less

Open Policy Agent (OPA) is a versatile tool used to enforce policies and ensure compliance within a DevSecOps environment. However, security misconfigurations in OPA can lead to significant vulnerabilities. One common issue is overly permissive policies, where misconfigured rules allow more access than intended. This can result in unauthorized access to sensitive data or critical services. Additionally, inadequate logging and monitoring of policy decisions can obscure the detection of policy violations and security incidents, making it difficult for security teams to respond promptly. Ensuring that policies are correctly scoped, thoroughly tested, and continuously monitored is crucial to maintaining a secure DevSecOps environment.

Another critical aspect of OPA security misconfiguration is the improper integration with other DevSecOps tools and services. For instance, if OPA is not correctly

integrated with Kubernetes or CI/CD pipelines, it can lead to gaps in policy enforcement, allowing insecure code or configurations to be deployed. Moreover, the lack of proper version control and review processes for policy changes can introduce vulnerabilities, as unreviewed changes might weaken security postures. Therefore, it is essential to establish a robust governance framework for managing policy changes, coupled with automated testing and validation to ensure that security policies remain effective and aligned with organizational security standards. Regular audits and updates to policies are necessary to adapt to evolving threats and ensure comprehensive security coverage.

Allowed Repositories

To enforce policies that restrict container images to specific repositories using Open Policy Agent (OPA) and Gatekeeper, follow these steps. The following example will demonstrate how to create a ConstraintTemplate and Constraint to ensure container images begin with a specified list of prefixes.

1. Create the ConstraintTemplate

First, define the ConstraintTemplate. This template describes the policy logic and the parameters it requires.

Create a file named `k8sallowedrepos_template.yaml` with the following content:

COPY

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sallowedrepos
  annotations:
    metadata.gatekeeper.sh/title: "Allowed Repositories"
    metadata.gatekeeper.sh/version: 1.0.1
    description: >-
      Requires container images to begin with a string from the
specified list.
```

```
spec:
  crd:
    spec:
      names:
        kind: K8sAllowedRepos
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          properties:
            repos:
              description: The list of prefixes a container image is
allowed to have.
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8sallowedrepos

        violation[{"msg": msg}] {
          container := input.review.object.spec.containers[_]
          not startswith_any(container.image, input.parameters.repos)
          msg := sprintf("container <%v> has an invalid image repo
<%v>, allowed repos are %v", [container.name, container.image,
input.parameters.repos])
        }

        violation[{"msg": msg}] {
          container := input.review.object.spec.initContainers[_]
          not startswith_any(container.image, input.parameters.repos)
          msg := sprintf("initContainer <%v> has an invalid image repo
<%v>, allowed repos are %v", [container.name, container.image,
input.parameters.repos])
        }
```

```
        violation[{"msg": msg}] {
          container := input.review.object.spec.ephemeralContainers[_]
          not startswith_any(container.image, input.parameters.repos)
          msg := sprintf("ephemeralContainer <%v> has an invalid image
repo <%v>, allowed repos are %v", [container.name, container.image,
input.parameters.repos])
        }

        startswith_any(image, repos) {
          some repo
          startswith(image, repo)
        }
```

Apply the template to your Kubernetes cluster:

```
kubectl apply -f k8sallowedrepos_template.yaml
```

COPY

2. Create the Constraint

Next, define a Constraint that uses the ConstraintTemplate to specify the allowed repository prefixes.

Create a file named `k8sallowedrepos_constraint.yaml` with the following content:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: allowed-repos
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
```

COPY

```
  parameters:
    repos:
      - "myregistry.io/myrepo"
      - "anotherregistry.io/anotherrepo"
```

Apply the constraint to your Kubernetes cluster:

```
kubectl apply -f k8sallowedrepos_constraint.yaml
```

3. Usage

To test this setup, try deploying a Pod with an image that does not match the allowed repository prefixes:

```
apiVersion: v1
kind: Pod
metadata:
  name: invalid-image-pod
spec:
  containers:
    - name: myapp
      image: invalidregistry.io/myapp:latest
```

Save this as `invalid-image-pod.yaml` and attempt to apply it:

```
kubectl apply -f invalid-image-pod.yaml
```

Automount Service Account Token for Pod

To control the ability of any Pod to enable `automountServiceAccountToken` using Open Policy Agent (OPA) and Gatekeeper, you can create a ConstraintTemplate and a Constraint. This setup will enforce policies that prevent Pods from automatically mounting service account tokens unless explicitly allowed.

1. Create the ConstraintTemplate

First, define the ConstraintTemplate. This template describes the policy logic and the parameters it requires.

Create a file named `k8spspautomountserviceaccounttokenpod_template.yaml` with the following content:

COPY

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spspautomountserviceaccounttokenpod
  annotations:
    metadata.gatekeeper.sh/title: "Automount Service Account Token for
Pod"
    metadata.gatekeeper.sh/version: 1.0.1
    description: >-
      Controls the ability of any Pod to enable
automountServiceAccountToken.
spec:
  crd:
    spec:
      names:
        kind: K8sPSPAutomountServiceAccountTokenPod
      validation:
        openAPIV3Schema:
          type: object
          description: >-
            Controls the ability of any Pod to enable
automountServiceAccountToken.
  targets:
```

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

Apply the template to your Kubernetes cluster:

```
kubectl apply -f k8spspautomountserviceaccounttokenpod_template.yaml
```

COPY

2. Create the Constraint

Next, define a Constraint that uses the ConstraintTemplate to specify the enforcement policy.

Create a file named `k8spspautomountserviceaccounttokenpod_constraint.yaml` with the following content:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPAutomountServiceAccountTokenPod
metadata:
  name: disallow-automount-sa-token
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
```

COPY

Apply the constraint to your Kubernetes cluster:

```
kubectl apply -
```

COPY

```
f k8spspautomountserviceaccounttokenpod_constraint.yaml=
```

3. Usage

To test this setup, try deploying a Pod with `automountServiceAccountToken` enabled:

```
apiVersion: v1
kind: Pod
metadata:
  name: automount-sa-token-pod
spec:
  automountServiceAccountToken: true
  containers:
    - name: myapp
      image: nginx:latest
```

Save this as `automount-sa-token-pod.yaml` and attempt to apply it:

```
kubectl apply -f automount-sa-token-pod.yaml
```

Block Endpoint Edit Default Role

To prevent the `system:aggregate-to-edit` ClusterRole from granting permissions to create, patch, or update Endpoints in Kubernetes, you can create a ConstraintTemplate and a Constraint using Open Policy Agent (OPA) and Gatekeeper. This setup addresses the security concern outlined in CVE-2021-25740.

1. Create the ConstraintTemplate

First, define the ConstraintTemplate. This template describes the policy logic and the parameters it requires.

Create a file named `k8sblockendpointeditdefaultrole_template.yaml` with the following content:

COPY

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sblockendpointeditdefaultrole
  annotations:
    metadata.gatekeeper.sh/title: "Block Endpoint Edit Default Role"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Many Kubernetes installations by default have a
system:aggregate-to-edit
      ClusterRole which does not properly restrict access to editing
Endpoints.
      This ConstraintTemplate forbids the system:aggregate-to-edit
ClusterRole
      from granting permission to create/patch/update Endpoints.

      ClusterRole/system:aggregate-to-edit should not allow
      Endpoint edit permissions due to CVE-2021-25740, Endpoint &
EndpointSlice
      permissions allow cross-Namespace forwarding,
      https://github.com/kubernetes/kubernetes/issues/103675
spec:
  crd:
    spec:
      names:
        kind: K8sBlockEndpointEditDefaultRole
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8sblockendpointeditdefaultrole

        violation[{"msg": msg}] {
            input.review.object.metadata.name == "system:aggregate-to-
```

```
edit"
            endpointRule(input.review.object.rules[_])
            msg := "ClusterRole system:aggregate-to-edit should not
allow endpoint edit permissions. For k8s version < 1.22, the Cluster
Role should be annotated with
rbac.authorization.kubernetes.io/autoupdate=false to prevent
autoreconciliation back to default permissions for this role."
        }

        endpointRule(rule) {
            "endpoints" == rule.resources[_]
            hasEditVerb(rule.verbs)
        }

        hasEditVerb(verbs) {
            "create" == verbs[_]
        }

        hasEditVerb(verbs) {
            "patch" == verbs[_]
        }

        hasEditVerb(verbs) {
            "update" == verbs[_]
        }
```

Apply the template to your Kubernetes cluster:

```
kubectl apply -f k8sblockendpointeditdefaultrole_template.yaml
```

COPY

2. Create the Constraint

Next, define a Constraint that uses the ConstraintTemplate to specify the enforcement policy.

Create a file named `k8sblockendpointeditdefaultrole_constraint.yaml` with the following content:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sBlockEndpointEditDefaultRole
metadata:
  name: block-endpoint-edit-default-role
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["ClusterRole"]
```

Apply the constraint to your Kubernetes cluster:

```
kubectl apply -f k8sblockendpointeditdefaultrole_constraint.yaml
```

3. Usage

To test this setup, try applying a ClusterRole named `system:aggregate-to-edit` with permissions to edit Endpoints:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: system:aggregate-to-edit
rules:
  - apiGroups: [""]
    resources: ["endpoints"]
    verbs: ["create", "patch", "update"]
```

Save this as `test-clusterrole.yaml` and attempt to apply it:

```
kubectl apply -f test-clusterrole.yaml
```

Block Services with type LoadBalancer

To disallow all Services with type `LoadBalancer` in your Kubernetes cluster using Open Policy Agent (OPA) and Gatekeeper, you can create a ConstraintTemplate and a Constraint. This setup will enforce policies that prevent the creation of LoadBalancer-type Services, adhering to the best practices for network security.

1. Create the ConstraintTemplate

First, define the ConstraintTemplate. This template describes the policy logic.

Create a file named `k8sblockloadbalancer_template.yaml` with the following content:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sblockloadbalancer
  annotations:
    metadata.gatekeeper.sh/title: "Block Services with type
LoadBalancer"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Disallows all Services with type LoadBalancer.

      https://kubernetes.io/docs/concepts/services-
networking/service/#loadbalancer
spec:
  crd:
    spec:
```

```
      names:
        kind: K8sBlockLoadBalancer
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8sblockloadbalancer

        violation[{"msg": msg}] {
          input.review.kind.kind == "Service"
          input.review.object.spec.type == "LoadBalancer"
          msg := "User is not allowed to create service of type
LoadBalancer"
        }
```

Apply the template to your Kubernetes cluster:

COPY

```
kubectl apply -f k8sblockloadbalancer_template.yaml
```

2. Create the Constraint

Next, define a Constraint that uses the ConstraintTemplate to specify the enforcement policy.

Create a file named `k8sblockloadbalancer_constraint.yaml` with the following content:

COPY

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sBlockLoadBalancer
metadata:
  name: block-loadbalancer-services
spec:
  match:
    kinds:
```

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

Apply the constraint to your Kubernetes cluster:

```
kubectl apply -f k8sblockloadbalancer_constraint.yaml
```

3. Usage

To test this setup, try creating a Service with type `LoadBalancer`:

```
apiVersion: v1
kind: Service
metadata:
  name: test-loadbalancer-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

Save this as `test-loadbalancer-service.yaml` and attempt to apply it:

```
kubectl apply -f test-loadbalancer-service.yaml
```

The OPA Gatekeeper should deny the creation of the Service and return a violation message specifying that creating a Service of type LoadBalancer is not allowed.

Block NodePort

To block the creation of Services with type `NodePort` in your Kubernetes cluster using Open Policy Agent (OPA) and Gatekeeper, you can use a ConstraintTemplate. This setup will enforce a policy that disallows the creation of `NodePort` type Services.

ConstraintTemplate

Define the ConstraintTemplate. This template describes the policy logic.

Create a file named `k8sblocknodeport_template.yaml` with the following content:

COPY

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sblocknodeport
  annotations:
    metadata.gatekeeper.sh/title: "Block NodePort"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Disallows all Services with type NodePort.

      https://kubernetes.io/docs/concepts/services-
networking/service/#nodeport
spec:
  crd:
    spec:
      names:
        kind: K8sBlockNodePort
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
```

```
        package k8sblocknodeport

        violation[{"msg": msg}] {
          input.review.kind.kind == "Service"
          input.review.object.spec.type == "NodePort"
          msg := "User is not allowed to create service of type
NodePort"
        }
```

Apply the template to your Kubernetes cluster:

```
kubectl apply -f k8sblocknodeport_template.yaml
```

COPY

Usage

You can apply this ConstraintTemplate directly from the Gatekeeper library:

```
kubectl apply -f https://raw.githubusercontent.com/open-
policy-agent/gatekeeper-library/master/library/general/block-
nodeport-services/template.yaml
```

COPY

Block Wildcard Ingress

To block the creation of Ingresses with a blank or wildcard (`*`) hostname in your Kubernetes cluster using Open Policy Agent (OPA) and Gatekeeper, you can use a ConstraintTemplate. This setup ensures that users cannot create Ingresses that can intercept traffic for other services in the cluster.

ConstraintTemplate

Define the ConstraintTemplate. This template describes the policy logic.

Create a file named `k8sblockwildcardingress_template.yaml` with the following content:

COPY

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sblockwildcardingress
  annotations:
    metadata.gatekeeper.sh/title: "Block Wildcard Ingress"
    metadata.gatekeeper.sh/version: 1.0.1
    description: >-
      Users should not be able to create Ingresses with a blank or
wildcard (*) hostname since that would enable them to intercept
traffic for other services in the cluster, even if they don't have
access to those services.
spec:
  crd:
    spec:
      names:
        kind: K8sBlockWildcardIngress
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package K8sBlockWildcardIngress

        contains_wildcard(hostname) = true {
          hostname == ""
        }

        contains_wildcard(hostname) = true {
          contains(hostname, "*")
        }

        violation[{"msg": msg}] {
          input.review.kind.kind == "Ingress"
          # object.get is required to detect omitted host fields
          hostname := object.get(input.review.object.spec.rules[_],
```

```
"host", "")
          contains_wildcard(hostname)
          msg := sprintf("Hostname '%v' is not allowed since it counts
as a wildcard, which can be used to intercept traffic from other
applications.", [hostname])
        }
```

Apply the template to your Kubernetes cluster:

`kubectl apply -f k8sblockwildcardingress_template.yaml`

Usage

You can apply this ConstraintTemplate directly from the Gatekeeper library:

`kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/general/block-wildcard-ingress/template.yaml`

Disallow Interactive TTY Containers

when applying the `Disallow Interactive TTY Containers` ConstraintTemplate using Gatekeeper, you can follow these instructions:

Step-by-Step Instructions

1. **Create ConstraintTemplate YAML File:**
   Save the following YAML content into a file named `k8sdisallowinteractivetty_template.yaml`.

COPY

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sdisallowinteractivetty
  annotations:
```

```
    metadata.gatekeeper.sh/title: "Disallow Interactive TTY
Containers"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Requires that objects have the fields `spec.tty` and
`spec.stdin` set to false or unset.
spec:
  crd:
    spec:
      names:
        kind: K8sDisallowInteractiveTTY
    validation:
      openAPIV3Schema:
        type: object
        properties:
          exemptImages:
            description: >-
              Any container that uses an image that matches an entry
in this list will be excluded
              from enforcement. Prefix-matching can be signified with
`*`. For example: `my-image-*`.

              It is recommended that users use the fully-qualified
Docker image name (e.g. start with a domain name)
              in order to avoid unexpectedly exempting images from an
untrusted repository.
            type: array
            items:
              type: string
  targets:
  - target: admission.k8s.gatekeeper.sh
    rego: |
      package k8sdisallowinteractivetty

      import data.lib.exempt_container.is_exempt

      violation[{"msg": msg, "details": {}}] {
```

```
          c := input_containers[_]
          not is_exempt(c)
          input_allow_interactive_fields(c)
          msg := sprintf("Containers using tty or stdin (%v) are not
allowed running image: %v", [c.name, c.image])
      }

      input_allow_interactive_fields(c) {
          has_field(c, "stdin")
          not c.stdin == false
      }
      input_allow_interactive_fields(c) {
          has_field(c, "tty")
          not c.tty == false
      }
      input_containers[c] {
          c := input.review.object.spec.containers[_]
      }
      input_containers[c] {
          c := input.review.object.spec.ephemeralContainers[_]
      }
      input_containers[c] {
          c := input.review.object.spec.initContainers[_]
      }
      has_field(object, field) = true {
          object[field]
      }
    libs:
      - |
        package lib.exempt_container

        is_exempt(container) {
            exempt_images := object.get(object.get(input,
"parameters", {}), "exemptImages", [])
            img := container.image
            exemption := exempt_images[_]
            _matches_exemption(img, exemption)
```

```
        }

        _matches_exemption(img, exemption) {
            not endswith(exemption, "*")
            exemption == img
        }

        _matches_exemption(img, exemption) {
            endswith(exemption, "*")
            prefix := trim_suffix(exemption, "*")
            startswith(img, prefix)
        }
```

Apply the ConstraintTemplate:

Use `kubectl apply` to apply the ConstraintTemplate directly from its URL:

COPY

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/general/disallowinteractive/template.yaml
```

This command will deploy the ConstraintTemplate `k8sdisallowinteractivetty` into your Kubernetes cluster.

Allow Privilege Escalation in Container

when applying the `Allow Privilege Escalation in Container` ConstraintTemplate using Gatekeeper, follow these steps:

Step-by-Step Instructions

1. **Create ConstraintTemplate YAML File:**

Save the following YAML content into a file named `k8spspallowprivilegeescalationcontainer_template.yaml`.

COPY

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spspallowprivilegeescalationcontainer
  annotations:
    metadata.gatekeeper.sh/title: "Allow Privilege Escalation in
Container"
    metadata.gatekeeper.sh/version: 1.1.0
    description: >-
      Controls restricting escalation to root privileges. Corresponds
to the
      `allowPrivilegeEscalation` field in a PodSecurityPolicy. For
more
      information, see
      https://kubernetes.io/docs/concepts/policy/pod-security-
policy/#privilege-escalation
spec:
  crd:
    spec:
      names:
        kind: K8sPSPAllowPrivilegeEscalationContainer
    validation:
      openAPIV3Schema:
        type: object
        description: >-
          Controls restricting escalation to root privileges.
Corresponds to the
          `allowPrivilegeEscalation` field in a PodSecurityPolicy. For
more
          information, see
          https://kubernetes.io/docs/concepts/policy/pod-security-
policy/#privilege-escalation
        properties:
```

```
          exemptImages:
            description: >-
              Any container that uses an image that matches an entry
in this list will be excluded
              from enforcement. Prefix-matching can be signified with
`*`. For example: `my-image-*`.

              It is recommended that users use the fully-qualified
Docker image name (e.g. start with a domain name)
              in order to avoid unexpectedly exempting images from an
untrusted repository.
            type: array
            items:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      code:
        - engine: K8sNativeValidation
          source:
            variables:
              - name: containers
                expression: 'has(variables.anyObject.spec.containers)
? variables.anyObject.spec.containers : []'
              - name: initContainers
                expression:
'has(variables.anyObject.spec.initContainers) ?
variables.anyObject.spec.initContainers : []'
              - name: ephemeralContainers
                expression:
'has(variables.anyObject.spec.ephemeralContainers) ?
variables.anyObject.spec.ephemeralContainers : []'
              - name: exemptImagePrefixes
                expression: |
                  !has(variables.params.exemptImages) ? [] :
                    variables.params.exemptImages.filter(image,
image.endsWith("*")).map(image, string(image).replace("*", ""))
              - name: exemptImageExplicit
```

```
              expression: |
                !has(variables.params.exemptImages) ? [] :
                  variables.params.exemptImages.filter(image,
!image.endsWith("*"))
            - name: exemptImages
              expression: |
                (variables.containers + variables.initContainers +
variables.ephemeralContainers).filter(container,
                  container.image in variables.exemptImageExplicit
||
                  variables.exemptImagePrefixes.exists(exemption,
string(container.image).startsWith(exemption))
                ).map(container, container.image)
            - name: badContainers
              expression: |
                (variables.containers + variables.initContainers +
variables.ephemeralContainers).filter(container,
                  !(container.image in variables.exemptImages) && (
                    !has(container.securityContext) ||

!has(container.securityContext.allowPrivilegeEscalation) ||

container.securityContext.allowPrivilegeEscalation != false
                  )
                )
          validations:
            - expression: '(has(request.operation) &&
request.operation == "UPDATE") || size(variables.badContainers) == 0'
              messageExpression: '"Privilege escalation container is
not allowed: " + variables.badContainers.map(c, c.image).join(", ")'
      - engine: Rego
        source:
          rego: |
            package k8spspallowprivilegeescalationcontainer

            import data.lib.exclude_update.is_update
            import data.lib.exempt_container.is_exempt
```

```
            violation[{"msg": msg, "details": {}}] {
                #
spec.containers.securityContext.allowPrivilegeEscalation field is
immutable.
                not is_update(input.review)

                c := input_containers[_]
                not is_exempt(c)
                input_allow_privilege_escalation(c)
                msg := sprintf("Privilege escalation container is
not allowed: %v", [c.name])
            }

            input_allow_privilege_escalation(c) {
                not has_field(c, "securityContext")
            }
            input_allow_privilege_escalation(c) {
                not c.securityContext.allowPrivilegeEscalation ==
false
            }
            input_containers[c] {
                c := input.review.object.spec.containers[_]
            }
            input_containers[c] {
                c := input.review.object.spec.initContainers[_]
            }
            input_containers[c] {
                c := input.review.object.spec.ephemeralContainers[_]
            }
            # has_field returns whether an object has a field
            has_field(object, field) = true {
                object[field]
            }
        libs:
          - |
            package lib.exclude_update
```

```
            is_update(review) {
                review.operation == "UPDATE"
            }
          - |
            package lib.exempt_container

            is_exempt(container) {
                exempt_images := object.get(object.get(input,
"parameters", {}), "exemptImages", [])
                img := container.image
                exemption := exempt_images[_]
                _matches_exemption(img, exemption)
            }

            _matches_exemption(img, exemption) {
                not endswith(exemption, "*")
                exemption == img
            }

            _matches_exemption(img, exemption) {
                endswith(exemption, "*")
                prefix := trim_suffix(exemption, "*")
                startswith(img, prefix)
            }
```

Apply the ConstraintTemplate:

Use `kubectl apply` to apply the ConstraintTemplate directly from its URL:

COPY

```
kubectl apply -f https://raw.githubusercontent.com/open-
policy-agent/gatekeeper-library/master/library/pod-
security-policy/allow-privilege-escalation/template.yaml
```

Privileged Container

when applying the `Privileged Container` ConstraintTemplate using Gatekeeper, follow these steps:

1. **Create ConstraintTemplate YAML File:**
   Save the following YAML content into a file named `k8spspprivilegedcontainer_template.yaml`.

COPY

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spspprivilegedcontainer
  annotations:
    metadata.gatekeeper.sh/title: "Privileged Container"
    metadata.gatekeeper.sh/version: 1.1.0
    description: >-
      Controls the ability of any container to enable privileged mode.
      Corresponds to the `privileged` field in a PodSecurityPolicy.
For more
      information, see
      https://kubernetes.io/docs/concepts/policy/pod-security-
policy/#privileged
spec:
  crd:
    spec:
      names:
        kind: K8sPSPPrivilegedContainer
    validation:
      openAPIV3Schema:
        type: object
        description: >-
          Controls the ability of any container to enable privileged
mode.
          Corresponds to the `privileged` field in a
PodSecurityPolicy. For more
```

```
          information, see
          https://kubernetes.io/docs/concepts/policy/pod-security-
policy/#privileged
        properties:
          exemptImages:
            description: >-
              Any container that uses an image that matches an entry
in this list will be excluded
              from enforcement. Prefix-matching can be signified with
`*`. For example: `my-image-*`.

              It is recommended that users use the fully-qualified
Docker image name (e.g. start with a domain name)
              in order to avoid unexpectedly exempting images from an
untrusted repository.
            type: array
            items:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      code:
        - engine: K8sNativeValidation
          source:
            variables:
              - name: containers
                expression: 'has(variables.anyObject.spec.containers)
? variables.anyObject.spec.containers : []'
              - name: initContainers
                expression:
'has(variables.anyObject.spec.initContainers) ?
variables.anyObject.spec.initContainers : []'
              - name: ephemeralContainers
                expression:
'has(variables.anyObject.spec.ephemeralContainers) ?
variables.anyObject.spec.ephemeralContainers : []'
              - name: exemptImagePrefixes
                expression: |
```

```
                !has(variables.params.exemptImages) ? [] :
                  variables.params.exemptImages.filter(image,
image.endsWith("*")).map(image, string(image).replace("*", ""))
              - name: exemptImageExplicit
                expression: |
                  !has(variables.params.exemptImages) ? [] :
                    variables.params.exemptImages.filter(image,
!image.endsWith("*"))
              - name: exemptImages
                expression: |
                  (variables.containers + variables.initContainers +
variables.ephemeralContainers).filter(container,
                    container.image in variables.exemptImageExplicit
||
                    variables.exemptImagePrefixes.exists(exemption,
string(container.image).startsWith(exemption)))
              - name: badContainers
                expression: |
                  (variables.containers + variables.initContainers +
variables.ephemeralContainers).filter(container,
                    !(container.image in variables.exemptImages) &&
                    (has(container.securityContext) &&
has(container.securityContext.privileged) &&
container.securityContext.privileged == true)
                  ).map(container, "Privileged container is not
allowed: " + container.name +", securityContext: " +
container.securityContext)
            validations:
              - expression: '(has(request.operation) &&
request.operation == "UPDATE") || size(variables.badContainers) == 0'
                messageExpression:
'variables.badContainers.join("\n")'
        - engine: Rego
          source:
            rego: |
              package k8spspprivileged
```

```
            import data.lib.exclude_update.is_update
            import data.lib.exempt_container.is_exempt

            violation[{"msg": msg, "details": {}}] {
                # spec.containers.privileged field is immutable.
                not is_update(input.review)

                c := input_containers[_]
                not is_exempt(c)
                c.securityContext.privileged
                msg := sprintf("Privileged container is not allowed:
%v, securityContext: %v", [c.name, c.securityContext])
            }

            input_containers[c] {
                c := input.review.object.spec.containers[_]
            }

            input_containers[c] {
                c := input.review.object.spec.initContainers[_]
            }

            input_containers[c] {
                c := input.review.object.spec.ephemeralContainers[_]
            }
          libs:
            - |
              package lib.exclude_update

              is_update(review) {
                  review.operation == "UPDATE"
              }
            - |
              package lib.exempt_container

              is_exempt(container) {
                  exempt_images := object.get(object.get(input,
```

```
"parameters", {}), "exemptImages", [])
                    img := container.image
                    exemption := exempt_images[_]
                    _matches_exemption(img, exemption)
                }

                _matches_exemption(img, exemption) {
                    not endswith(exemption, "*")
                    exemption == img
                }

                _matches_exemption(img, exemption) {
                    endswith(exemption, "*")
                    prefix := trim_suffix(exemption, "*")
                    startswith(img, prefix)
                }
```

Apply the ConstraintTemplate:

Use `kubectl apply` to apply the ConstraintTemplate directly from its URL:

COPY

```
kubectl apply -f https://raw.githubusercontent.com/open-
policy-agent/gatekeeper-library/master/library/pod-
security-policy/privileged-containers/template.yaml
```

Read Only Root Filesystem

the `Read Only Root Filesystem` ConstraintTemplate using Gatekeeper and skip steps 2 and 3, follow these instructions:

1. **Create ConstraintTemplate YAML File:**
   Save the following YAML content into a file named `k8spspreadonlyrootfilesystem_template.yaml`.

COPY

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spspreadonlyrootfilesystem
  annotations:
    metadata.gatekeeper.sh/title: "Read Only Root Filesystem"
    metadata.gatekeeper.sh/version: 1.1.0
    description: >-
      Requires the use of a read-only root file system by pod
containers.
      Corresponds to the `readOnlyRootFilesystem` field in a
      PodSecurityPolicy. For more information, see
      https://kubernetes.io/docs/concepts/policy/pod-security-
policy/#volumes-and-file-systems
spec:
  crd:
    spec:
      names:
        kind: K8sPSPReadOnlyRootFilesystem
    validation:
      openAPIV3Schema:
        type: object
        description: >-
          Requires the use of a read-only root file system by pod
containers.
          Corresponds to the `readOnlyRootFilesystem` field in a
          PodSecurityPolicy. For more information, see
          https://kubernetes.io/docs/concepts/policy/pod-security-
policy/#volumes-and-file-systems
        properties:
          exemptImages:
            description: >-
              Any container that uses an image that matches an entry
in this list will be excluded
              from enforcement. Prefix-matching can be signified with
`*`. For example: `my-image-*`.
```

```
              It is recommended that users use the fully-qualified
Docker image name (e.g. start with a domain name)
              in order to avoid unexpectedly exempting images from an
untrusted repository.
            type: array
            items:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      code:
        - engine: K8sNativeValidation
          source:
            variables:
              - name: containers
                expression: 'has(variables.anyObject.spec.containers)
? variables.anyObject.spec.containers : []'
              - name: initContainers
                expression:
'has(variables.anyObject.spec.initContainers) ?
variables.anyObject.spec.initContainers : []'
              - name: ephemeralContainers
                expression:
'has(variables.anyObject.spec.ephemeralContainers) ?
variables.anyObject.spec.ephemeralContainers : []'
              - name: exemptImagePrefixes
                expression: |
                  !has(variables.params.exemptImages) ? [] :
                    variables.params.exemptImages.filter(image,
image.endsWith("*")).map(image, string(image).replace("*", ""))
              - name: exemptImageExplicit
                expression: |
                  !has(variables.params.exemptImages) ? [] :
                    variables.params.exemptImages.filter(image,
!image.endsWith("*"))
              - name: exemptImages
                expression: |
```

```
                (variables.containers + variables.initContainers +
variables.ephemeralContainers).filter(container,
                  container.image in variables.exemptImageExplicit
||
                  variables.exemptImagePrefixes.exists(exemption,
string(container.image).startsWith(exemption)))
              - name: badContainers
                expression: |
                  (variables.containers + variables.initContainers +
variables.ephemeralContainers).filter(container,
                  !(container.image in variables.exemptImages) &&
                  (!has(container.securityContext) ||

!has(container.securityContext.readOnlyRootFilesystem) ||
                  container.securityContext.readOnlyRootFilesystem
!= true)
                ).map(container, container.name)
            validations:
              - expression: '(has(request.operation) &&
request.operation == "UPDATE") || size(variables.badContainers) == 0'
                messageExpression: '"only read-only root filesystem
container is allowed: " + variables.badContainers.join(", ")'

        - engine: Rego
          source:
            rego: |
              package k8spspreadonlyrootfilesystem

              import data.lib.exclude_update.is_update
              import data.lib.exempt_container.is_exempt

              violation[{"msg": msg, "details": {}}] {
                  # spec.containers.readOnlyRootFilesystem field is
immutable.
                  not is_update(input.review)

                  c := input_containers[_]
```

```
                not is_exempt(c)
                input_read_only_root_fs(c)
                msg := sprintf("only read-only root filesystem
container is allowed: %v", [c.name])
            }

            input_read_only_root_fs(c) {
                not has_field(c, "securityContext")
            }
            input_read_only_root_fs(c) {
                not c.securityContext.readOnlyRootFilesystem == true
            }

            input_containers[c] {
                c := input.review.object.spec.containers[_]
            }
            input_containers[c] {
                c := input.review.object.spec.initContainers[_]
            }
            input_containers[c] {
                c := input.review.object.spec.ephemeralContainers[_]
            }

            # has_field returns whether an object has a field
            has_field(object, field) = true {
                object[field]
            }
          libs:
            - |
              package lib.exclude_update

              is_update(review) {
                  review.operation == "UPDATE"
              }
            - |
              package lib.exempt_container
```

```
                is_exempt(container) {
                    exempt_images := object.get(object.get(input,
"parameters", {}), "exemptImages", [])
                    img := container.image
                    exemption := exempt_images[_]
                    _matches_exemption(img, exemption)
                }

                _matches_exemption(img, exemption) {
                    not endswith(exemption, "*")
                    exemption == img
                }

                _matches_exemption(img, exemption) {
                    endswith(exemption, "*")
                    prefix := trim_suffix(exemption, "*")
                    startswith(img, prefix)
                }
```

Apply the ConstraintTemplate:

Use `kubectl apply` to apply the ConstraintTemplate directly from its URL:

COPY

```
kubectl apply -f https://raw.githubusercontent.com/open-
policy-agent/gatekeeper-library/master/library/pod-
security-policy/read-only-root-filesystem/template.yaml
```

Host Networking Ports

the `Host Networking Ports` ConstraintTemplate using Gatekeeper and skip steps 2 and 3, follow these instructions:

1. **Create ConstraintTemplate YAML File:**

Save the following YAML content into a file named `k8spsphostnetworkingports_template.yaml`.

COPY

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spsphostnetworkingports
  annotations:
    metadata.gatekeeper.sh/title: "Host Networking Ports"
    metadata.gatekeeper.sh/version: 1.1.1
    description: >-
      Controls usage of host network namespace by pod containers.
HostNetwork verification happens without exception for exemptImages.
Specific
      ports must be specified. Corresponds to the `hostNetwork` and
      `hostPorts` fields in a PodSecurityPolicy. For more information,
see
      https://kubernetes.io/docs/concepts/policy/pod-security-
policy/#host-namespaces
spec:
  crd:
    spec:
      names:
        kind: K8sPSPHostNetworkingPorts
    validation:
      # Schema for the `parameters` field
      openAPIV3Schema:
        type: object
        description: >-
          Controls usage of host network namespace by pod containers.
HostNetwork verification happens without exception for exemptImages.
Specific
          ports must be specified. Corresponds to the `hostNetwork`
and
          `hostPorts` fields in a PodSecurityPolicy. For more
information, see
```

```
          https://kubernetes.io/docs/concepts/policy/pod-security-
policy/#host-namespaces
        properties:
          exemptImages:
            description: >-
              Any container that uses an image that matches an entry
in this list will be excluded
              from enforcement. Prefix-matching can be signified with
`*`. For example: `my-image-*`.

              It is recommended that users use the fully-qualified
Docker image name (e.g. start with a domain name)
              in order to avoid unexpectedly exempting images from an
untrusted repository.
            type: array
            items:
              type: string
          hostNetwork:
            description: "Determines if the policy allows the use of
HostNetwork in the pod spec."
            type: boolean
          min:
            description: "The start of the allowed port range,
inclusive."
            type: integer
          max:
            description: "The end of the allowed port range,
inclusive."
            type: integer
  targets:
    - target: admission.k8s.gatekeeper.sh
      code:
        - engine: K8sNativeValidation
          source:
            variables:
              - name: containers
                expression: 'has(variables.anyObject.spec.containers)
```

```
? variables.anyObject.spec.containers : []'
              - name: initContainers
                expression:
'has(variables.anyObject.spec.initContainers) ?
variables.anyObject.spec.initContainers : []'
              - name: ephemeralContainers
                expression:
'has(variables.anyObject.spec.ephemeralContainers) ?
variables.anyObject.spec.ephemeralContainers : []'
              - name: exemptImagePrefixes
                expression: |
                  !has(variables.params.exemptImages) ? [] :
                    variables.params.exemptImages.filter(image,
image.endsWith("*")).map(image, string(image).replace("*", ""))
              - name: exemptImageExplicit
                expression: |
                  !has(variables.params.exemptImages) ? [] :
                    variables.params.exemptImages.filter(image,
!image.endsWith("*"))
              - name: exemptImages
                expression: |
                  (variables.containers + variables.initContainers +
variables.ephemeralContainers).filter(container,
                    container.image in variables.exemptImageExplicit
||
                    variables.exemptImagePrefixes.exists(exemption,
string(container.image).startsWith(exemption)))
              - name: badContainers
                expression: |
                  (variables.containers + variables.initContainers +
variables.ephemeralContainers).filter(container,
                    !(container.image in variables.exemptImages) &&
                    (
                      (container.ports.all(port, has(port.hostPort) &&
has(variables.params.min) && port.hostPort < variables.params.min)) ||
                      (container.ports.all(port, has(port.hostPort) &&
has(variables.params.max) && port.hostPort > variables.params.max))
```

```
                )
              )
          validations:
            - expression: '(has(request.operation) &&
request.operation == "UPDATE") || size(variables.badContainers) == 0'
              messageExpression: '"The specified hostNetwork and
hostPort are not allowed, pod: " + variables.anyObject.metadata.name +
". Allowed values: " + variables.params'
            - expression: |
                (has(request.operation) && request.operation ==
"UPDATE") ||
                (!has(variables.params.hostNetwork) ||
!variables.params.hostNetwork ?
(has(variables.anyObject.spec.hostNetwork) &&
!variables.anyObject.spec.hostNetwork) : true)
              messageExpression: '"The specified hostNetwork and
hostPort are not allowed, pod: " + variables.anyObject.metadata.name +
". Allowed values: " + variables.params'
        - engine: Rego
          source:
            rego: |
              package k8spsphostnetworkingports

              import data.lib.exclude_update.is_update
              import data.lib.exempt_container.is_exempt

              violation[{"msg": msg, "details": {}}] {
                  # spec.hostNetwork field is immutable.
                  not is_update(input.review)

                  input_share_hostnetwork(input.review.object)
                  msg := sprintf("The specified hostNetwork and
hostPort are not allowed, pod: %v. Allowed values: %v",
[input.review.object.metadata.name, input.parameters])
              }

              input_share_hostnetwork(o) {
```

```
                not input.parameters.hostNetwork
                o.spec.hostNetwork
            }

            input_share_hostnetwork(_) {
                hostPort := input_containers[_].ports[_].hostPort
                hostPort < input.parameters.min
            }

            input_share_hostnetwork(_) {
                hostPort := input_containers[_].ports[_].hostPort
                hostPort > input.parameters.max
            }

            input_containers[c] {
                c := input.review.object.spec.containers[_]
                not is_exempt(c)
            }

            input_containers[c] {
                c := input.review.object.spec.initContainers[_]
                not is_exempt(c)
            }

            input_containers[c] {
                c :=
input.review.object.spec.ephemeralContainers[_]
                not is_exempt(c)
            }
        libs:
          - |
            package lib.exclude_update

            is_update(review) {
                review.operation == "UPDATE"
            }
          - |
```

```
                package lib.exempt_container

                is_exempt(container) {
                    exempt_images := object.get(object.get(input,
"parameters", {}), "exemptImages", [])
                    img := container.image
                    exemption := exempt_images[_]
                    _matches_exemption(img, exemption)
                }

                _matches_exemption(img, exemption) {
                    not endswith(exemption, "*")
                    exemption == img
                }

                _matches_exemption(img, exemption) {
                    e
```

Apply the ConstraintTemplate:

Use `kubectl apply` to apply the ConstraintTemplate directly from its URL:

```
kubectl apply -f https://raw.githubusercontent.com/open-
policy-agent/gatekeeper-library/master/library/pod-
security-policy/host-network-ports/template.yaml
```

This command will deploy the ConstraintTemplate `k8spsphostnetworkingports` into your Kubernetes cluster.

App Armor

the `App Armor` ConstraintTemplate using Gatekeeper, follow these streamlined instructions:

1. **Create ConstraintTemplate YAML File:**
   Save the following YAML content into a file named `k8spspapparmor_template.yaml`.

COPY

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spspapparmor
  annotations:
    metadata.gatekeeper.sh/title: "App Armor"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Configures an allow-list of AppArmor profiles for use by
containers.
      This corresponds to specific annotations applied to a
PodSecurityPolicy.
      For information on AppArmor, see
      https://kubernetes.io/docs/tutorials/clusters/apparmor/
spec:
  crd:
    spec:
      names:
        kind: K8sPSPAppArmor
    validation:
      # Schema for the `parameters` field
      openAPIV3Schema:
        type: object
        description: >-
          Configures an allow-list of AppArmor profiles for use by
containers.
          This corresponds to specific annotations applied to a
PodSecurityPolicy.
          For information on AppArmor, see
          https://kubernetes.io/docs/tutorials/clusters/apparmor/
        properties:
          exemptImages:
            description: >-
```

```
              Any container that uses an image that matches an entry
in this list will be excluded
              from enforcement. Prefix-matching can be signified with
`*`. For example: `my-image-*`.

              It is recommended that users use the fully-qualified
Docker image name (e.g. start with a domain name)
              in order to avoid unexpectedly exempting images from an
untrusted repository.
            type: array
            items:
              type: string
          allowedProfiles:
            description: "An array of AppArmor profiles. Examples:
`runtime/default`, `unconfined`."
            type: array
            items:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8spspapparmor

        import data.lib.exempt_container.is_exempt

        violation[{"msg": msg, "details": {}}] {
            metadata := input.review.object.metadata
            container := input_containers[_]
            not is_exempt(container)
            not input_apparmor_allowed(container, metadata)
            msg := sprintf("AppArmor profile is not allowed, pod: %v,
container: %v. Allowed profiles: %v",
[input.review.object.metadata.name, container.name,
input.parameters.allowedProfiles])
        }

        input_apparmor_allowed(container, metadata) {
```

```
            get_annotation_for(container, metadata) ==
input.parameters.allowedProfiles[_]
        }

        input_containers[c] {
            c := input.review.object.spec.containers[_]
        }
        input_containers[c] {
            c := input.review.object.spec.initContainers[_]
        }
        input_containers[c] {
            c := input.review.object.spec.ephemeralContainers[_]
        }

        get_annotation_for(container, metadata) = out {
            out =
metadata.annotations[sprintf("container.apparmor.security.beta.kuberne
tes.io/%v", [container.name])]
        }
        get_annotation_for(container, metadata) = out {
            not
metadata.annotations[sprintf("container.apparmor.security.beta.kuberne
tes.io/%v", [container.name])]
            out = "runtime/default"
        }
      libs:
        - |
          package lib.exempt_container

          is_exempt(container) {
              exempt_images := object.get(object.get(input,
"parameters", {}), "exemptImages", [])
              img := container.image
              exemption := exempt_images[_]
              _matches_exemption(img, exemption)
          }
```

```
        _matches_exemption(img, exemption) {
            not endswith(exemption, "*")
            exemption == img
        }

        _matches_exemption(img, exemption) {
            endswith(exemption, "*")
            prefix := trim_suffix(exemption, "*")
            startswith(img, prefix)
        }
```

Apply the ConstraintTemplate:

Use `kubectl apply` to apply the ConstraintTemplate directly from its URL

COPY

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/pod-security-policy/apparmor/template.yaml
```

SELinux V2

the `SELinux V2` ConstraintTemplate using Gatekeeper while skipping steps 2 and 3, follow these streamlined instructions:

1. **Create ConstraintTemplate YAML File:**
   Save the following YAML content into a file named `k8spspselinuxv2_template.yaml`.

COPY

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spspselinuxv2
```

```
  annotations:
    metadata.gatekeeper.sh/title: "SELinux V2"
    metadata.gatekeeper.sh/version: 1.0.3
    description: >-
      Defines an allow-list of seLinuxOptions configurations for pod
      containers. Corresponds to a PodSecurityPolicy requiring SELinux
configs.
      For more information, see
      https://kubernetes.io/docs/concepts/policy/pod-security-
policy/#selinux
spec:
  crd:
    spec:
      names:
        kind: K8sPSPSELinuxV2
    validation:
      # Schema for the `parameters` field
      openAPIV3Schema:
        type: object
        description: >-
          Defines an allow-list of seLinuxOptions configurations for
pod
          containers. Corresponds to a PodSecurityPolicy requiring
SELinux configs.
          For more information, see
          https://kubernetes.io/docs/concepts/policy/pod-security-
policy/#selinux
        properties:
          exemptImages:
            description: >-
              Any container that uses an image that matches an entry
in this list will be excluded
              from enforcement. Prefix-matching can be signified with
`*`. For example: `my-image-*`.

              It is recommended that users use the fully-qualified
Docker image name (e.g. start with a domain name)
```

```
              in order to avoid unexpectedly exempting images from an
untrusted repository.
            type: array
            items:
              type: string
          allowedSELinuxOptions:
            type: array
            description: "An allow-list of SELinux options
configurations."
            items:
              type: object
              description: "An allowed configuration of SELinux
options for a pod container."
              properties:
                level:
                  type: string
                  description: "An SELinux level."
                role:
                  type: string
                  description: "An SELinux role."
                type:
                  type: string
                  description: "An SELinux type."
                user:
                  type: string
                  description: "An SELinux user."
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8spspselinux

        import data.lib.exclude_update.is_update
        import data.lib.exempt_container.is_exempt

        # Disallow top level custom SELinux options
        violation[{"msg": msg, "details": {}}] {
            # spec.securityContext.seLinuxOptions field is immutable.
```

```
            not is_update(input.review)

            has_field(input.review.object.spec.securityContext,
"seLinuxOptions")
            not
input_seLinuxOptions_allowed(input.review.object.spec.securityContext.
seLinuxOptions)
            msg := sprintf("SELinux options is not allowed, pod: %v.
Allowed options: %v", [input.review.object.metadata.name,
input.parameters.allowedSELinuxOptions])
        }
        # Disallow container level custom SELinux options
        violation[{"msg": msg, "details": {}}] {
            # spec.containers.securityContext.seLinuxOptions field is
immutable.
            not is_update(input.review)

            c := input_security_context[_]
            not is_exempt(c)
            has_field(c.securityContext, "seLinuxOptions")
            not
input_seLinuxOptions_allowed(c.securityContext.seLinuxOptions)
            msg := sprintf("SELinux options is not allowed, pod: %v,
container: %v. Allowed options: %v",
[input.review.object.metadata.name, c.name,
input.parameters.allowedSELinuxOptions])
        }

        input_seLinuxOptions_allowed(options) {
            params := input.parameters.allowedSELinuxOptions[_]
            field_allowed("level", options, params)
            field_allowed("role", options, params)
            field_allowed("type", options, params)
            field_allowed("user", options, params)
        }

        field_allowed(field, options, params) {
```

```
            params[field] == options[field]
        }
        field_allowed(field, options, _) {
            not has_field(options, field)
        }

        input_security_context[c] {
            c := input.review.object.spec.containers[_]
            has_field(c.securityContext, "seLinuxOptions")
        }
        input_security_context[c] {
            c := input.review.object.spec.initContainers[_]
            has_field(c.securityContext, "seLinuxOptions")
        }
        input_security_context[c] {
            c := input.review.object.spec.ephemeralContainers[_]
            has_field(c.securityContext, "seLinuxOptions")
        }

        # has_field returns whether an object has a field
        has_field(object, field) = true {
            object[field]
        }
      libs:
        - |
          package lib.exclude_update

          is_update(review) {
              review.operation == "UPDATE"
          }
        - |
          package lib.exempt_container

          is_exempt(container) {
              exempt_images := object.get(object.get(input,
"parameters", {}), "exemptImages", [])
              img := container.image
```

```
        exemption := exempt_images[_]
        _matches_exemption(img, exemption)
    }

    _matches_exemption(img, exemption) {
        not endswith(exemption, "*")
        exemption == img
    }

    _matches_exemption(img, exemption) {
        endswith(exemption, "*")
        prefix := trim_suffix(exemption, "*")
        startswith(img, prefix)
    }
```

Apply the ConstraintTemplate:

Use `kubectl apply` to apply the ConstraintTemplate directly from its URL:

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/pod-security-policy/selinux/template.yaml
```

Resources

- https://open-policy-agent.github.io/gatekeeper-library/website/validation/selinux/
- https://devsecopsguides.com
- https://www.openpolicyagent.org/integrations

Policy Devops openpolicyagent opa Docker Kubernetes Cloud AWS DevSecOps

Written by

RR **Reza Rashidi**

Add your bio

Published on

DevSecOpsGuides

Add blog description

MORE ARTICLES

RR **Reza Rashidi**

Attacking IaC

Attacking Infrastructure as Code (IaC) methods involves exploiting vulnerabilities and misconfigurat...

RR **Reza Rashidi**

Attacking Vagrant

Vagrant, a tool for building and managing virtual machine environments, is widely used for developme...

RR **Reza Rashidi**

Attacking Golang

Golang (or Go) is a statically typed, compiled programming language designed at Google. It is known ...