Role Based Access Control Good Practices

Archived: 2026-04-05 21:38:08 UTC

Principles and practices for good RBAC design for cluster operators.

Kubernetes RBAC is a key security control to ensure that cluster users and workloads have only the access to resources required to execute their roles. It is important to ensure that, when designing permissions for cluster users, the cluster administrator understands the areas where privilege escalation could occur, to reduce the risk of excessive access leading to security incidents.

The good practices laid out here should be read in conjunction with the general RBAC documentation.

General good practice

Least privilege

Ideally, minimal RBAC rights should be assigned to users and service accounts. Only permissions explicitly required for their operation should be used. While each cluster will be different, some general rules that can be applied are :

- Assign permissions at the namespace level where possible. Use RoleBindings as opposed to ClusterRoleBindings to give users rights only within a specific namespace.
- Avoid providing wildcard permissions when possible, especially to all resources. As Kubernetes is an extensible system, providing wildcard access gives rights not just to all object types that currently exist in the cluster, but also to all object types which are created in the future.
- Administrators should not use `cluster-admin` accounts except where specifically needed. Providing a low privileged account with impersonation rights can avoid accidental modification of cluster resources.
- Avoid adding users to the `system:masters` group. Any user who is a member of this group bypasses all RBAC rights checks and will always have unrestricted superuser access, which cannot be revoked by removing RoleBindings or ClusterRoleBindings. As an aside, if a cluster is using an authorization webhook, membership of this group also bypasses that webhook (requests from users who are members of that group are never sent to the webhook)

Minimize distribution of privileged tokens

Ideally, pods shouldn't be assigned service accounts that have been granted powerful permissions (for example, any of the rights listed under privilege escalation risks). In cases where a workload requires powerful permissions, consider the following practices:

- Limit the number of nodes running powerful pods. Ensure that any DaemonSets you run are necessary and are run with least privilege to limit the blast radius of container escapes.

- Avoid running powerful pods alongside untrusted or publicly-exposed ones. Consider using Taints and Toleration, NodeAffinity, or PodAntiAffinity to ensure pods don't run alongside untrusted or less-trusted Pods. Pay special attention to situations where less-trustworthy Pods are not meeting the **Restricted** Pod Security Standard.

Hardening

Kubernetes defaults to providing access which may not be required in every cluster. Reviewing the RBAC rights provided by default can provide opportunities for security hardening. In general, changes should not be made to rights provided to `system:` accounts some options to harden cluster rights exist:

- Review bindings for the `system:unauthenticated` group and remove them where possible, as this gives access to anyone who can contact the API server at a network level.
- Avoid the default auto-mounting of service account tokens by setting `automountServiceAccountToken: false`. For more details, see using default service account token. Setting this value for a Pod will overwrite the service account setting, workloads which require service account tokens can still mount them.

Periodic review

It is vital to periodically review the Kubernetes RBAC settings for redundant entries and possible privilege escalations. If an attacker is able to create a user account with the same name as a deleted user, they can automatically inherit all the rights of the deleted user, especially the rights assigned to that user.

Kubernetes RBAC - privilege escalation risks

Within Kubernetes RBAC there are a number of privileges which, if granted, can allow a user or a service account to escalate their privileges in the cluster or affect systems outside the cluster.

This section is intended to provide visibility of the areas where cluster operators should take care, to ensure that they do not inadvertently allow for more access to clusters than intended.

Listing secrets

It is generally clear that allowing `get` access on Secrets will allow a user to read their contents. It is also important to note that `list` and `watch` access also effectively allow for users to reveal the Secret contents. For example, when a List response is returned (for example, via `kubectl get secrets -A -o yaml`), the response includes the contents of all Secrets.

Workload creation

Permission to create workloads (either Pods, or workload resources that manage Pods) in a namespace implicitly grants access to many other resources in that namespace, such as Secrets, ConfigMaps, and PersistentVolumes that can be mounted in Pods. Additionally, since Pods can run as any ServiceAccount, granting permission to create workloads also implicitly grants the API access levels of any service account in that namespace.

Users who can run privileged Pods can use that access to gain node access and potentially to further elevate their privileges. Where you do not fully trust a user or other principal with the ability to create suitably secure and isolated Pods, you should enforce either the **Baseline** or **Restricted** Pod Security Standard. You can use Pod Security admission or other (third party) mechanisms to implement that enforcement.

For these reasons, namespaces should be used to separate resources requiring different levels of trust or tenancy. It is still considered best practice to follow least privilege principles and assign the minimum set of permissions, but boundaries within a namespace should be considered weak.

Persistent volume creation

If someone - or some application - is allowed to create arbitrary PersistentVolumes, that access includes the creation of `hostPath` volumes, which then means that a Pod would get access to the underlying host filesystem(s) on the associated node. Granting that ability is a security risk.

There are many ways a container with unrestricted access to the host filesystem can escalate privileges, including reading data from other containers, and abusing the credentials of system services, such as Kubelet.

You should only allow access to create PersistentVolume objects for:

- Users (cluster operators) that need this access for their work, and who you trust.
- The Kubernetes control plane components which creates PersistentVolumes based on PersistentVolumeClaims that are configured for automatic provisioning. This is usually setup by the Kubernetes provider or by the operator when installing a CSI driver.

Where access to persistent storage is required trusted administrators should create PersistentVolumes, and constrained users should use PersistentVolumeClaims to access that storage.

Access to `proxy` subresource of Nodes

Users with access to the `nodes/proxy` sub-resource have rights to the Kubelet API, which allows for command execution on every pod on the node(s) to which they have rights. This access bypasses audit logging and admission control, so care should be taken before granting any rights to this resource. These APIs can be exercised via websocket HTTP `GET` requests, which only requires authorization of the **get** verb. This means that **get** permission on `nodes/proxy` is not a read-only permission.

See Kubelet authentication/authorization for more information.

Escalate verb

Generally, the RBAC system prevents users from creating clusterroles with more rights than the user possesses. The exception to this is the `escalate` verb. As noted in the RBAC documentation, users with this right can effectively escalate their privileges.

Bind verb

Similar to the `escalate` verb, granting users this right allows for the bypass of Kubernetes in-built protections against privilege escalation, allowing users to create bindings to roles with rights they do not already have.

Impersonate verb

This verb allows users to impersonate and gain the rights of other users in the cluster. Care should be taken when granting it, to ensure that excessive permissions cannot be gained via one of the impersonated accounts.

CSRs and certificate issuing

The CSR API allows for users with `create` rights to CSRs and `update` rights on `certificatesigningrequests/approval` where the signer is `kubernetes.io/kube-apiserver-client` to create new client certificates which allow users to authenticate to the cluster. Those client certificates can have arbitrary names including duplicates of Kubernetes system components. This will effectively allow for privilege escalation.

Token request

Users with `create` rights on `serviceaccounts/token` can create TokenRequests to issue tokens for existing service accounts.

Control admission webhooks

Users with control over `validatingwebhookconfigurations` or `mutatingwebhookconfigurations` can control webhooks that can read any object admitted to the cluster, and in the case of mutating webhooks, also mutate admitted objects.

Namespace modification

Users who can perform **patch** operations on Namespace objects (through a namespaced RoleBinding to a Role with that access) can modify labels on that namespace. In clusters where Pod Security Admission is used, this may allow a user to configure the namespace for a more permissive policy than intended by the administrators. For clusters where NetworkPolicy is used, users may be set labels that indirectly allow access to services that an administrator did not intend to allow.

Kubernetes RBAC - denial of service risks

Object creation denial-of-service

Users who have rights to create objects in a cluster may be able to create sufficient large objects to create a denial of service condition either based on the size or number of objects, as discussed in etcd used by Kubernetes is vulnerable to OOM attack. This may be specifically relevant in multi-tenant clusters if semi-trusted or untrusted users are allowed limited access to a system.

One option for mitigation of this issue would be to use resource quotas to limit the quantity of objects which can be created.

What's next

- To learn more about RBAC, see the RBAC documentation.
