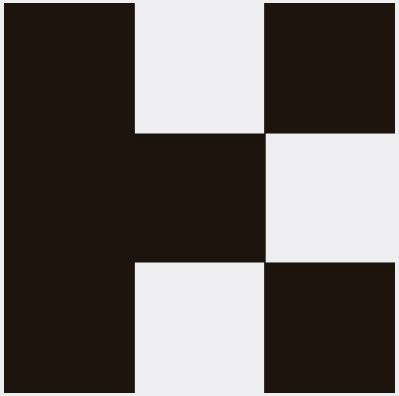
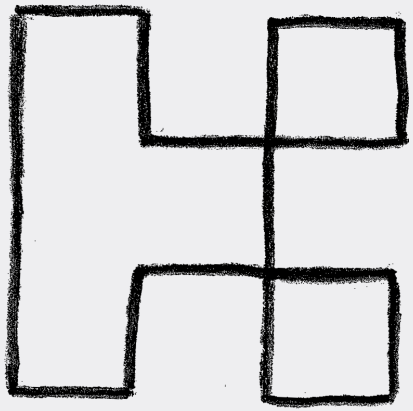


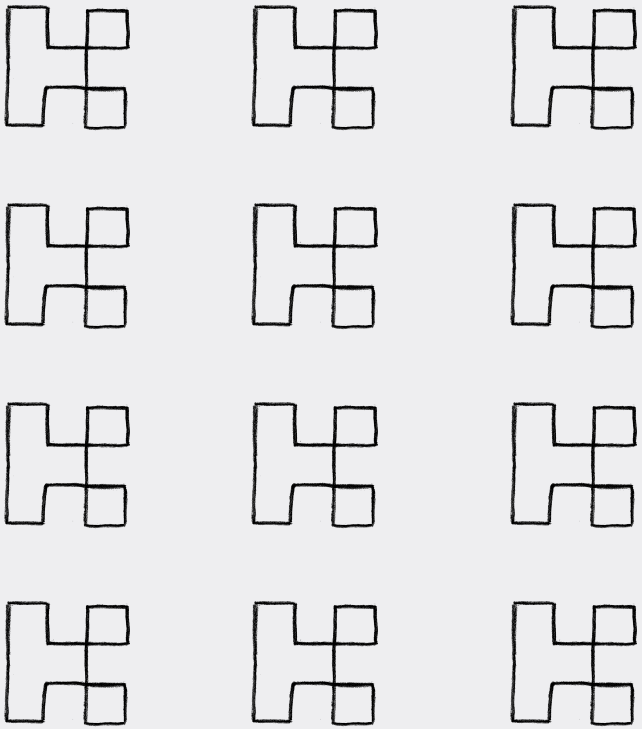
RBAC to the Future: Untangling Authorization in Kubernetes

Jimmy Mesta
CTO / Co-Founder, KSOC



tinyurl.com/rbac-to-the-future

Contents



Kubernetes is an open-source platform built to automate deployment, scaling and orchestration of containers

Identity is the new perimeter.

Access control in Kubernetes can be complicated.

K00 I Introduction

K01 I Insecure Workload Configurations

K02 I Supply Chain Vulnerabilities

K03 I Overly Permissive RBAC Configurations

K04 I Lack of Centralized Policy Enforcement

K05 I Inadequate Logging and Monitoring

K06 I Broken Authentication Mechanisms

K07 I Missing Network Segmentation Controls

K08 I Secrets Management Failures

K09 I Misconfigured Cluster Components

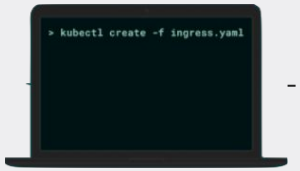
K10 I Outdated and Vulnerable Kubernetes Components

OWASP Kubernetes Top Ten

owasp.org/www-project-kubernetes-top-ten

@jimmesta

API Request Flow



kubectl user

pod

Service Account

client-go

API Server

Authentication

Is this a valid user?

Authorization

What can this user do?

Admission Control

Which policies govern this request?

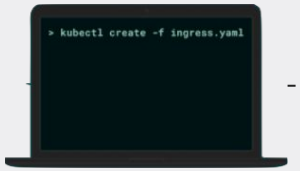
etcd

Node 1

Node 2

@jimmesta

API Request Flow



kubectl user

pod

Service Account

client-go

API Server

Authentication

Is this a valid user?

Authorization

What can this user do?

Admission Control

Which policies govern this request?

etcd

Node 1

Node 2

@jimmesta

Role-Based Access Control enables fine grained access for users, groups, and service accounts within Kubernetes. RBAC can be extremely difficult to scope appropriately which opens up additional privileges.

Users	API Resources	Operations
you@email.com	Namespaces	Get
Service Account	Pod	List
	Service	Delete
	Secrets	Patch

RBAC: Role

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: dev-group-1
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["*"]
    verbs: ["get",”watch”,”list”]
```

RBAC:ClusterRole

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: dev-group-1
rules:
- apiGroups: [""]
  resources: ["secrets"]
    verbs: ["get",”watch”,”list”]
```

RBAC: RoleBinding

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
 -kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

RBAC: ClusterRoleBinding

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
subjects:
 -kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

User-facing ClusterRoles

cluster-admin	Allows super-user access to perform any action on any resource.
admin	Allows admin access, intended to be granted within a namespace using a RoleBinding.
edit	Allows read/write access to most objects in a namespace. This role does not allow viewing or modifying roles or role bindings
view	Allows read-only access to see most objects in a namespace. It does not allow viewing roles or role bindings.

https://kubernetes.io/docs/concepts/security/rbac-good-practices/

What's wrong with this picture?

```yaml
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
 name: redacted-rbac
subjects:
 - kind: ServiceAccount
   name: default
   namespace: default
roleRef:
 kind: ClusterRole
 name: cluster-admin
 apiGroup: rbac.authorization.k8s.io
```

@jimmesta

RBAC Aggregate Permissions

```
# rakkess --namespace default
NAME                                               LIST  CREATE  UPDATE  DELETE
bindings                                                 ✖
configmaps                                         ✔     ✔       ✔       ✔
controllerrevisions.apps                           ✔     ✖       ✖       ✖
cronjobs.batch                                     ✔     ✔       ✔       ✔
daemonsets.apps                                    ✔     ✔       ✔       ✔
daemonsets.extensions                              ✔     ✔       ✔       ✔
deployments.apps                                   ✔     ✔       ✔       ✔
deployments.extensions                             ✔     ✔       ✔       ✔
endpoints                                          ✔     ✔       ✔       ✔
events                                             ✔     ✖       ✖       ✖
events.events.k8s.io                               ✖     ✖       ✖       ✖
horizontalpodautoscalers.autoscaling               ✔     ✔       ✔       ✔
ingresses.extensions                               ✔     ✔       ✔       ✔
jobs.batch                                         ✔     ✔       ✔       ✔
leases.coordination.k8s.io                         ✖     ✖       ✖       ✖
limitranges                                        ✔     ✖       ✖       ✖
localsubjectaccessreviews.authorization.k8s.io           ✖
networkpolicies.extensions                         ✔     ✔       ✔       ✔
```

api-policy
service account

Assigned Roles and Cluster Roles

Name	Kind	Namespace
api-policy-ksoc-pod-readonly	cluster role	*
api-policy	role	ksoc

Aggregate Permissions

Namespace: Cluster-wide

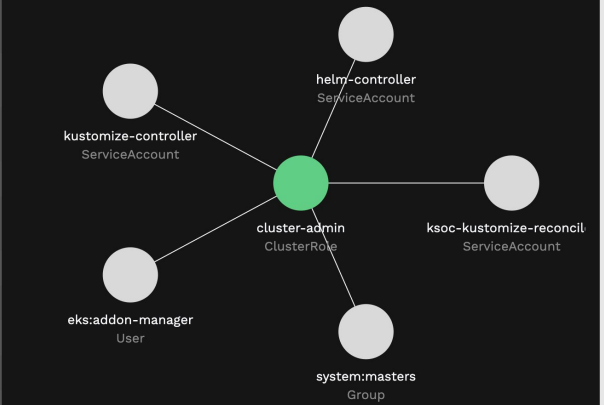
Object	Verbs
pods	get, list, watch

cluster-admin
cluster role

Permissions

Object	Verbs
*	*

RBAC Graph



Assigned Subjects

Subject Name	Kind
system:masters	Group
eks:addon-manager	User
kustomize-controller	ServiceAccount
helm-controller	ServiceAccount

github.com/corneliusweig/rakkess
github.com/alcideio/rbac-tool
github.com/appvia/krane

@jimmesta

Overly Permissive RBAC Configurations

```
curl http://127.0.0.1:8001/api/v1/namespaces/default/secrets/abc -H "Authorization: Bearer $(kubectl -n default get secrets -ojson | jq '.items[]| select(.metadata.anno
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {
  },
  "status": "Failure",
  "message": "secrets \"abc\" is forbidden: User \"system:serviceaccount:default:only-list-secrets-sa\" cannot get resource \"secrets\" in API group \"\" in the namespa
  "reason": "Forbidden",
  "details": {
    "name": "abc",
    "kind": "secrets"
  },
  "code": 403
}
# Now to get all secrets in the default namespace, despite not having "get" permission
curl http://127.0.0.1:8001/api/v1/namespaces/default/secrets?limit=500 -H "Authorization: Bearer $(kubectl -n default get secrets -ojson | jq '.items[]| select(.metadat
{
  "kind": "SecretList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/namespaces/default/secrets",
    "resourceVersion": "17718246"
  },
  "items": [
  REDACTED : REDACTED
  ]
}
```

Overly Permissive RBAC Configurations

```yaml
kind: Role
apiVersion:
rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: secret-list
rules:
- apiGroups: [""]
  resources: ["secrets"]
     verbs: ["list"]
```

GET Request

```
curl http://127.0.0.1:8001/api/v1/namespaces/default/secrets/abc
  "kind": "Status",=
  "metadata": {
  },
  "status": "Failure",
  (...))
  },
  "code": 403
}
```

LIST Request

```
curl http://127.0.0.1:8001/api/v1/namespaces/default/secrets?limit=500
{
  "kind": "SecretList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/namespaces/default/secrets",
  },
  "items": [
  foo : bar
  ]
}
```

```
kubectl get secrets -A -o yaml
```

@jimmesta

CVE-2023-30512 (CubeFS)

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cfs-csi-cluster-role
rules:
- apiGroups: [""]
  resources: ["secrets"]
     verbs: [“get”,"list"]
```

CubeFS through 3.2.1 allows Kubernetes cluster-level privilege escalation. This occurs because DaemonSet has cfs-csi-cluster-role and can thus list all secrets, including the admin secret.

ksoc.com/blog/a-busy-week-for-kubernetes-vulnerabilities-addressing-cve-2023-30512-cve-2023-30513

Minimize wildcard use in Roles and ClusterRoles:Resources

Violating Resource	flux
Violation Message	ClusterRole flux set with resources:* for apiGroups:["*"] and verbs:["*"]. Roles and cluster roles should not use wildcards for resource entries.

Violation Details

Description	Kubernetes Roles and ClusterRoles provide access to resources based on sets of objects and actions that can be taken on those objects. It is possible to set either of these to be the wildcard "*" which matches all items. Use of wildcards is not optimal from a security perspective as it may allow for inadvertent access to be granted when new resources are added to the Kubernetes API either as CRDs or in later versions of the product. Remediation: Remove wildcards in Role and ClusterRole resources.
Policy ID	KSOC-K8S-WILDCARD-RESOURCES
Policy Name	policy-0025-role-wildcard-resources
Time First Detected	2022-11-28T12:52:55Z

Violated Manifest

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    helm.fluxcd.io/antecedent: flux:helmrelease/flux
    meta.helm.sh/release-name: flux
    meta.helm.sh/release-namespace: flux
  creationTimestamp: "2022-11-28T11:09:59Z"
  labels:
    app: flux
    app.kubernetes.io/managed-by: Helm
    chart: flux-1.11.2
    heritage: Helm
    release: flux
  name: flux
  resourceVersion: "2899"
  uid: 84ac948f-299e-4413-8593-f85a97d50e2a
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- nonResourceURLs:
  - '*'
  verbs:
  - '*'
```

Detecting and stopping misconfigured RBAC

@jimmesta

Better Logs == Better RBAC

A Kubernetes Environment has the ability to generate logs at a variety of levels from many different components. When logs are not captured, stored, or actively monitored attackers have the ability to exploit vulnerabilities while going largely undetected.

Better Logs == Better RBAC


```
{
  "kind": "Event",
  "verb": "get",
  "user": {
    "username": "serviceaccount:inventory:inventory",
    "uid": "3e94bfaf-8edc-4562-b2ed-44e9a9e565fb",
    "groups": [
      "serviceaccounts",
      "serviceaccounts:inventory",
    ]
  },
  "sourceIPs": [
    "10.16.4.17"
  ],
  "userAgent": "server/v0",
  },
  "responseStatus": {
    "metadata": {},
    "code": 200
  },
  "requestReceivedTimestamp": "2021-07-16T14:33:24.429585Z",
  "stageTimestamp": "2021-07-16T14:33:24.434300Z",
  "annotations": {
    "authorization.k8s.io/decision": "allow",
    "authorization.k8s.io/reason": "RBAC: allowed by RoleBinding \"inventory\" of Role \"inventory\" to ServiceAccount \"inventory/inventory\""
  }
}
```


https://github.com/liggitt/audit2rbac

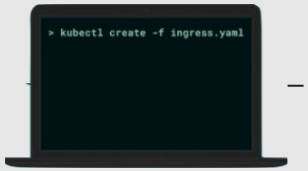
@jimmesta

RBAC Considerations

Watch for the use of the Escalate, Impersonate, and Bind verb as they can lead to RBAC bypass and privilege escalation.

Only grant RBAC permission to create PersistentVolumes when absolutely necessary as those workloads can then access the underlying host filesystem.

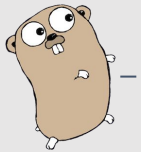
AWS EKS AuthN/AuthZ



kubectl user

pod

Service Account



client-go

IAM Token
Kube Action

Allow/Deny

API Server

IAM Token

K8s Subject

aws-iam-authenticator-server

IAM Token

AWS IAM

IAM Identity

K8s Subject

IAM Identity

aws-auth ConfigMap

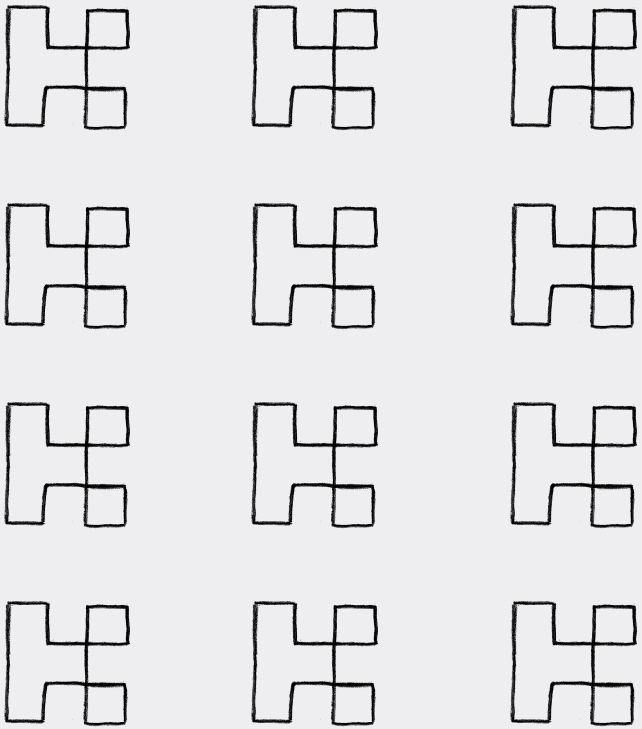
K8s Subject
Kube Action

RBAC

Roles/RoleBindings

Allow/Deny

@jimmesta

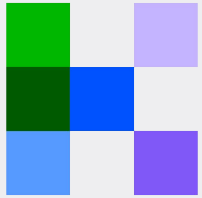


Thanks,
& let’s hang out!

jimmy@ksoc.com

https://github.com/cncf/tag-security/issues/1051

ksoc.com



KSOC

Secure your clusters in real-time