Kubernetes Hardening Guide

This is the working document for the Kubernetes hardening guide. The initial scoping and discussion is In the brainstorming doc . In particular, we should keep in mind the audience for this document and the scope.

Audience

The audience for this document is cluster operators who are interested in hardening/securing the cluster's they run. It may also be useful to other groups of course.

Scope

The scope of this document is Kubernetes and necessary sub-components (e.g. CNI, CRI) as mentioned in the Kubernetes Content Guide

Existing Content

This document is expected to augment things like the existing securing a cluster guide and also a general principal should be that where we can link to existing content on the k/k docs we should, rather than rewriting content

Approach

In contrast to things like the CIS benchmark, which presents a pass/fail set of goals, the expectation is that this document will be risk based and lay out options for administrators to consider and good practices that can be followed.

Example Distribution

Where things like file paths or specific configuration defaults are mentioned we should be consistent with the base distribution used. Kubeadm seems like the logical choice here.

Style

Worth reading the Content Guide and Style Guide before writing.

Section List

There's a number of sections which we could include in this document. If people want to work on a specific section, tag a comment on the section heading. Initial notes in blue are just to provide some ideas of what kind of content we could place in each section, **not intended to be prescriptive, please add more :)**.

Introduction

Kubernetes, like any other software stack, makes a set of choices about default security settings and controls, which need to be customized to suit individual clusters users needs. The goal of this hardening guide is to explore the main areas of Kubernetes cluster security and provide information about the available controls, the defaults which have been chosen and the options available to cluster operators to harden their cluster's security.
The hardening guide is not intended to provide prescriptive controls to meet a level of security, as there are existing documents (for example the CIS benchmark for Kubernetes) which can be used where this is a requirement.

The scope of this hardening guide is core Kubernetes components and any other areas required to get a functioning cluster (for example etcd and CNI).

Threat Model

When thinking about which security configuration options will make sense for your clusters, it is important to consider the threat model you're working to. Depending on the profile of users who can make use of the cluster, and the exposure of the workloads, different configurations will make more sense.

Obviously, from a security standpoint, ideally every cluster would be fully hardened, in most cases there are limited resources which can be applied to security activities, so prioritization of effort is important.

Threat models can sound like a bit of an abstract concept, but they don't need to be really complex to help in securing a cluster. Some examples might be :

- The cluster will be Internet facing and have workloads running multiple web sites available to the general public all deployed by a single team. From this brief description we can see there's some areas where we'd want to place more focus
    - Having the API server on the Internet means we need to ensure that our authentication options are robust and not susceptible to brute force attack
    - Having multiple Internet facing web sites running in the cluster means we need to worry about what happens if one is compromised. Here controls like network policy and hardening the workloads with Security Contexts will be important.
    - With a single team managing the cluster, whilst RBAC is still important, we probably don't need to spend as much effort tightening the rights of operators.
- The cluster will be on a corporate network running a large number of applications from different teams designed for internal use only. Each of the applications already has strong authentication controls to restrict who can make use of them and are not internet facing.
    - Here having a number of teams running applications on the cluster will mean that more attention should be paid to RBAC design to reduce the risk of one team getting access to anthers environment either accidentally or on purpose

    - Also resource management will be more important to ensure that “noisy neighbours” don’t disrupt other applications
    - Whilst network policy is still moderately important as there are a large number of applications, Security Context may be less so, as we’re running relatively trusted applications which have controls on who can access them.

Control Plane Configuration

Some topics we can discuss here

- Permissions for key directories and files used by the control plane..
- Which areas are sensitive for write operations (e.g. most of them) and which are sensitive for read operations as well (e.g. the cryptographic Key files used)
- Per component, recommendations on values for security sensitive parameters (e.g. insecure-port in the API server). In contrast to the CIS Benchmark, I don’t think we need to hit every single param here, but guidance on key ones would be good.
- Whilst it’s not strictly part of the Kubernetes project, it would likely make sense to cover etcd configuration here.

The security of the Kubernetes control plane components is an important part of the overall cluster security as allowing for modification of their configuration or, in some cases, allowing for read access to files used by the control plane could compromise the security of a cluster.

The main control plane components are :-

- API Server
- Scheduler
- Controller Manager

Additionally, depending on the configuration of the cluster the following components may be running on control plane nodes

- Kubelet
- Etcd
- Kube-proxy

API Server Configuration

The Kubernetes API Server has a large number of configuration options . Some of the key parameters to consider, from a security perspective are :

- --allow-privileged (default: false). Setting this flag to false will prevent the use of privileged containers (privileged containers effectively remove all the security isolation mechanisms provided with standard Linux runtimes) at the API server level. Whilst this has obvious security benefits, it is not often set as there are some pods (for example the kube-proxy) which will make use of it. The alternative for controlling the deployment of privileged containers is to use an admission controller which implements pod restrictions.
- --anonymous-auth (default: true). Setting this flag to false will prevent any anonymous access to the API server. When set to true unauthenticated users will be allowed access to resources based on the rights assigned to the

system:unauthenticated group. There are obvious security benefits to setting this flag to false, especially where the API server is exposed to an untrusted network such as the Internet. These benefits need to be balanced against the overhead of requiring valid credentials for any external process (e.g. monitoring services) which needs to access the Kubernetes API server.

- --authorization-mode (default: "AlwaysAllow"). This flag sets the plugins that will be used for authorization checks. As the default is “AlwaysAllow” which means that all authenticated requests will be permitted, it should always be set to a value that does not include AlwaysAllow. Typically setting to RBAC,Node will ensure that authorization requests are handled appropriately. One additional point to note about this flag is that authorization modes are additive, so care should be taken before using additional modes. Setting RBAC enables Kubernetes Role Based Access control, and the Node setting is used by kubelets to use information relating to their operation
- --enable-admission-plugins and --disable-admission-plugins . These two parameters control which admission plugins will operate on the API server. If nothing is specified the default list will be enabled. At a minimum the NodeRestriction plugin should be enabled as it is designed to restrict the kubelet’s rights to the lowest level required for their operation.
- --encryption-provider-config. This parameter should be set in order to ensure that resources which contain sensitive information are encrypted at rest. Whilst secrets should always have this enabled, also consider if there are other resource types in your cluster which contain sensitive information. For example, in some distributions, configmaps may contain sensitive information.
- --token-auth-file. In general token authentication should be avoided and this parameter not set. Token authentication makes use of a flat file of credentials held in clear text on the API server nodes. If this form of authentication is used, the file used to store the tokens should be carefully protected.

Scheduler Configuration

From a security standpoint the scheduler’s configuration is much less complex than the API server, there’s a couple of parameters to watch out for though:

Controller Manager Configuration

File Permissions

An important aspect of maintaining the security of the control plane nodes, is ensuring that file permissions are set appropriately. An attacker who is able to modify (or in some cases view) Kubernetes configuration files, may be able to use that access to compromise the cluster. There are several sets of files which need to be protected.

- **Static Manifest Files -** When using kubeadm, the configuration for the API server, controller manager, scheduler and etcd is generally loaded from static manifests stored in /etc/kubernetes/manifests/ . Write access to this directory should be restricted to the root user only. Any user who can write to the manifests directory can effectively create pods in the cluster, as the kubelet will load any manifest copied to that directory into the cluster
- **Component Kubeconfig files -** The control plane components (scheduler, controller manager and kubelet) have kubeconfig files that are used to authenticate their communications with the API server. Read and write access to these files (stored by default in /etc/kubernetes) should be restricted to the root user only, as anyone with access to them would be able to authenticate to the cluster as that component.
- **Admin Kubeconfig file -** Where the distribution creates a default set of user credentials using client certificate authentication (with Kubeadm this is admin.conf) access to this file should be carefully protected as it is a generic irrevocable administrator account. This file should be removed from the API servers and held in a secure location. If online access to this file is required, permissions should be set such that only the root user can read it, and all access to it should be audited.
- **PKI Files -** Kubernetes clusters have one or more certificate authorities associated with them. The files for these are generally stored on the control plane nodes (For Kubeadm in /etc/kubernetes/pki). The certificate files (ending in .crt) should be owned by the root user and protected from unauthorised modification. The key files should be owned by the root user and only readable or writable by that user. Managing these files is particularly important to the cluster's security as an attacker who is able to read them is likely to be able to gain persistent access to the cluster. In particular, access to the main certificate authority key files allows for creation of cluster-admin level users.
- **CRI Files -** Where the Kubernetes control plane components are running as pods in the cluster, a Container Runtime will be running on the control plane (e.g. Docker, ContainerD, CRI-O). It is important to protect access to the programs and configuration of their CRI, in particular access to any socket files used by the CRI (e.g. docker.sock, containerd.sock) should be closely controlled as unauthorised access to these sockets will allow attackers to compromise the control plane services.

Worker Node Configuration

This is generally going to follow the same path as the Control plane configuration, but focused on the kubelet, kube-proxy, and network plugins. (Ideally this should cover Windows as well as Linux)

PKI Management

Some possible areas for this section would be to look at how cluster owners could securely manage the keys for their certificate authorities. Also looking at the different CAs deployed by a cluster (main, etcd and aggregated API server) and how their separate keys should be managed

Plus details like making sure that the kubelet's TLS listener is using a signed / trusted certificate.

Cluster Authentication

Some areas we could/should cover here are tradeoffs and security concerns for various forms of authentication. Obviously coverage of the in-built mechanisms makes sense (token, client cert) but given that most prod. Clusters will not rely on only the in-built mechanisms, some coverage of the external Authentication options would make sense.

Also seems like this might be a good place to discuss service account token auth, and bootstrap tokens.

A general topic for discussion could include how credential revocation is handled for the in-built options.

Authorization

Topics that could definitely be handled here are the choices & tradeoffs from the different authorization modes and things like the benefits of node authorization. The main topic of discussion (RBAC) may be a separate document (The RBAC Guide) .

Workload Security Configuration

There's a number of topics we might want to cover here. Whilst the technical details of things like Security Context and PSPs/PSS are well covered in existing documents, what could be good is to talk about use cases and tradeoffs and how different options can be applied.

Network Policy Configuration

For this section we could go into common good designs for Network policies in clusters. For example how to set up a default deny ingress/egress policy per namespace and then add individual requirements for applications to communicate inside and outside the cluster.
It could also be worth flagging up specific risks that can be addressed by network policies, like container access to cloud metadata services.
Another topic that we could touch on would be the risks that pods using host networking will bypass network policy controls

Resource Limits

For this section we can look at some of the options for resource limit settings, some tradeoffs/pitfalls to setting limits too low, and perhaps talk about less risky limits that could be set more widely (I'm thinking pid limit here which *I think* is relatively low risk)

Kubernetes Audit Configuration

Here we could look at basic audit configuration, perhaps sample policies (or link to examples like the ones that google provides). As a possibility mentioning remote storage of audit logs (and indeed general component logs)

Add-On configuration

Bit of a catch-all topic. If there's areas to talk about, around things like CoreDNS configuration then that'd definitely make sense, also possibly other add-ons?