StackRox

Guide to Kubernetes Configuration

A comprehensive list of tips, tricks, and best practices for hardening Kubernetes and preventing misconfigurations

Table of Contents

Executive Summary

By now most of us have heard about the role human error plays in causing data breaches. The Capital One breach from July is just the latest in a long line of security incidents that can trace their success back to a misconfigured infrastructure or security setting. As organizations accelerate their use of containers and Kubernetes and move their application development and deployment to cloud platforms, preventing avoidable misconfigurations in their environment becomes increasingly crucial.

Fortunately, most organizations understand that containers and Kubernetes are just like previous waves of infrastructure, where security starts with a securely configured infrastructure. In a recent survey of IT and security practitioners, respondents identified user-driven misconfigurations as their biggest concern for container security.

To help customers securely configure their Docker containers, we recently published our Docker security 101 blog. In this white paper, we will take a deep dive into key Kubernetes security configurations and recommended best practices you should follow.

It should be noted, however, that ensuring adherence to these best practices requires more than just knowing what they are. The level of success you have in consistently following these recommendations will also be determined by the degree to which you can automate the process of checking your environment for misconfigurations.

That's because in a sprawling Kubernetes environment with several clusters spanning tens, hundreds, or even thousands of nodes, created by hundreds of different developers, manually checking the configurations is not feasible. And like all humans, developers can make mistakes – especially given that Kubernetes configuration options are complicated, security features are not enabled by default, and most of the community is learning how to effectively use components including Pod Security Policies and Security Context, Network Policies, RBAC, the API server, kubelet, and other Kubernetes controls.

As you and your teams come up to speed on all the details of Kubernetes security, use the following best practices to build a strong foundation.

1. Update Kubernetes to the latest version

If you haven't already done so, update your Kubernetes deployments to the latest version (1.16), which includes several [new and exciting features](). Every new release is typically bundled with a host of different security features. Be sure to check out our blog post that highlights [7 reasons]() why you should upgrade Kubernetes to the latest version.

2. Use Pod Security Policies to prevent risky containers/Pods from being used

`PodSecurityPolicy` is a cluster-level resources available in Kubernetes (via kubectl) that is highly recommended. You must enable the `PodSecurityPolicy` admission controller to use it. Given the nature of admission controllers, you must authorize at least one policy - otherwise no pods will be allowed to be created in the cluster.

Pod Security Policies address several critical security use cases, including:

- Preventing containers from running with privileged flag - this type of container will have most of the capabilities available to the underlying host. This flag also overwrites any rules you set using CAP DROP or CAP ADD.
- Preventing sharing of host PID/IPC namespace, networking, and ports - this step ensures proper isolation between Docker containers and the underlying host
- Limiting use of volume types - writable hostPath directory volumes, for example, allow containers to write to the filesystem in a manner that allows them to traverse the host filesystem outside the `pathPrefix`, so `readOnly: true` must be used
- Putting limits on host filesystem use
- Enforcing read only for root file system via the ReadOnlyRootFilesystem
- Preventing privilege escalation to root privileges
- Rejecting containers with root privileges
- Restricting Linux capabilities to bare minimum in adherence with least privilege principles

Some of these attributes can also be controlled via `securityContext`. You can learn more about security context at [kubernetes.io/docs/tasks/configure-pod-container/security-context](). However, it's generally recommended that you shouldn't customize the pod-level security context but should instead use Pod Security Policies (see Recommendation #6 on how to apply these controls).

You can learn more about Pod Security Policies at kubernetes.io/docs/concepts/policy/pod-security-policy/.

You can learn more about admission controllers at kubernetes.io/blog/2019/03/21/a-guide-to-kubernetes-admission-controllers/ and at www.stackrox.com/post/2019/03/11-tips-to-operationalizing-kubernetes-admission-controllers-for-better-security/.

3. Use Kubernetes namespaces to properly isolate your Kubernetes resources

Namespaces give you the ability to create logical partitions and enforce separation of your resources as well as limit the scope of user permissions. You can learn more about namespaces kubernetes.io/docs/concepts/overview/working-with-objects/namespaces.

4. Use Network Policies to segment and limit container and pod communication

Network Policies are used to determine how pods are allowed to communicate. Check out our blog post that takes a deep dive into building secure Kubernetes Network Policies.

5. Create policies to govern image provenance using the ImagePolicyWebhook

Prevent unapproved images from being used with the admission controller `ImagePolicyWebhook` to reject pods that use unapproved images including:

- Images that haven't been scanned recently
- Images that use a base image that's not whitelisted
- Images from insecure registries

You can learn more about `ImagePolicyWebhook` at kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#imagepolicywebhook.

6. Securely configure the Kubernetes API server

The Kubernetes API server handles all the REST API calls between external users and Kubernetes components.

Run the below command on your master node:

```
ps -ef | grep kube-apiserver
```

In the output, check to ensure that the:

- `--anonymous-auth` argument shows as `false`. This setting ensures that requests not rejected by other authentication methods are not treated as anonymous and therefore allowed against policy.

- `--basic-auth-file` argument isn't there. Basic auth uses plaintext credentials, instead of the preferred tokens or certificates, for authentication.

- `--insecure-allow-any-token` argument isn't there. This setting will ensure that only secure tokens that are authenticated are allowed.

- `--kubelet-https` argument either isn't there or shows as `true`. This configuration ensures that connections between the API server and the kubelets are protected in transit via Transport Layer Security (TLS).

- `--insecure-bind-address` argument isn't there. This configuration will prevent the API Server from binding to an insecure address, preventing non-authenticated and unencrypted access to your master node, which minimizes your risk of attackers potentially reading sensitive data in transit.

- `--insecure-port` argument shows as `0`. This setting will prevent the API Server from serving on an insecure port, which would prevent unauthenticated and unencrypted access to the master node and minimize the risk of an attacker taking control of the cluster.

- `--secure-port` argument either doesn't exist or shows up as an integer between 1 and 65535. The goal here is to make sure all your traffic is served over https with authentication and authorization.

- `--profiling` argument shows as `false`. Unless you're experiencing bottlenecks or need to troubleshoot something that needs investigation, there's no need for the profiler, and having it there unnecessarily opens you to exposure of system and program details.

- `--repair-malformed-updates` argument shows as `false`. This setting will ensure that intentionally malformed requests from clients are rejected by the API Server.

- `--enable-admission-plugins` argument is set with a value that doesn't contain `AlwaysAdmit`. If you configure this setting to always admit, then it will admit requests even if they're not explicitly allowed by the admissions control plugin, which would decrease the plugin's effectiveness.

- `--enable-admission-plugins` argument is set with a value that contains `AlwaysPullImages`. This configuration ensures that users aren't allowed to pull images from the node to any pod by simply knowing the name of the image. With this control enabled, images will always be pulled prior to starting a container, which will require valid credentials.

- `--enable-admission-plugins` argument is set with a value that contains `SecurityContextDeny`. This control ensures that you can't customize pod-level security context in a way not outlined in the Pod Security Policy. See the Pod Security Policy section (#2) for additional information on security context.

- `--disable-admission-plugins` argument is set with a value that does not contain `NamespaceLifecycle`. You don't want to disable this control, because it ensures that objects aren't created in non-existent namespaces or in those namespaces set to be terminated.

- `--audit-log-path` argument is set to an appropriate path where you want your audit logs to be stored. It's always a good security practice to enable auditing for any Kubernetes components, when available, including the Kubernetes API server.

- `--audit-log-maxage` argument is set to `30` or whatever number of days you must store your audit log files to comply with internal and external data retention policies.

- `--audit-log-maxbackup` argument is set to `10` or any number that helps you meet your compliance requirements for retaining the number of old log files.

- `--audit-log-maxsize` argument is set to `100` or whatever number that helps you meet your compliance requirements. Note that number 100 represents 100 MB.

- `--authorization-mode` argument is there and is not set to `AlwaysAllow`. This setting ensures that only authorized requests are allowed by the API Server, especially in production clusters.

- `--token-auth-file` argument is not there. This argument, when present, uses static token-based authentication, which have several security flaws; use alternate authentication methods instead, such as certificates.

- `--kubelet-certificate-authority` argument is there. This setting helps prevent a man-in-the-middle attack when there's a connection between the API Server and the kubelet.

- `--kubelet-client-certificate` and `--kubelet-client-key` arguments are there. This configuration ensures that the API Server authenticates itself to the kubelet's HTTPS endpoints. (By default, the API Server doesn't take this step.)

- `--service-account-lookup` argument is there and set to `true`. This setting helps prevent an instance where the API Server verifies only the validity of the authentication token without ensuring that the service account token included in the request is present in etcd.

- `--enable-admission-plugins` argument is set to a value that contains `PodSecurityPolicy`. See above section on Pod Security Policies (#2) for more details.

- `--service-account-key-file` argument is there and is set to a separate public/private key pair for signing service account tokens. If you don't specify public/private key pair, it will use the private key from the TLS serving certificate, which would inhibit your ability to rotate the keys for service account tokens.

- `--etcd-certfile` and `--etcd-keyfile` arguments are there so that the API server identifies itself to the etcd server using client cert and key. Note that etcd stores objects that are likely sensitive in nature, so any client connections must use TLS encryption.

- `--disable-admission-plugins` argument is set and doesn't contain `ServiceAccount`. This configuration will make sure that when a new pod is created, it will not use a default service account within the same namespace.

- `--tls-cert-file` and `--tls-private-key-file` arguments are there such that the API Server serves only HTTPS traffic via TLS.

- `--client-ca-file` argument exists to ensure that TLS and client cert authentication is configured for Kube cluster deployments.

- `--etcd-cafile` argument exists and it is set such that the API Server must verify itself to the etcd server via SSL Certificate Authority file.

- `--tls-cipher-suites` argument is set in a way that uses strong crypto ciphers.

- `--authorization-mode` argument is there with a value containing `Node`. This configuration limits which objects kubelets can read associated with their nodes.

- `--enable-admission-plugins` argument is set and contains the value `NodeRestriction`. This plugin ensures that a kubelet is allowed to modify only its own Node API object and those Pod API objects associated to its node.

- `--encryption-provider-config` argument is set to a `EncryptionConfig` file and this file should have all the needed resources. This setting ensures that all the REST API objects stored in the etcd key-value store are encrypted at rest.

- Make sure `aescbc` encryption provider is utilized for all desired resources as this provider of encryption is considered the strongest.

- `--enable-admission-plugins` argument contains the value `EventRateLimit` to set a limit on the number of events accepted by the API Server for performance optimization of the cluster.

- `--feature-gates` argument is not set with a value containing `AdvancedAuditing=false`. In other words, make sure advanced auditing is not disabled for auditing and investigation purposes.

- `--request-timeout` argument is either not set or set to an appropriate value (neither too short, nor too long). Default value is 60 seconds.

- `--authorization-mode` argument exists and is set to a value that includes `RBAC`. This setting ensures that Role-based access control (RBAC) is turned on. Beyond simply turning it on, you should follow several other recommendations for how to best use Kubernetes RBAC, including:

    - Avoid giving users cluster-admin role because it gives very broad powers over the environment and should be used very sparingly, if at all.

    - Audit your role aggregation rules to ensure you're using them properly

    - Don't grant duplicated permissions to subjects because it can make access revocation more difficult

    - Regularly remove unused Roles

7. Securely configure the kube-scheduler

As the default scheduler for Kubernetes, kube-scheduler selects the node that a newly created Pod should run on. You can learn more about kube-scheduler at https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/.

Run the below command on your master node:

```
ps -ef | grep kube-scheduler
```

In the output, check to ensure that the:

- `--profiling` argument is set to `false` so that you have a reduced attack surface. While profiling can be useful when you have a performance bottleneck by identifying the bottleneck, it can also be exploited to reveal details about your system.

- `--address` argument is set to 127.0.0.1 so that the scheduler is not bound to a non-loopback insecure address, since the scheduler API service is available without authentication or encryption.

8. Securely configure the kube-controller-manager

Run the below command on your master node:

```
ps -ef | grep kube-controller-manager
```

In the output, check to ensure that the:

- `--terminated-pod-gc-threshold` argument is set to a value that ensures you have enough resources available and performance isn't degraded.

- `--profilingargument` is set to `false`.

- `--use-service-account-credentials` argument is set to `true`. When combined with RBAC, this setting ensures that control loops run with minimum permissions required to adherence to least privilege design principles.

- `--service-account-private-key-file` argument is set such that a separate public/private key pair is used for signing service account tokens.

- `--root-ca-file` argument exists and is set to a cert file containing the root cert for the API Server's serving cert, which will allow pods to verify the API Server's serving cert before making a connection.

- `RotateKubeletServerCertificate` argument is there and set as `true`, and applies only when kubelets get their certs from the API Server.

- `--address argument` is set to 127.0.0.1, so that the controller manager service is not bound to non-loopback insecure addresses.

9. Secure the configuration files on the master node

Secure the API server pod specification file permissions.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %a /etc/kubernetes/manifests/kube-apiserver.yaml
```

In the output, check to ensure that permissions are `644` or more restrictive to maintain the integrity of the file.

Secure the API Server pod specification file ownership.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %U:%G /etc/kubernetes/manifests/kube-apiserver.yaml
```

In the output, check to ensure that ownership is set as `root:root` to maintain the integrity of the file.

Secure the controller manager pod specification file permissions.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %a /etc/kubernetes/manifests/kube-controller-manager.yaml
```

In the output, check to ensure that permissions are `644` or more restrictive to maintain the integrity of the file.

Secure the controller manager pod specification file ownership.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %U:%G /etc/kubernetes/manifests/kube-controller-manager.yaml
```

In the output, check to ensure that ownership is set as `root:root` to maintain the integrity of the file.

Secure the scheduler pod specification file permissions.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %a /etc/kubernetes/manifests/kube-scheduler.yaml
```

In the output, check to ensure that permissions are `644` or more restrictive to maintain the integrity of the file.

Secure the scheduler pod specification file ownership.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %U:%G /etc/kubernetes/manifests/kube-scheduler.yaml
```

In the output, check to ensure that ownership is set as `root:root` to maintain the integrity of the file.

Secure the etcd pod specification file permissions.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %a /etc/kubernetes/manifests/etcd.yaml
```

In the output, check to ensure that permissions are `644` or more restrictive to maintain the integrity of the file. As a reminder on a topic already discussed, etcd is a key-value store, and protecting it is of the utmost importance, since it contains your REST API objects.

Secure the etcd pod specification file ownership.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %U:%G /etc/kubernetes/manifests/etcd.yaml
```

In the output, check to ensure that ownership is set as `root:root` to maintain the integrity of the file.

Secure the Container Network Interface file permissions.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %a <path/to/cni/files>
```

In the output, check to ensure that permissions are `644` or more restrictive to maintain the integrity of the file.

Secure the Container Network Interface file ownership.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %U:%G <path/to/cni/files>
```

In the output, check to ensure that ownership is set as `root:root` to maintain the integrity of the file.

Secure the etcd data directory permissions.

First run the following command to the get etcd data directory:

```
ps -ef | grep etcd
```

Now run the following command based on the etcd data directory you found from the previous command:

```
stat -c %a /var/lib/etcd
```

In the output, check to ensure that permissions are `700` or more restrictive to ensure your etcd data directory is protected against unauthorized reads/writes.

Secure the etcd data directory ownership.

First run the following command to the get etcd data directory:

```
ps -ef | grep etcd
```

Now run the following command based on the etcd data directory you found from the previous command:

```
stat -c %a /var/lib/etcd
```

In the output, check to ensure that ownership is `etcd:etcd` to ensure your etcd data directory is protected against unauthorized reads/writes.

Secure the admins.conf file permissions.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %a /etc/kubernetes/admin.conf
```

In the output, check to ensure that permissions are `644` or more restrictive to maintain the integrity of the file.

Secure the admins.conf file ownership.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %U%G /etc/kubernetes/admin.conf
```

In the output, check to ensure that ownership is set as `root:root` to maintain the integrity of the file.

Secure the scheduler.conf file permissions.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %a /etc/kubernetes/scheduler.conf
```

In the output, check to ensure that permissions are `644` or more restrictive to maintain the integrity of the file.

Secure the scheduler.conf file ownership.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %U%G /etc/kubernetes/scheduler.conf
```

In the output, check to ensure that ownership is set as `root:root` to maintain the integrity of the file.

Secure the controller-manager.conf file permissions.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %a /etc/kubernetes/controller-manager.conf
```

In the output, check to ensure that permissions are `644` or more restrictive to maintain the integrity of the file

Secure the controller-manager.conf file ownership.

Run the following command on the master node (specifying your file location on your system):

```
stat -c %U%G /etc/kubernetes/controller-manager.conf
```

In the output, check to ensure that ownership is set as `root:root` to maintain the integrity of the file.

Secure the Kubernetes PKI directory and file ownership.

Run the following command on the master node (specifying your file location on your system):

```
ls -laR /etc/kubernetes/pki
```

In the output, check to ensure that ownership is set as `root:root` to maintain the integrity of the file.

Secure the Kubernetes PKI directory and file permissions.

Run the following command on the master node (specifying your file location on your system):

```
ls -laR /etc/kubernetes/pki/*.crt
```

In the output, check to ensure that permissions are `644` or more restrictive to maintain the integrity of the file.

Secure the Kubernetes PKI key file permissions.

Run the following command on the master node (specifying your file location on your system):

```
ls -laR /etc/kubernetes/pki/*.key
```

In the output, check to ensure that permissions are `600` to maintain the integrity of the file.

10. Securely configure etcd

As mentioned in previous sections, etcd (a CNCF project) is a key-value store (a CNCF project) used by distributed systems such as Kubernetes for data access. etcd is considered the source of truth for Kubernetes, and you can read data from and write into etcd as needed. Securely configuring etcd and communications to its servers are of utmost criticality.

Run the following command on the etcd server node:

```
ps -ef | grep etcd
```

In the output, check to ensure that the:

- `--cert-file` and the `--key-file` arguments are set as needed to ensure client connections are served only over TLS (in transit encryption).

- `--client-cert-auth` argument shows as `true` to ensure all access attempts from clients include a valid client cert.

- `--auto-tls` argument is there and is not `true`, or isn't there at all, which will prohibit clients from using self-signed certs for TLS.

- If you're using a etcd cluster (instead of a single etcd server), check to see that `--peer-cert-file` and `--peer-key-file` arguments are appropriately set to ensure etcd peer connections is encrypted within the etcd cluster. In addition, check that `--peer-client-cert-auth` argument is set to `true`, as this setting would ensure that only authenticated etcd peers can access the etcd cluster. Lastly verify that if `--peer-auto-tls` argument is there, it is not set to `true`.

- As a best practice, don't use the same certificate authority for etcd as you do for Kubernetes. You can ensure this separation by verifying that the file referenced by the `--client-ca-file` for API Server is different from the `--trusted-ca-file` used by etcd.

11. Securely configure the Kubelet

The kubelet is the main "node agent" running on each node. Misconfiguring kubelet can expose you to a host of security risks, as this Medium article last year outlines. You can either use arguments on the running kubelet executable or a kubelet config file to set the configuration of your kubelet.

To find the kubelet config file, run the following command:

```
ps -ef | grep kubelet | grep config
```

Look for `--config` argument, which will give you the location of the kubelet config file.

Then run the following command on each node:

```
ps -ef | grep kubelet
```

In the output, make sure that the:

- `--anonymous-auth` argument is `false`. In the kubelet article previously referenced, one of the misconfigurations exploited was one where anonymous (and unauthenticated) requests were allowed to be served by the kubelet server.

- `--authorization-mode` argument shows as `AlwaysAllow` if it's there. If it is not there, make sure there's a kubelet config file specified by `--config` and that file has set `authorization: mode` to something besides `AlwaysAllow`.

- `--client-ca-file` argument is there and set to the location of the client certificate authority file. If it's not there, make sure there's a kubelet config file specified by `--config` and that file has set `authentication: x509: clientCAFile` to the location of the client certificate authority file.

- `--read-only-port` argument is there and set to `0`. If it's not there, make sure there's a kubelet config file specified by `--config`, and `readOnlyPort` is set to `0` if it's there.

- `--protect-kernel-defaults` shows as true. If it's not there, make sure there's a kubelet config file specified by `--config`, and that file has set `protectKernelDefaults` as `true`.

- `--hostname-override` argument is not there, to ensure that the TLS setup between the kubelet and the API Server doesn't break.

- `--event-qps` argument is there and set to `0`. If it's not there, make sure there's a kubelet config file specified by `--config` and `eventRecordQPS` shows as `0`.

- `--tls-cert-file` and `--tls-private-key-file` arguments are set appropriately or the kubelet config specified by `--config` contains appropriate settings for `tlsCertFile` and `tlsPrivateKeyFile`. This configuration ensures that all connections happen over TLS on the kubelets.

- `RotateKubeletServerCertificate` and `--rotate-certificates` is set to `true` if your kubelets get their certs from the API Server, and make sure your kubelet uses only strong crypto ciphers.

12. Secure the worker node configuration files

Secure the kubelet service file permissions.

Run the following command on each worker node (specifying your file location on your system):

```
stat -c %a /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
```

In the output, check to ensure that permissions are `644` or more restrictive to maintain the integrity of the file.

Secure the kubelet.conf file permissions.

Run the following command on each worker node (specifying your file location on your system):

```
stat -c %a /etc/kubernetes/kubelet.conf
```

In the output, check to ensure that permissions are `644` or more restrictive to maintain the integrity of the file.

Secure the kubelet.conf file ownership.

Run the following command on each worker node (specifying your file location on your system):

```
stat -c %U%G /etc/kubernetes/kubelet.conf
```

In the output, check to ensure that ownership is set as `root:root` to maintain the integrity of the file.

Secure the kublete service file ownership.

Run the following command on each worker node (specifying your file location on your system):

```
stat -c %U%G /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
```

In the output, check to ensure that ownership is set as `root:root` to maintain the integrity of the file.

Secure the proxy kubeconfig file permissions.

Run the following command to first find the kubeconfig file being used:

```
ps -ef | grep kube-proxy
```

Get the kube-proxy file location (if it's running) from `--kubeconfig`, then run the following command on each worker node (specifying your file location on your system).

```
stat -c %a <proxy kubeconfig file>
```

In the output, check to make sure permissions are `644` or more restrictive to maintain the integrity of the file.

Secure the proxy kubeconfig file ownership.

Run the following command first to find the kubeconfig file being used:

```
ps -ef | grep kube-proxy
```

Get the kube-proxy file location (if it's running) from `--kubeconfig`, then run the following command on each worker node (specifying your file location on your system):

```
stat -c %U%G <proxy kubeconfig file>
```

In the output, check to make sure ownership is set as `root:root` to maintain the integrity of the file.

Secure the certificate authorities file permissions.

Run the following command first:

```
ps -ef | grep kubelet
```

Look for the file name that's identified by `--client-ca-file` argument. Then run the following command, specifying the previous file name:

```
stat -c %a <filename>
```

In the output, check to make sure permissions are `644` or more restrictive to maintain the integrity of the file.

Secure the client certificate authorities file ownership.

Run the following command first:

```
ps -ef | grep kubelet
```

Look for the file name that's identified by `--client-ca-file` argument. Then run the following command, specifying the previous file name:

```
stat -c %U%G <filename>
```

In the output, check to make sure ownership is set as `root:root` to maintain the integrity of the file.

Secure the kubelet configuration file permissions.

First locate the kubelet config file with following command:

```
ps -ef | grep kubelet | grep config
```

In the output, you may see the location of the config file if it exists. It would look something like /var/lib/kubelet/configuration.yaml.

Using the location of the file (we'll use the file location from this previous example), run the following command to identify the file's permissions:

```
stat -c %a /var/lib/kubelet/configuration.yaml
```

In the output, check to make sure permissions are set to `644` or more restrictive to ensure the integrity of the file.

Secure the kubelet configuration file ownership.

Run the following command:

```
ps -ef | grep kubelet | grep config
```

In the output, you may see the location of the config file if it exists - it would look something like /var/lib/kubelet/configuration.yaml.

Using the location of the file (we'll use the file location from this previous example), run the following command to identify the file's permissions:

```
stat -c %U%G /var/lib/kubelet/configuration.yaml
```

In the output, check to make sure ownership is set to `root:root` to maintain the integrity of the file.

This cloud-native stack offers compelling capabilities for building the most secure applications we've ever created - we just need to make sure we've got all the knobs and dials set correctly. Leverage these configurations, code examples, and detailed recommendations to avoid the security risks associated with the most common Kubernetes misconfigurations.

References

https://www.cisecurity.org/benchmark/kubernetes/

https://docs.docker.com/v17.09/compliance/cis/

https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/

https://kubernetes.io/docs/concepts/policy/pod-security-policy/

https://kubernetes.io/docs/tasks/configure-pod-container/security-context/

https://kubernetes.io/blog/2017/04/rbac-support-in-kubernetes/

https://kubernetes.io/blog/2018/07/18/11-ways-not-to-get-hacked/