Kyverno (PSP Alternative)

- Overview
- How it works
- OAL OKE Kyverno Cluster Policies
- Implementation details
- How to identify resources that are non-compliant
- How to resolve failed policy report resources
- Recommendations
- Need More Help ?

Overview

As a replacement to Pod Security Policies (PSP) which is deprecated in Kubernetes-1.21, we will be using Kyverno. Kyverno is a policy engine designed for Kubernetes. It can validate, mutate, and generate configurations using admission controls and background scans.

This guide is to help developers on how to identify resources that fail to adhere to defined pod security standards and fix them.

How it works

Kyverno runs as a dynamic admission controller in a Kubernetes cluster. Kyverno receives validating and mutating admission webhook HTTP callbacks from the kube-apiserver and applies matching policies to return results that enforce admission policies or reject requests. Read more about Kyverno here

OAL OKE Kyverno Cluster Policies

Kyverno cluster policies that will enforced on our OKE clusters are based out of Kubernetes Pod Security Standards Restrictive profile.

- **deny-privilege-escalation:** Privilege escalation, such as via set-user-ID or set-group-ID file mode, should not be allowed. This policy ensures the `allowPrivilegeEscalation` fields are either undefined or set to `false`
- **disallow-add-capabilities:** Capabilities permit privileged actions without giving full root access. Adding capabilities beyond the default set must not be allowed. This policy ensures users cannot add any additional capabilities to a Pod
- **disallow-host-namespaces:** Host namespaces (Process ID namespace, Inter-Process Communication namespace, and network namespace) allow access to shared information and can be used to elevate privileges. Pods should not be allowed access to host namespaces. This policy ensures fields which make use of these host namespaces are set to `false`
- **disallow-host-path:** HostPath volumes let Pods use host directories and volumes in containers. Using host resources can be used to access shared data or escalate privileges and should not be allowed. This policy ensures no hostPath volumes are in use
- **disallow-host-ports:** Access to host ports allows potential snooping of network traffic and should not be allowed, or at minimum restricted to a known list. This policy ensures the `hostPort` fields are empty.
- **disallow-privileged-containers:** Privileged mode disables most security mechanisms and must not be allowed. This policy ensures Pods do not call for privileged mode
- **disallow-selinux:** SELinux options can be used to escalate privileges and should not be allowed. This policy ensures that the `seLinuxOptions` field is undefined
- **require-default-proc-mount:** The default /proc masks are set up to reduce attack surface and should be required. This policy ensures nothing but the default procMount can be specified.
- **require-non-root-groups:** Containers should be forbidden from running with a root primary or supplementary GID. This policy ensures the `runAsGroup`, `supplementalGroups`, and `fsGroup` fields are set to a number greater than zero (i.e., non root).
- **require-run-as-non-root:** Containers must be required to run as non-root users. This policy ensures `runAsNonRoot` is set to `true`.
- **restrict-apparmor-profiles:** On supported hosts, the 'runtime/default' AppArmor profile is applied by default. The default policy should prevent overriding or disabling the policy, or restrict overrides to an allowed set of profiles. This policy ensures Pods do not specify any other AppArmor profiles than `runtime/default`.
- **restrict-seccomp:** The seccomp profile must not be explicitly set to Unconfined. This policy, requiring Kubernetes v1.19 or later, ensures that seccomp is not set or set to `RuntimeDefault` or `Localhost`
- **restrict-sysctls:** Sysctls can disable security mechanisms or affect all containers on a host, and should be disallowed except for an allowed "safe" subset. A sysctl is considered safe if it is namespaced in the container or the Pod, and it is isolated from other Pods or processes on the same Node. This policy ensures that only those "safe" subsets can be specified in a Pod
- **restrict-volume-types:** In addition to restricting HostPath volumes, the restricted pod security profile limits usage of non-core volume types to those defined through PersistentVolumes. This policy blocks a number of different non-core volume types such as nfs, iscsi, glusterfs, flexvolume, etc.
- **restrict-image-registries:** Images from unknown, public registries can be of dubious quality and may not be scanned and secured, representing a high degree of risk. Requiring use of known, approved registries helps reduce threat exposure by ensuring image pulls only come from them. This policy validates that container images only originate from the registry `iad.ocir.io/oalprod`

Implementation details

In initial phase below policies will be created with Validation Fail Action to report, this will scan all the existing resources and generates reports and also ensures any new deployments created by developer are not blocked. During this phase developers are requested to update their resource manifest files to meet the policy requirements. In second phase, these policies will be updated to enforce, after which any new deployments that fail to be complaint with the defined policies will be rejected.

Policy	DEV	UAT	PROD
deny-privilege-escalation	Enforce	Enforce	Enforce
disallow-add-capabilities	Report	Enforce	Enforce
disallow-host-namespaces	Enforce	Enforce	Enforce
disallow-host-path	Report	Enforce	Enforce
disallow-host-ports	Report	Enforce	Enforce
disallow-privileged-containers	Report	Enforce	Enforce
disallow-selinux	Enforce	Enforce	Enforce
require-default-proc-mount	Enforce	Enforce	Enforce
require-non-root-group	Enforce	Enforce	Enforce
require-run-as-non-root	Enforce	Enforce	Enforce
restrict-apparmor-profiles	Enforce	Enforce	Enforce
restrict-seccomp	Enforce	Enforce	Enforce
restrict-sysctls	Enforce	Enforce	Enforce
restrict-volume-types	Enforce	Enforce	Enforce
restrict-image-registries	Report	Enforce	Enforce

How to identify resources that are non-compliant

After the policies are created, a policy report object will be created in each namespace that will have the results of the scanned resources of that namespace.

To view policyreport in a namespace

```
kubectl -n application get policyreport
NAME                  PASS   FAIL   WARN   ERROR   SKIP   AGE
polr-ns-application   66     4      0      0       0      17d
```

Message field in the below output gives details on how to resolve the issue

To identify which resource has failed against which policy

```
kubectl -n application describe policyreport polr-ns-application | grep "Result: \+fail" -B10
    Seconds:  1643373339
  Category:   Pod Security Standards (Restricted)
  Message:    validation error: Running as root is not allowed. The fields spec.securityContext.runAsNonRoot,
spec.containers[*].securityContext.runAsNonRoot, and spec.initContainers[*].securityContext.runAsNonRoot must
be `true`. Rule autogen-check-containers[0] failed at path /spec/template/spec/securityContext/runAsNonRoot/.
Rule autogen-check-containers[1] failed at path /spec/template/spec/containers/0/securityContext/.
  Policy:     require-run-as-non-root
  Resources:
    API Version:  apps/v1
    Kind:         Deployment
    Name:         nginx-deployment
    Namespace:    application
    UID:          e4abe5c3-6146-4ebb-a0d6-30fdb7014c1c
  Result:         fail
```

How to resolve failed policy report resources

deny-privilege-escalation: Privilege escalation is disallowed. The fields *spec.containers[*].securityContext.allowPrivilegeEscalation*, and *spec.initContainers[*].securityContext.allowPrivilegeEscalation*
must be undefined or set to `false`.

```
spec:
  initContainers:
    - name: demo-init-pod
      securityContext:
        allowPrivilegeEscalation: "false"
  containers:
    - name: demo-pod
      securityContext:
        allowPrivilegeEscalation: "false"
```

disallow-add-capabilities: Adding of additional capabilities beyond the default set is not allowed. The fields *spec.containers[*].securityContext.capabilities.add* must be empty.

```
spec:
  containers:
    - name: demo-pod
      securityContext:
        capabilities:
          X(add): "null"
```

Negation X() The tag cannot be specified. The value of the tag is not evaluated (use exclamation point to negate a value). The value should ideally be set to null.
e.g. Hostpath tag cannot be defined.
X(hostPath):

disallow-host-namespaces: Sharing the host namespaces is disallowed. The fields *spec.hostNetwork, spec.hostIPC,* and *spec.hostPID* must not be set to true.

```
spec:
  hostPID: "false"
  hostIPC: "false"
  hostNetwork: "false"
```

disallow-host-path: HostPath volumes are forbidden. The fields *spec.volumes[*].hostPath* must not be set.

```
spec:
  volumes:
    - X(hostPath): "null"
```

Negation X() The tag cannot be specified. The value of the tag is not evaluated (use exclamation point to negate a value). The value should ideally be set to null.
e.g. Hostpath tag cannot be defined.
X(hostPath):

disallow-host-ports: Use of host ports is disallowed. The fields *spec.containers[*].ports[*].hostPort* and *spec.initContainers[*].ports[*].hostPort* must be empty.

```
spec:
  initContainers:
    - ports:
        - X(hostPort): 0
  containers:
    - ports:
        - X(hostPort): 0
```

disallow-privileged-containers: Privileged mode is disallowed. The fields *spec.containers[*].securityContext.privileged* and *spec.initContainers[*].securityContext.privileged* must not be set to true.

```
spec:
  initContainers:
    - name: demo-init-pod
      securityContext:
        privileged: "false"
  containers:
    - name: demo-pod
      securityContext:
        privileged: "false"
```

disallow-selinux: Setting custom SELinux options is disallowed. The fields *spec.securityContext.seLinuxOptions, spec.containers[*].securityContext.seLinuxOptions,* and *spec.initContainers[*].securityContext.seLinuxOptions* must be empty.

```
spec:
  securityContext:
    seLinuxOptions: "null"
  initContainers:
    - name: demo-init-pod
      securityContext:
        seLinuxOptions: "null"
  containers:
    - name: demo-pod
      securityContext:
        seLinuxOptions: "null"
```

require-default-proc-mount: Changing the proc mount from the default is not allowed. The fields *spec.containers[*].securityContext.procMount* and *spec.initContainers[*].securityContext.procMount*
must not be changed from `Default`.

```
spec:
  initContainers:
    - name: demo-init-pod
      securityContext:
        procMount: "Default"
  containers:
    - name: demo-pod
      securityContext:
        procMount: "Default"
```

require-non-root-groups:

- **check-runasgroup:** Running with root group IDs is disallowed. The fields *spec.securityContext.runAsGroup, spec.containers[*].securityContext.runAsGroup* must be empty or greater than zero.

```
    spec:
      securityContext:
        runAsGroup: ">0"
      containers:
        - name: demo-pod
          securityContext:
            runAsGroup: ">0"
```

- **check-supplementalGroups:** Adding of supplemental group IDs is not allowed. The field *spec.securityContext.supplementalGroups* must not be defined.

```
spec:
  securityContext:
  supplementalGroups: ">0"
```

- **check-fsGroup**: Changing to root group ID is disallowed. The field *spec.securityContext.fsGroup* must be empty or greater than zero.

```
spec:
  securityContext:
  fsGroup: ">0"
```

require-run-as-non-root: Running as root is not allowed. The fields *spec.securityContext.runAsNonRoot, spec.containers[*].securityContext.runAsNonRoot* must be `true`.

> ⚠ Make sure the process running inside the container doesn't use UID=0 (root). Kubernetes will fail to create pod if you configure runAsNonRoot: true and your process is expecting to run with root user.

```
    spec:
      securityContext:
        runAsNonRoot: true
      initContainers:
        - name: demo-init-pod
          securityContext:
            runAsNonRoot: true
      containers:
        - name: demo-pod
          securityContext:
            runAsNonRoot: true
```

restrict-apparmor-profiles: Specifying other AppArmor profiles is disallowed. The annotation *container.apparmor.security.beta.kubernetes.io* must not be defined, or must not be set to anything other than `runtime/default`.

```
    metadata:
      annotations:
        container.apparmor.security.beta.kubernetes.io/*: "runtime/default"
```

restrict-seccomp: Use of custom Seccomp profiles is disallowed. The fields *spec.securityContext.seccompProfile.type, spec.containers[*].securityContext.seccompProfile.type, spec.initContainers[*].securityContext.seccompProfile.type,* and *spec.ephemeralContainers[*].securityContext.seccompProfile.type* must not be set, or set to `RuntimeDefault` or `Localhost`.

```
    spec:
      securityContext:
        seccompProfile:
          type: "RuntimeDefault | Localhost"
      containers:
      - securityContext:
          seccompProfile:
          type: "RuntimeDefault | Localhost"
      initContainers:
      - securityContext:
          seccompProfile:
          type: "RuntimeDefault | Localhost"
      ephemeralContainers:
      - securityContext:
          seccompProfile:
          type: "RuntimeDefault | Localhost"
```

restrict-sysctls: Setting additional sysctls above the allowed type is disallowed. The field *spec.securityContext.sysctls* must not use any other names than 'kernel.shm_rmid_forced', 'net.ipv4.ip_local_port_range', 'net.ipv4.tcp_syncookies' and 'net.ipv4.ping_group_range'.

Recommendations

Running pods as non-root user:

- Update Dockerfile to create required user and group while building the image

Dockerfile

```
FROM busybox

RUN addgroup --gid 1001 oracle

RUN adduser -u 1001 -G oracle -h /home/app oracle -D -H

RUN mkdir /home/app

USER oracle

WORKDIR /home/app
```

- Update Pod manifest file to ensure pods runs with the created userid and groupid

Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  labels:
    name: myapp
spec:
  securityContext:
    runAsNonRoot: true
    fsGroup: 1001
  containers:
  - name: myapp
    image: local/busybox
    command: ['sleep']
    args: ['3600']
    resources:
      limits:
        memory: "128Mi"
        cpu: "500m"
    securityContext:
      runAsNonRoot: true
      runAsGroup: 1001
      runAsUser: 1001
```

Standards to follow for pods using NFS backed Persistent Volumes

UserId and GroupID are only recommendations and not strict guidelines to follow. Teams who has requirements to use different UID and GID to access data from NFS can update accordingly

- Running pods as non-root is a prerequisite, please follow above mentioned guidelines
- Team whose application are dependent on NFS backed PV's need to reach to us, we will update the access level permissions of existing files /directories to match with the that of user and group of the running application

Need More Help ?

Reach us at slack channel #oal-oke-uptake for any further clarifications