KubeCon | CloudNativeCon

North America 2024

KubeCon CloudNativeCon

North America 2024

Rogue No More: Securing Kubernetes with Node-Specific Restrictions

Speakers

James Munnelly

@munnerz

Anish Ramasekar

@aramase

Background

Background

- Nodes have a set of ‘required’ permissions
- Kubelet, DaemonSet, possibly others
- What prevents one instance modifying/posting another’s information?

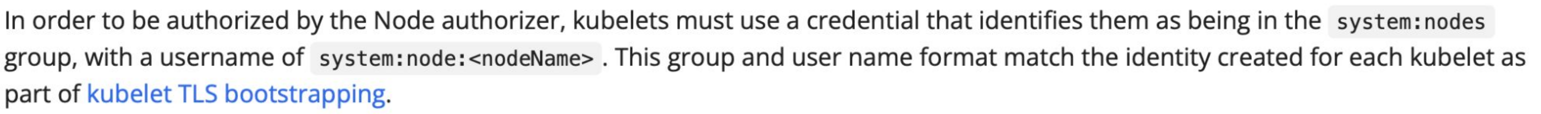
kubelet

Limiting kubelet permissions

- kubelets need to manage/mutate:
  - Pods
  - Nodes
  - PersistentVolumeClaims
  - ServiceAccounts
  - Leases
  - CSINodes
  - CertificateSigningRequests

Limiting kubelet permissions

- NodeRestriction admission plugin:
    - Restricts nodes to only be able to mutate objects ‘associated’ with themselves
    - Inspects authenticated kubelet’s `username` and `groups` in the request
    - Matches against existing object or request body



In order to be authorized by the Node authorizer, kubelets must use a credential that identifies them as being in the `system:nodes` group, with a username of `system:node:<nodeName>`. This group and user name format match the identity created for each kubelet as part of kubelet TLS bootstrapping.

https://kubernetes.io/docs/reference/access-authn-authz/node/#overview

But what about DaemonSets?

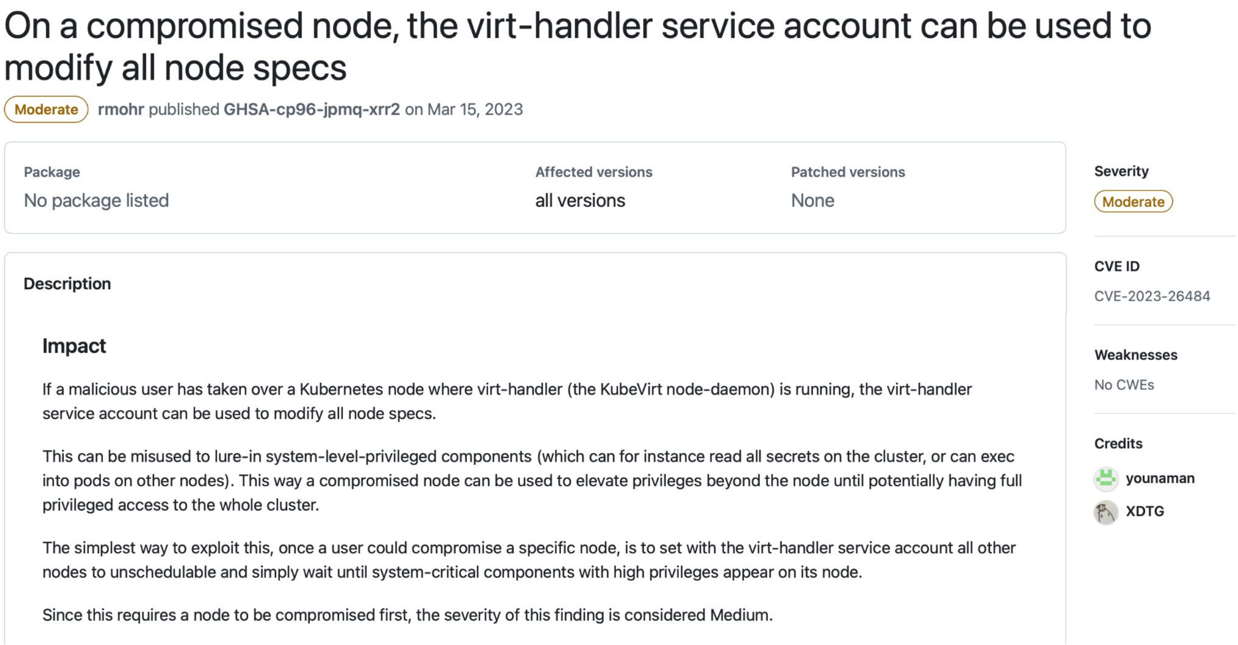
Limiting DaemonSet permissions

- DaemonSet runs across all Nodes in the cluster
- Lack of fine-grained permissions means any instance can modify any object

This attack-vector could allow lateral movement between nodes

Examples - KubeVirt

CVE-2023-26484



On a compromised node, the virt-handler service account can be used to modify all node specs

Moderate **rmohr** published **GHSA-cp96-jpmq-xrr2** on Mar 15, 2023

Package	Affected versions	Patched versions
No package listed	all versions	None

Severity

Moderate

CVE ID

CVE-2023-26484

Weaknesses

No CWEs

Credits

younaman

XDTG

Description

Impact

If a malicious user has taken over a Kubernetes node where virt-handler (the KubeVirt node-daemon) is running, the virt-handler service account can be used to modify all node specs.

This can be misused to lure-in system-level-privileged components (which can for instance read all secrets on the cluster, or can exec into pods on other nodes). This way a compromised node can be used to elevate privileges beyond the node until potentially having full privileged access to the whole cluster.

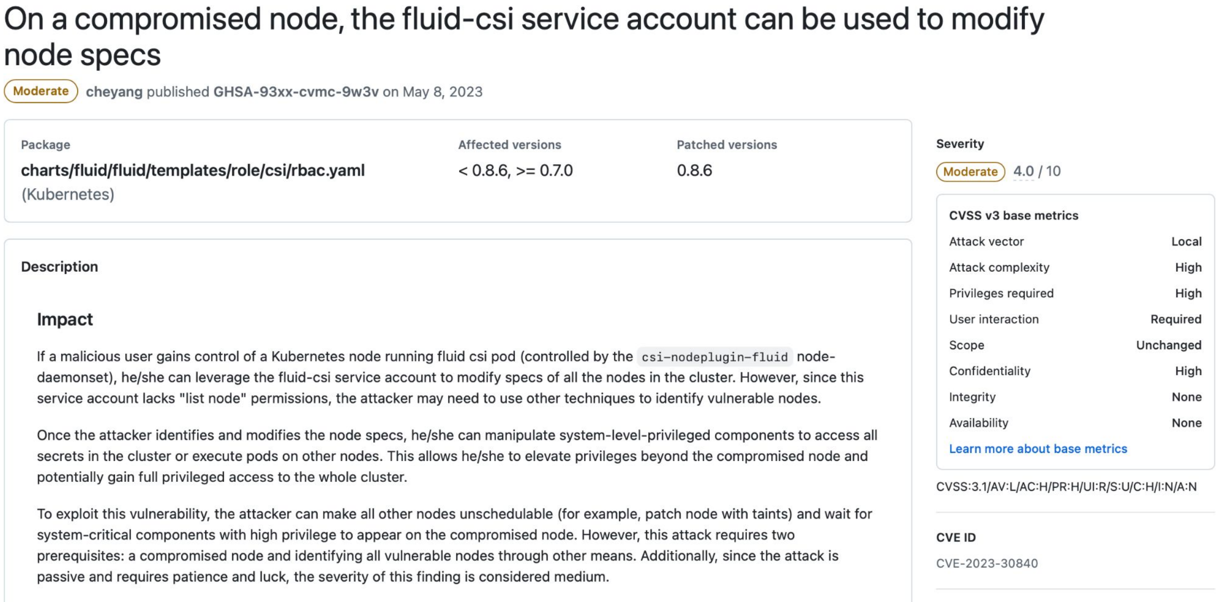
The simplest way to exploit this, once a user could compromise a specific node, is to set with the virt-handler service account all other nodes to unschedulable and simply wait until system-critical components with high privileges appear on its node.

Since this requires a node to be compromised first, the severity of this finding is considered Medium.

https://github.com/kubevirt/kubevirt/security/advisories/GHSA-cp96-jpmq-xrr2

Examples - Fluid

CVE-2023-30840



On a compromised node, the fluid-csi service account can be used to modify node specs

Moderate **cheyang** published **GHSA-93xx-cvmc-9w3v** on May 8, 2023

Package	Affected versions	Patched versions
charts/fluid/fluid/templates/role/csi/rbac.yaml (Kubernetes)	< 0.8.6, >= 0.7.0	0.8.6

Description

Impact

If a malicious user gains control of a Kubernetes node running fluid csi pod (controlled by the `csi-nodeplugin-fluid` node-daemonset), he/she can leverage the fluid-csi service account to modify specs of all the nodes in the cluster. However, since this service account lacks "list node" permissions, the attacker may need to use other techniques to identify vulnerable nodes.

Once the attacker identifies and modifies the node specs, he/she can manipulate system-level-privileged components to access all secrets in the cluster or execute pods on other nodes. This allows he/she to elevate privileges beyond the compromised node and potentially gain full privileged access to the whole cluster.

To exploit this vulnerability, the attacker can make all other nodes unschedulable (for example, patch node with taints) and wait for system-critical components with high privilege to appear on the compromised node. However, this attack requires two prerequisites: a compromised node and identifying all vulnerable nodes through other means. Additionally, since the attack is passive and requires patience and luck, the severity of this finding is considered medium.

Severity

Moderate 4.0 / 10

CVSS v3 base metrics

Attack vector	Local
Attack complexity	High
Privileges required	High
User interaction	Required
Scope	Unchanged
Confidentiality	High
Integrity	None
Availability	None

Learn more about base metrics

CVSS:3.1/AV:L/AC:H/PR:H/UI:R/S:U/C:H/I:N/A:N

CVE ID

CVE-2023-30840

https://github.com/fluid-cloudnative/fluid/security/advisories/GHSA-93xx-cvmc-9w3v

KEP-4193: bound service account token improvements

(we are not very imaginative with naming…)

KEP-4193 overview

- **Embedding the name of the Node a Pod is bound to in Pod bound tokens**
- Permit binding directly to Node objects
- Including the JTI field for precise token tracking in audit logs

https://github.com/kubernetes/enhancements/tree/master/keps/sig-auth/4193-bound-service-account-token-improvements

Changes to service account tokens

```
{
  "aud": [
    "https://kubernetes.default.svc.cluster.local"
  ],
  "exp": 1730388792,
  "iat": 1730385192,
  "iss": "https://kubernetes.default.svc.cluster.local",
  "jti": "0eeb7c44-eba4-4d9c-8b53-51d9c952e28e",
  "kubernetes.io": {
    "namespace": "kube-system",
    "pod": {
      "name": "coredns-55cb58b774-k4ln6",
      "uid": "604fc11d-baf4-4075-9b04-2ac502c15289"
    },
    "serviceaccount": {
      "name": "coredns",
      "uid": "50dd6856-2da2-11e9-9cd9-2afc33b31a7e"
    }
  },
  "nbf": 1730385192,
  "sub": "system:serviceaccount:kube-system:coredns"
}
```

```
{
  "aud": [
    "https://kubernetes.default.svc.cluster.local"
  ],
  "exp": 1730388792,
  "iat": 1730385192,
  "iss": "https://kubernetes.default.svc.cluster.local",
  "jti": "0eeb7c44-eba4-4d9c-8b53-51d9c952e28e",
  "kubernetes.io": {
    "namespace": "kube-system",
    "node": {
      "name": "node1.lab.dev",
      "uid": "646e7c5e-32d6-4d42-9dbd-e504e6cbe6b1"
    },
    "pod": {
      "name": "coredns-55cb58b774-k4ln6",
      "uid": "604fc11d-baf4-4075-9b04-2ac502c15289"
    },
    "serviceaccount": {
      "name": "coredns",
      "uid": "50dd6856-2da2-11e9-9cd9-2afc33b31a7e"
    }
  },
  "nbf": 1730385192,
  "sub": "system:serviceaccount:kube-system:coredns"
}
```

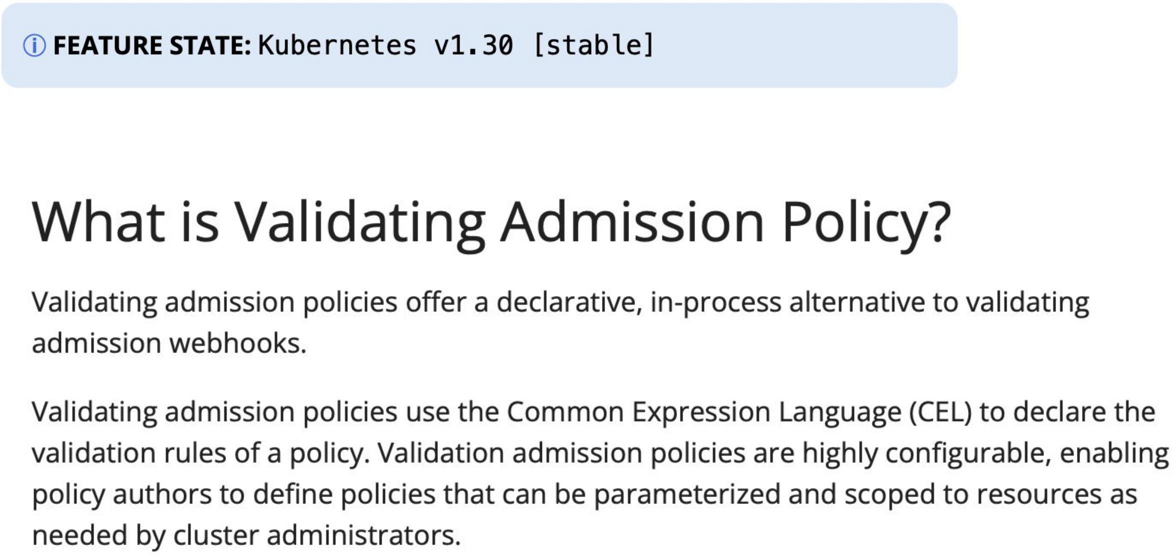
KEP-4193 overview

- This allows admission plugins to utilize embedded Node information in Pod-bound tokens when admitting any kind of object!

http://kep.k8s.io/4193

Combining with `ValidatingAdmissionPolicy`

ValidatingAdmissionPolicy & CEL



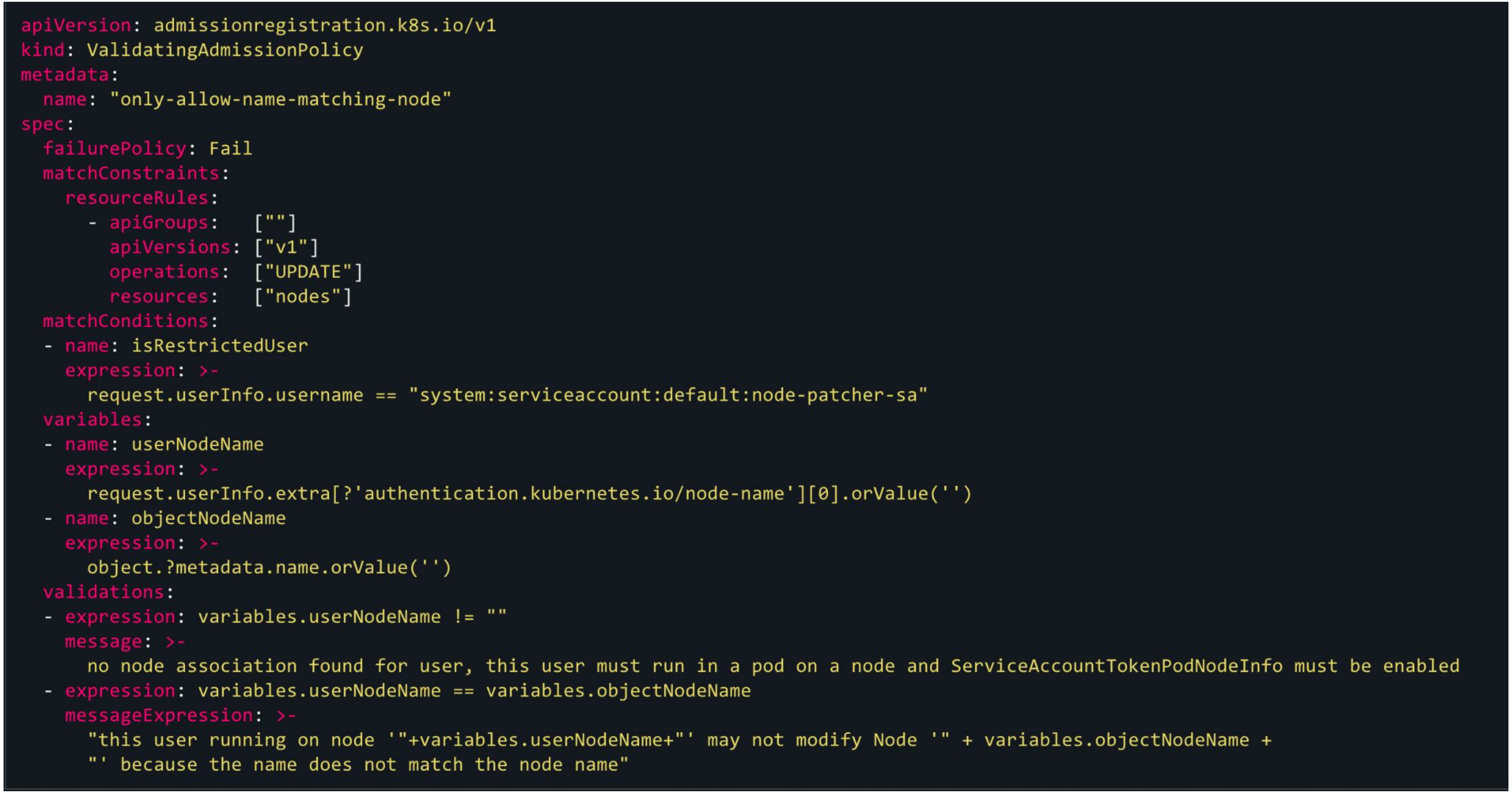
ⓘ **FEATURE STATE:** `Kubernetes v1.30 [stable]`

What is Validating Admission Policy?

Validating admission policies offer a declarative, in-process alternative to validating admission webhooks.

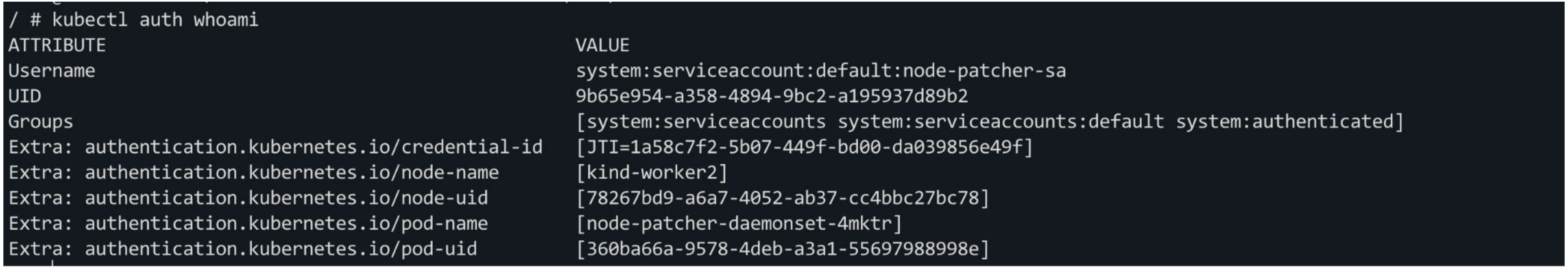
Validating admission policies use the Common Expression Language (CEL) to declare the validation rules of a policy. Validation admission policies are highly configurable, enabling policy authors to define policies that can be parameterized and scoped to resources as needed by cluster administrators.

ValidatingAdmissionPolicy & CEL



```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "only-allow-name-matching-node"
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups:   [""]
        apiVersions: ["v1"]
        operations:  ["UPDATE"]
        resources:   ["nodes"]
  matchConditions:
  - name: isRestrictedUser
    expression: >-
      request.userInfo.username == "system:serviceaccount:default:node-patcher-sa"
  variables:
  - name: userNodeName
    expression: >-
      request.userInfo.extra[?'authentication.kubernetes.io/node-name'][0].orValue('')
  - name: objectNodeName
    expression: >-
      object.?metadata.name.orValue('')
  validations:
  - expression: variables.userNodeName != ""
    message: >-
      no node association found for user, this user must run in a pod on a node and ServiceAccountTokenPodNodeInfo must be enabled
  - expression: variables.userNodeName == variables.objectNodeName
    messageExpression: >-
      "this user running on node '"+variables.userNodeName+"' may not modify Node '" + variables.objectNodeName +
      "' because the name does not match the node name"
```

User info



```
/ # kubectl auth whoami
ATTRIBUTE                                     VALUE
Username                                      system:serviceaccount:default:node-patcher-sa
UID                                           9b65e954-a358-4894-9bc2-a195937d89b2
Groups                                        [system:serviceaccounts system:serviceaccounts:default system:authenticated]
Extra: authentication.kubernetes.io/credential-id   [JTI=1a58c7f2-5b07-449f-bd00-da039856e49f]
Extra: authentication.kubernetes.io/node-name       [kind-worker2]
Extra: authentication.kubernetes.io/node-uid        [78267bd9-a6a7-4052-ab37-cc4bbc27bc78]
Extra: authentication.kubernetes.io/pod-name        [node-patcher-daemonset-4mktr]
Extra: authentication.kubernetes.io/pod-uid         [360ba66a-9578-4deb-a3a1-55697988998e]
```

Demo

Follow-up

- Scoped authentication to image registries
    - Service account token restriction
    - Projected Service Account Tokens for Kubelet Credential Providers KEP

- Check out Squashing Trampoline Pods: The Future of Securely Enabling Hardware Extensions by Joe Betz and David Eads

Give us feedback