RKE2 Hardening Guide with CIS v1.23 Benchmark

Contents

Overview 3

Host-level requirements 4

Setting up hosts 5

Kubernetes runtime requirements 6

API Server audit configuration 8

Known issues 9

Reference Hardened RKE2 Template Configuration 10

Conclusion 15

This document provides prescriptive guidance for hardening a production installation of a RKE2 cluster to be provisioned with Rancher v2.6.5. It outlines the configurations and controls required to address Kubernetes benchmark controls from the Center for Information Security (CIS).

> This hardening guide describes how to secure the nodes in your cluster, and it is recommended to follow this guide before installing Kubernetes.

This hardening guide is intended to be used for RKE2 clusters and associated with specific versions of the CIS Kubernetes Benchmark, Kubernetes, and Rancher:

Rancher Version	CIS Benchmark Version	Kubernetes Version
Rancher v2.6.5+	Benchmark v1.23	Kubernetes v1.22 up to v1.24

Click here to download a PDF version of this document.

- Overview
- Host-level requirements
- Setting up hosts
- Kubernetes runtime requirements
- API Server audit configuration
- Known issues
- Reference Hardened RKE2 Template Configuration
- Conclusion

Overview

This document provides prescriptive guidance for hardening a RKE2 cluster to be provisioned through Rancher v2.6.5+ with Kubernetes v1.22 up to v1.24. It outlines the configurations required to address Kubernetes benchmark controls from the Center for Information Security (CIS).

For more details about evaluating a hardened RKE2 cluster against the official CIS benchmark, refer to the RKE2 - CIS 1.23 Benchmark - Self-Assessment Guide - Rancher v2.6.

RKE2 is designed to be "hardened by default" and pass the majority of the Kubernetes CIS controls without modification. There are a few

notable exceptions to this that require manual intervention to fully pass the CIS Benchmark:

1. RKE2 will not modify the host operating system. Therefore, you, the operator, must make a few host-level modifications.
2. Certain CIS policy controls for `PodSecurityPolicies` and `NetworkPolicies` will restrict the functionality of the cluster. You must opt into having RKE2 configuring these out of the box.

To help ensure these above requirements are met, RKE2 can be started with the `profile` flag set to `cis-1.23`. This flag generally does two things:

1. Checks that host-level requirements have been met. If they haven't, RKE2 will exit with a fatal error describing the unmet requirements.
2. Configures runtime pod security policies and network policies that allow the cluster to pass associated controls.

> The profile's flag only valid values are `cis-1.5` or `cis-1.6` or `cis-1.23`. It accepts a string value to allow for other profiles in the future.

The following section outlines the specific actions that are taken when the `profile` flag is set to `cis-1.23`.

Host-level requirements

There are two areas of host-level requirements: kernel parameters and etcd process/directory configuration. These are outlined in this section.

Ensure `protect-kernel-defaults` is set

This is a kubelet flag that will cause the kubelet to exit if the required kernel parameters are unset or are set to values that are different from the kubelet's defaults.

When the `profile` flag is set, RKE2 will set the flag to `true`.

> `protect-kernel-defaults` is exposed as a configuration flag for RKE2. If you have set `profile` to "cis-1.x" and

> `protect-kernel-defaults` to `false` explicitly, RKE2 will exit with an error.

RKE2 will also check the same kernel parameters that the kubelet does and exit with an error following the same rules as the kubelet. This is done as a convenience to help the operator more quickly and easily identify what kernel parameters are violating the kubelet defaults.

Both `protect-kernel-defaults` and `profile` flags can be set in RKE2 template configuration file.

```
spec:
  rkeConfig:
    machineSelectorConfig:
      - config:
          profile: cis-1.23
          protect-kernel-defaults: true
```

Ensure etcd is configured properly

The CIS Benchmark requires that the etcd data directory be owned by the `etcd` user and group. This implicitly requires the etcd process to be ran as the host-level `etcd` user. To achieve this, RKE2 takes several steps when started with a valid "cis-1.x" profile:

1. Check that the `etcd` user and group exists on the host. If they don't, exit with an error.
2. Create etcd's data directory with `etcd` as the user and group owner.
3. Ensure the etcd process is ran as the `etcd` user and group by setting the etcd static pod's `SecurityContext` appropriately.

Setting up hosts

This section gives you the commands necessary to configure your host to meet the above requirements.

Set kernel parameters

The following `sysctl` configuration is recommended for all nodes type in the cluster. Set the following parameters in `/etc/sysctl.d/90-kubelet.conf`:

```
vm.panic_on_oom=0
vm.overcommit_memory=1
kernel.panic=10
kernel.panic_on_oops=1
```

Run `sudo sysctl -p /etc/sysctl.d/90-kubelet.conf` to enable the settings.

Please perform this step only on fresh installations, before actually deploying RKE2 through Rancher.

Create the etcd user

On some Linux distributions, the `useradd` command will not create a group. The `-U` flag is included below to account for that. This flag tells `useradd` to create a group with the same name as the user.

```
sudo useradd -r -c "etcd user" -s /sbin/nologin -M etcd -U
```

Kubernetes runtime requirements

The runtime requirements to pass the CIS Benchmark are centered around pod security and network policies. These are outlined in this section.

`PodSecurityPolicies`

RKE2 always runs with the `PodSecurityPolicy` admission controller turned on. However, when it is not started with a valid "cis-1.x" profile, RKE2 will put an unrestricted policy in place that allows Kubernetes to run as though the `PodSecurityPolicy` admission controller was not enabled.

When ran with a valid "cis-1.x" profile, RKE2 will put a much more restrictive set of policies in place. These policies meet the requirements outlined in section 5.2 of the CIS Benchmark.

> The Kubernetes control plane components and critical additions such as CNI, DNS, and Ingress are ran as pods in the `kube-system` namespace. Therefore, this namespace will have a policy that is less restrictive so that these components can run properly.

`NetworkPolicies`

When ran with a valid "cis-1.x" profile, RKE2 will put `NetworkPolicies` in place that passes the CIS Benchmark for Kubernetes' built-in namespaces. These namespaces are: `kube-system`, `kube-public`, `kube-node-lease`, and `default`.

The `NetworkPolicy` used will only allow pods within the same namespace to talk to each other. The notable exception to this is that it allows DNS requests to be resolved.

> Operators must manage network policies as normal for additional namespaces that are created.

Configure `default` service account

Set `automountServiceAccountToken` to `false` for `default` service accounts

Kubernetes provides a `default` service account which is used by cluster workloads where no specific service account is assigned to the pod. Where access to the Kubernetes API from a pod is required, a specific service account should be created for that pod, and rights granted to that service account. The `default` service account should be configured such that it does not provide a service account token and does not have any explicit rights assignments.

For each namespace including `default` and `kube-system` on a standard RKE2 install, the `default` service account must include this value:

```
automountServiceAccountToken: false
```

For namespaces created by the cluster operator, the following script and configuration file can be used to configure the `default` service account.

The configuration bellow must be saved to a file called `account_update.yaml`.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
automountServiceAccountToken: false
```

Create a bash script file called `account_update.sh`. Be sure to `sudo chmod +x account_update.sh` so the script has execute permissions.

```
#!/bin/bash -e

for namespace in $(kubectl get namespaces -A -o=jsonpath="{.it
ems[*]['metadata.name']}"); do
  echo -n "Patching namespace $namespace - "
  kubectl patch serviceaccount default -n ${namespace} -p "$(c
at account_update.yaml)"
done
```

Execute this script to apply the `account_update.yaml` configuration to `default` service account in all namespaces.

API Server audit configuration

CIS requirements 1.2.19 to 1.2.22 are related to configuring audit logs for the API Server. When RKE2 is started with the `profile` flag set to `cis-1.23`, it will automatically configure hardened `--audit-log-` parameters in the API Server to pass those CIS checks.

RKE2's default audit policy is configured to not log requests in the API Server. This is done to allow cluster operators flexibility to customize an audit policy that suits their auditing requirements and needs, as these are specific to each users' environment and policies.

A default audit policy is created by RKE2 when started with the `profile` flag set to `cis-1.23`. The policy is defined in `/etc/rancher/rke2/audit-policy.yaml`.

```
apiVersion: audit.k8s.io/v1
kind: Policy
metadata:
  creationTimestamp: null
rules:
- level: None
```

To start logging requests to the API Server, at least `level` parameter must be modified, for example, to `Metadata`. Detailed information about policy configuration for the API server can be found in the Kubernetes documentation.

After adapting the audit policy, RKE2 must be restarted to load the new configuration.

```
sudo systemctl restart rke2-server.service
```

API Server audit logs will be written to `/var/lib/rancher/rke2/server/logs/audit.log`.

Known issues

The following are controls that RKE2 currently does not pass. Each gap will be explained and whether it can be passed through manual operator intervention or if it will be addressed in a future release.

Control 1.1.12

Ensure that the etcd data directory ownership is set to `etcd:etcd`.

Rationale etcd is a highly-available key-value store used by Kubernetes deployments for persistent storage of all of its REST API objects. This data directory should be protected from any unauthorized reads or writes. It should be owned by `etcd:etcd`.

Remediation This can be remediated by creating an `etcd` user and group as described above.

Control 5.1.5

Ensure that default service accounts are not actively used

Rationale Kubernetes provides a `default` service account which is used by cluster workloads where no specific service account is assigned to the pod.

Where access to the Kubernetes API from a pod is required, a specific service account should be created for that pod, and rights granted to that service account.

The `default` service account should be configured such that it does not provide a service account token and does not have any explicit rights assignments.

This can be remediated by updating the `automountServiceAccountToken` field to `false` for the `default` service account in each namespace.

Remediation You can manually update this field on service accounts in your cluster to pass the control as described above.

Control 5.3.2

Ensure that all Namespaces have Network Policies defined

Rationale Running different applications on the same Kubernetes cluster creates a risk of one compromised application attacking a neighboring application. Network segmentation is important to ensure that containers can communicate only with those they are supposed to. A network policy is a specification of how selections of pods are allowed to communicate with each other and other network endpoints.

Network Policies are namespace scoped. When a network policy is introduced to a given namespace, all traffic not allowed by the policy is denied. However, if there are no network policies in a namespace all traffic will be allowed into and out of the pods in that namespace.

Remediation This can be remediated by setting `profile: "cis-1.23"` in RKE2 template configuration file. An example can be found below.

Reference Hardened RKE2 Template Configuration

The reference template configuration is used in Rancher to create a hardened RKE2 custom cluster. This reference does not include other required cluster configuration directives which will vary depending on your environment.

```
apiVersion: provisioning.cattle.io/v1
kind: Cluster
```

```
metadata:
  name: <replace_with_cluster_name>
  annotations:
    {}
#    key: string
  labels:
    {}
#    key: string
  namespace: fleet-default
spec:
  defaultPodSecurityPolicyTemplateName: ''
  kubernetesVersion: <replace_with_kubernetes_version>
  localClusterAuthEndpoint:
    caCerts: ''
    enabled: false
    fqdn: ''
  rkeConfig:
    chartValues:
      rke2-canal:
        {}
    etcd:
      disableSnapshots: false
      s3:
#        bucket: string
#        cloudCredentialName: string
#        endpoint: string
#        endpointCA: string
#        folder: string
#        region: string
#        skipSSLVerify: boolean
      snapshotRetention: 5
      snapshotScheduleCron: 0 */5 * * *
    machineGlobalConfig:
      cni: canal
    machinePools:
#      - cloudCredentialSecretName: string
#        controlPlaneRole: boolean
#        displayName: string
```

```
#       drainBeforeDelete: boolean
#       etcdRole: boolean
#       labels:
#         key: string
#       machineConfigRef:
#         apiVersion: string
#         fieldPath: string
#         kind: string
#         name: string
#         namespace: string
#         resourceVersion: string
#         uid: string
#       machineDeploymentAnnotations:
#         key: string
#       machineDeploymentLabels:
#         key: string
#       machineOS: string
#       maxUnhealthy: string
#       name: string
#       nodeStartupTimeout: string
#       paused: boolean
#       quantity: int
#       rollingUpdate:
#         maxSurge: string
#         maxUnavailable: string
#       taints:
#         - effect: string
#           key: string
#           timeAdded: string
#           value: string
#       unhealthyNodeTimeout: string
#       unhealthyRange: string
#       workerRole: boolean
    machineSelectorConfig:
      - config:
          profile: cis-1.23
          protect-kernel-defaults: true
#     - config:
```

```
#
#       machineLabelSelector:
#         matchExpressions:
#           - key: string
#             operator: string
#             values:
#               - string
#         matchLabels:
#           key: string
    registries:
      configs:
        {}
        #authConfigSecretName: string
#         caBundle: string
#         insecureSkipVerify: boolean
#         tlsSecretName: string
      mirrors:
        {}
        #endpoint:
#           - string
#         rewrite:
#           key: string
    upgradeStrategy:
      controlPlaneConcurrency: 10%
      controlPlaneDrainOptions:
#       deleteEmptyDirData: boolean
#       disableEviction: boolean
#       enabled: boolean
#       force: boolean
#       gracePeriod: int
#       ignoreDaemonSets: boolean
#       ignoreErrors: boolean
#       postDrainHooks:
#         - annotation: string
#       preDrainHooks:
#         - annotation: string
#       skipWaitForDeleteTimeoutSeconds: int
#       timeout: int
```

```
      workerConcurrency: 10%
      workerDrainOptions:
#       deleteEmptyDirData: boolean
#       disableEviction: boolean
#       enabled: boolean
#       force: boolean
#       gracePeriod: int
#       ignoreDaemonSets: boolean
#       ignoreErrors: boolean
#       postDrainHooks:
#         - annotation: string
#       preDrainHooks:
#         - annotation: string
#       skipWaitForDeleteTimeoutSeconds: int
#       timeout: int
#    additionalManifest: string
#    etcdSnapshotCreate:
#      generation: int
#    etcdSnapshotRestore:
#      generation: int
#      name: string
#      restoreRKEConfig: string
#    infrastructureRef:
#      apiVersion: string
#      fieldPath: string
#      kind: string
#      name: string
#      namespace: string
#      resourceVersion: string
#      uid: string
#    provisionGeneration: int
#    rotateCertificates:
#      generation: int
#      services:
#        - string
#    rotateEncryptionKeys:
#      generation: int
  machineSelectorConfig:
```

```
    - config: {}
#  agentEnvVars:
#    - name: string
#      value: string
#  cloudCredentialSecretName: string
#  clusterAPIConfig:
#    clusterName: string
#  defaultClusterRoleForProjectMembers: string
#  enableNetworkPolicy: boolean
#  redeploySystemAgentGeneration: int
__clone: true
```

Conclusion

If you have followed this guide, your RKE2 custom cluster provisioned by Rancher will be configured to pass the CIS Kubernetes Benchmark. You can review our RKE2 CIS Benchmark Self-Assessment Guide v1.23 to understand how we verified each of the benchmarks and how you can do the same on your cluster.