CIS Center for Internet Security®

CIS Benchmarks™

CIS Google Kubernetes Engine (GKE) Autopilot Benchmark

v1.0.0 - 07-17-2024

Terms of Use

Please see the below link for our current terms of use:

https://www.cisecurity.org/cis-securesuite/cis-securesuite-membership-terms-of-use/

Table of Contents

Overview

All CIS Benchmarks™ focus on technical configuration settings used to maintain and/or increase the security of the addressed technology, and they should be used in **conjunction** with other essential cyber hygiene tasks like:

- Monitoring the base operating system for vulnerabilities and quickly updating with the latest security patches.
- Monitoring applications and libraries for vulnerabilities and quickly updating with the latest security patches.

In the end, the CIS Benchmarks are designed as a key **component** of a comprehensive cybersecurity program.

This document provides prescriptive guidance for running Google Kubernetes Engine (GKE) AutoPilot following recommended security controls. This benchmark only includes controls which can be modified by an end user of GKE AutoPilot.

To obtain the latest version of this guide, please visit www.cisecurity.org. If you have questions, comments, or have identified ways to improve this guide, please write us at support@cisecurity.org.

Intended Audience

This document is intended for cluster administrators, security specialists, auditors, and any personnel who plan to develop, deploy, assess, or secure solutions that incorporate Google Kubernetes Engine (GKE) AutoPilot.

Relevant links

- GKE Shared Responsibility Model
- GKE Control Plane Security
- GKE AutoPilot Security Capabilities
- GKE AutoPilot vs Standard Security Feature Comparison

Consensus Guidance

This CIS Benchmark™ was created using a consensus review process comprised of a global community of subject matter experts. The process combines real world experience with data-based information to create technology specific guidance to assist users to secure their environments. Consensus participants provide perspective from a diverse set of backgrounds including consulting, software development, audit and compliance, security research, operations, government, and legal.

Each CIS Benchmark undergoes two phases of consensus review. The first phase occurs during initial Benchmark development. During this phase, subject matter experts convene to discuss, create, and test working drafts of the Benchmark. This discussion occurs until consensus has been reached on Benchmark recommendations. The second phase begins after the Benchmark has been published. During this phase, all feedback provided by the Internet community is reviewed by the consensus team for incorporation in the Benchmark. If you are interested in participating in the consensus process, please visit https://workbench.cisecurity.org/.

Typographical Conventions

The following typographical conventions are used throughout this guide:

Convention	Meaning
`Stylized Monospace font`	Used for blocks of code, command, and script examples. Text should be interpreted exactly as presented.
`Monospace font`	Used for inline code, commands, UI/Menu selections or examples. Text should be interpreted exactly as presented.
`<Monospace font in brackets>`	Text set in angle brackets denote a variable requiring substitution for a real value.
Italic font	Used to reference other relevant settings, CIS Benchmarks and/or Benchmark Communities. Also, used to denote the title of a book, article, or other publication.
Bold font	Additional information or caveats things like **Notes**, **Warnings**, or **Cautions** (usually just the word itself and the rest of the text normal).

Recommendation Definitions

The following defines the various components included in a CIS recommendation as applicable. If any of the components are not applicable it will be noted or the component will not be included in the recommendation.

Title

Concise description for the recommendation's intended configuration.

Assessment Status

An assessment status is included for every recommendation. The assessment status indicates whether the given recommendation can be automated or requires manual steps to implement. Both statuses are equally important and are determined and supported as defined below:

Automated

Represents recommendations for which assessment of a technical control can be fully automated and validated to a pass/fail state. Recommendations will include the necessary information to implement automation.

Manual

Represents recommendations for which assessment of a technical control cannot be fully automated and requires all or some manual steps to validate that the configured state is set as expected. The expected state can vary depending on the environment.

Profile

A collection of recommendations for securing a technology or a supporting platform. Most benchmarks include at least a Level 1 and Level 2 Profile. Level 2 extends Level 1 recommendations and is not a standalone profile. The Profile Definitions section in the benchmark provides the definitions as they pertain to the recommendations included for the technology.

Description

Detailed information pertaining to the setting with which the recommendation is concerned. In some cases, the description will include the recommended value.

Rationale Statement

Detailed reasoning for the recommendation to provide the user a clear and concise understanding on the importance of the recommendation.

Impact Statement

Any security, functionality, or operational consequences that can result from following the recommendation.

Audit Procedure

Systematic instructions for determining if the target system complies with the recommendation.

Remediation Procedure

Systematic instructions for applying recommendations to the target system to bring it into compliance according to the recommendation.

Default Value

Default value for the given setting in this recommendation, if known. If not known, either not configured or not defined will be applied.

References

Additional documentation relative to the recommendation.

CIS Critical Security Controls® (CIS Controls®)

The mapping between a recommendation and the CIS Controls is organized by CIS Controls version, Safeguard, and Implementation Group (IG). The Benchmark in its entirety addresses the CIS Controls safeguards of (v7) “5.1 - Establish Secure Configurations” and (v8) '4.1 - Establish and Maintain a Secure Configuration Process” so individual recommendations will not be mapped to these safeguards.

Additional Information

Supplementary information that does not correspond to any other field but may be useful to the user.

Profile Definitions

The following configuration profiles are defined by this Benchmark:

- **Level 1**

  Items in this profile intend to:

  - Be practical and prudent
  - Provide a clear security benefit
  - Do not inhibit the utility of the technology beyond acceptable means

- **Level 2**

  Extends Level 1

Acknowledgements

This Benchmark exemplifies the great things a community of users, vendors, and subject matter experts can accomplish through consensus collaboration. The CIS community thanks the entire consensus team with special recognition to the following individuals who contributed greatly to the creation of this guide:

Authors

Randall Mowen
Andrew Peabody

Key Contributors

Poonam Lamba
Shannon Kularathna
Mark Larinde

Thanks to the Google team who contributed to this benchmark

Padmalatha (Padma) Ragunathan, Vinayak Goyal, Shannon Kularathna, Michele Chubirka, Cynthia Thomas, Glen Messenger, Greg Castle, Michael Taufen

Recommendations

1 Control Plane Components

Under the GKE shared responsibility model, Google GKE AutoPilot mode provides a more managed Kubernetes experience than standard mode and implements more pre-configured security best practices by default. It also utilizes Container-Optimized OS reducing controls required to harden a Kubernetes cluster. You as the end user are responsible for securing your nodes, containers, and Pods and that is what this Benchmark specifically addresses.

2 Control Plane Configuration

Under the GKE shared responsibility model, Google GKE AutoPilot mode provides a more managed Kubernetes experience than standard mode and implements more pre-configured security best practices by default. It also utilizes Container-Optimized OS reducing controls required to harden a Kubernetes cluster. You as the end user are responsible for securing your nodes, containers, and Pods and that is what this Benchmark specifically addresses.

The community has reviewed the Control Plane Configuration Section and have determined that at this time no additional security relevant configuration recommendations are required outside of the *pre-configured* default configurations supplied by Autopilot, which already follow the applicable full CIS GKE Benchmark recommendations.

3 Worker Nodes

Under the GKE shared responsibility model, Google GKE AutoPilot mode provides a more managed Kubernetes experience than standard mode and implements more pre-configured security best practices by default. It also utilizes Container-Optimized OS reducing controls required to harden a Kubernetes cluster. You as the end user are responsible for securing your nodes, containers, and Pods and that is what this Benchmark specifically addresses.

The community has reviewed the Worker Node Section and have determined that at this time additional security relevant recommendations are required outside of the *pre-configured* default configurations supplied by Autopilot, which already follow the applicable full CIS GKE Benchmark recommendations.

4 Policies

This section contains recommendations for various Kubernetes policies which are important to the security of the GKE AutoPilot Cluster environment.

4.1 RBAC and Service Accounts

4.1.1 Ensure that the cluster-admin role is only used where required (Automated)

Profile Applicability:

- Level 1

Description:

The RBAC role `cluster-admin` provides wide-ranging powers over the environment and should be used only where and when needed.

Rationale:

Kubernetes provides a set of default roles where RBAC is used. Some of these roles such as `cluster-admin` provide wide-ranging privileges which should only be applied where absolutely necessary. Roles such as `cluster-admin` allow super-user access to perform any action on any resource. When used in a `ClusterRoleBinding`, it gives full control over every resource in the cluster and in all namespaces. When used in a `RoleBinding`, it gives full control over every resource in the rolebinding's namespace, including the namespace itself.

Impact:

Care should be taken before removing any `clusterrolebindings` from the environment to ensure they were not required for operation of the cluster. Specifically, modifications should not be made to `clusterrolebindings` with the `system:` prefix as they are required for the operation of system components.

Audit:

Obtain a list of the principals who have access to the `cluster-admin` role by reviewing the `clusterrolebinding` output for each role binding that has access to the `cluster-admin` role.

```
kubectl get clusterrolebindings -o=custom-
columns=NAME:.metadata.name,ROLE:.roleRef.name,SUBJECT:.subjects[*].name
```

Review each principal listed and ensure that `cluster-admin` privilege is required for it.

Remediation:

Identify all clusterrolebindings to the cluster-admin role. Check if they are used and if they need this role or if they could use a role with fewer privileges.
Where possible, first bind users to a lower-privileged role and then remove the clusterrolebinding to the cluster-admin role :

```
kubectl delete clusterrolebinding [name]
```

Default Value:

By default a single `clusterrolebinding` called `cluster-admin` is provided with the `system:masters` group as its principal.

References:

1. https://kubernetes.io/docs/concepts/cluster-administration/
2. https://kubernetes.io/docs/reference/access-authn-authz/rbac/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.4 Restrict Administrator Privileges to Dedicated Administrator Accounts Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.	●	●	●
v7	4.3 Ensure the Use of Dedicated Administrative Accounts Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.	●	●	●

4.1.2 Minimize access to secrets (Automated)

Profile Applicability:

- Level 1

Description:

The Kubernetes API stores secrets, which may be service account tokens for the Kubernetes API or credentials used by workloads in the cluster. Access to these secrets should be restricted to the smallest possible group of users to reduce the risk of privilege escalation.

Rationale:

Inappropriate access to secrets stored within the Kubernetes cluster can allow for an attacker to gain additional access to the Kubernetes cluster or external resources whose credentials are stored as secrets.

Impact:

Care should be taken not to remove access to secrets to system components which require this for their operation

Audit:

Review the users who have `get`, `list` or `watch` access to `secrets` objects in the Kubernetes API.

Remediation:

Where possible, remove `get`, `list` and `watch` access to `secret` objects in the cluster.

Default Value:

```
CLUSTERROLEBINDING                                          SUBJECT
TYPE            SA-NAMESPACE
cluster-admin                                               system:masters
Group
system:controller:clusterrole-aggregation-controller  clusterrole-
aggregation-controller  ServiceAccount  kube-system
system:controller:expand-controller                         expand-controller
ServiceAccount  kube-system
system:controller:generic-garbage-collector                 generic-garbage-
collector           ServiceAccount  kube-system
system:controller:namespace-controller                      namespace-controller
ServiceAccount  kube-system
system:controller:persistent-volume-binder                  persistent-volume-
binder            ServiceAccount  kube-system
system:kube-controller-manager                              system:kube-controller-
manager      User
```

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.1 Establish and Maintain a Secure Configuration Process Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.	●	●	●
v7	5.2 Maintain Secure Images Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.		●	●

4.1.3 Minimize wildcard use in Roles and ClusterRoles (Automated)

Profile Applicability:

- Level 1

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.

Rationale:

The principle of least privilege recommends that users are provided only the access required for their role and nothing more. The use of wildcard rights grants is likely to provide excessive rights to the Kubernetes API.

Audit:

Retrieve the roles defined across each namespaces in the cluster and review for wildcards

```
kubectl get roles --all-namespaces -o yaml
```

Retrieve the cluster roles defined in the cluster and review for wildcards

```
kubectl get clusterroles -o yaml
```

Remediation:

Where possible replace any use of wildcards in clusterroles and roles with specific objects or actions.

References:

1. https://kubernetes.io/docs/reference/access-authn-authz/rbac/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.2 Use Unique Passwords Use unique passwords for all enterprise assets. Best practice implementation includes, at a minimum, an 8-character password for accounts using MFA and a 14-character password for accounts not using MFA.	●	●	●
v7	4.4 Use Unique Passwords Where multi-factor authentication is not supported (such as local administrator, root, or service accounts), accounts will use passwords that are unique to that system.		●	●

4.1.4 Ensure that default service accounts are not actively used (Automated)

Profile Applicability:

- Level 1

Description:

The `default` service account should not be used to ensure that rights granted to applications can be more easily audited and reviewed.

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.

Impact:

All workloads which require access to the Kubernetes API will require an explicit service account to be created.

Audit:

For each namespace in the cluster, review the rights assigned to the default service account and ensure that it has no roles or cluster roles bound to it apart from the defaults.
Additionally ensure that the `automountServiceAccountToken: false` setting is in place for each default service account.

Remediation:

Create explicit service accounts wherever a Kubernetes workload requires specific access to the Kubernetes API server.
Modify the configuration of each default service account to include this value

```
automountServiceAccountToken: false
```

Default Value:

By default the `default` service account allows for its service account token to be mounted in pods in its namespace.

References:

1. https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.3 Disable Dormant Accounts Delete or disable any dormant accounts after a period of 45 days of inactivity, where supported.	●	●	●
v7	4.3 Ensure the Use of Dedicated Administrative Accounts Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.	●	●	●
v7	5.2 Maintain Secure Images Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.		●	●
v7	16.9 Disable Dormant Accounts Automatically disable dormant accounts after a set period of inactivity.	●	●	●

4.1.5 Ensure that Service Account Tokens are only mounted where necessary (Automated)

Profile Applicability:

- Level 1

Description:

Service accounts tokens should not be mounted in pods except where the workload running in the pod explicitly needs to communicate with the API server

Rationale:

Mounting service account tokens inside pods can provide an avenue for privilege escalation attacks where an attacker is able to compromise a single pod in the cluster.

Avoiding mounting these tokens removes this attack avenue.

Impact:

Pods mounted without service account tokens will not be able to communicate with the API server, except where the resource is available to unauthenticated principals.

Audit:

Review pod and service account objects in the cluster and ensure that the option below is set, unless the resource explicitly requires this access.

```
automountServiceAccountToken: false
```

Remediation:

Modify the definition of pods and service accounts which do not need to mount service account tokens to disable it.

Default Value:

By default, all pods get a service account token mounted in them.

References:

1. https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.8 Uninstall or Disable Unnecessary Services on Enterprise Assets and Software Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.		●	●
v7	14.7 Enforce Access Control to Data through Automated Tools Use an automated tool, such as host-based Data Loss Prevention, to enforce access controls to data even when data is copied off a system.			●

4.1.6 Avoid use of system:masters group (Automated)

Profile Applicability:

- Level 1

Description:

The special group `system:masters` should not be used to grant permissions to any user or service account, except where strictly necessary (e.g. bootstrapping access prior to RBAC being fully available)

Rationale:

The `system:masters` group has unrestricted access to the Kubernetes API hard-coded into the API server source code. An authenticated user who is a member of this group cannot have their access reduced, even if all bindings and cluster role bindings which mention it, are removed.

When combined with client certificate authentication, use of this group can allow for irrevocable cluster-admin level credentials to exist for a cluster.

GKE includes the `CertificateSubjectRestriction` admission controller which rejects requests for the `system:masters` group.

`CertificateSubjectRestriction` "This admission controller observes creation of `CertificateSigningRequest` resources that have a spec.signerName of kubernetes.io/kube-apiserver-client. It rejects any request that specifies a 'group' (or 'organization attribute') of system:masters." https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#certificatesubjectrestriction

Impact:

Once the RBAC system is operational in a cluster `system:masters` should not be specifically required, as ordinary bindings from principals to the `cluster-admin` cluster role can be made where unrestricted access is required.

Audit:

Review a list of all credentials which have access to the cluster and ensure that the group `system:masters` is not used.

Remediation:

Remove the `system:masters` group from all users in the cluster.

Default Value:

By default some clusters will create a "break glass" client certificate which is a member of this group. Access to this client certificate should be carefully controlled and it should not be used for general cluster operations.

References:

1. https://github.com/kubernetes/kubernetes/blob/master/pkg/registry/rbac/escalation_check.go#L38

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.4 Restrict Administrator Privileges to Dedicated Administrator Accounts Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.	●	●	●
v7	4 Controlled Use of Administrative Privileges Controlled Use of Administrative Privileges			

4.1.7 Limit use of the Bind, Impersonate and Escalate permissions in the Kubernetes cluster (Manual)

Profile Applicability:

- Level 1

Description:

Cluster roles and roles with the impersonate, bind or escalate permissions should not be granted unless strictly required. Each of these permissions allow a particular subject to escalate their privileges beyond those explicitly granted by cluster administrators

Rationale:

The impersonate privilege allows a subject to impersonate other users gaining their rights to the cluster. The bind privilege allows the subject to add a binding to a cluster role or role which escalates their effective permissions in the cluster. The escalate privilege allows a subject to modify cluster roles to which they are bound, increasing their rights to that level.

Each of these permissions has the potential to allow for privilege escalation to cluster-admin level.

Impact:

There are some cases where these permissions are required for cluster service operation, and care should be taken before removing these permissions from system service accounts.

Audit:

Review the users who have access to cluster roles or roles which provide the impersonate, bind or escalate privileges.

Remediation:

Where possible, remove the impersonate, bind and escalate rights from subjects.

Default Value:

In a default kubeadm cluster, the `system:masters` group and `clusterrole-aggregation-controller` service account have access to the escalate privilege. The `system:masters` group also has access to bind and impersonate.

References:

1. https://www.impidio.com/blog/kubernetes-rbac-security-pitfalls
2. https://raesene.github.io/blog/2020/12/12/Escalating_Away/
3. https://raesene.github.io/blog/2021/01/16/Getting-Into-A-Bind-with-Kubernetes/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.4 Restrict Administrator Privileges to Dedicated Administrator Accounts Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.	●	●	●
v7	4 Controlled Use of Administrative Privileges Controlled Use of Administrative Privileges			

4.1.8 Avoid bindings to system:anonymous (Automated)

Profile Applicability:

- Level 1

Description:

Avoid ClusterRoleBindings nor RoleBindings with the user `system:anonymous`.

Rationale:

Kubernetes assigns user `system:anonymous` to API server requests that have no authentication information provided. Binding a role to user `system:anonymous` gives any unauthenticated user the permissions granted by that role and is strongly discouraged.

Impact:

Unauthenticated users will have privileges and permissions associated with roles associated with the configured bindings.

Care should be taken before removing any `clusterrolebindings` or `rolebindings` from the environment to ensure they were not required for operation of the cluster. Use a more specific and authenticated user for cluster operations.

Audit:

Both CusterRoleBindings and RoleBindings should be audited. Use the following command to confirm there are no `ClusterRoleBindings` to `system:anonymous`:

```
$ kubectl get clusterrolebindings -o json   | jq -r '["Name"], ["-----"],
(.items[] | select((.subjects | length) > 0) | select(any(.subjects[]; .name
== "system:anonymous")) | [.metadata.namespace, .metadata.name]) | @tsv'
```

There should be no `ClusterRoleBindings` listed. If any bindings exist, review their permissions with the following command and reassess their privilege.

```
$ kubectl get clusterrolebinding [CLUSTER_ROLE_BINDING_NAME] -o json \
    | jq ' .roleRef.name +" " + .roleRef.kind' \
    | sed -e 's/"//g' \
    | xargs -l bash -c 'kubectl get $1 $0 -o yaml'
```

Confirm that there are no `RoleBindings` including the `system:anonymous` user:

```
$ kubectl get rolebindings -A -o json \
  | jq -r '["Namespace", "Name"], ["---------", "-----"], (.items[] |
select((.subjects | length) > 0) | select(any(.subjects[]; .name ==
"system:anonymous")) | [.metadata.namespace, .metadata.name]) | @tsv'
```

There should be no `RoleBindings` listed.
If any bindings exist, review their permissions with the following command and reassess their privilege.

```
$ kubectl get rolebinding [ROLE_BINDING_NAME] --namespace
[ROLE_BINDING_NAMESPACE] -o json \
    | jq ' .roleRef.name +" " + .roleRef.kind' \
    | sed -e 's/"//g' \
    | xargs -l bash -c 'kubectl get $1 $0 -o yaml --namespace
[ROLE_BINDING_NAMESPACE]'
```

Remediation:

Identify all `clusterrolebindings` and `rolebindings` to the user system:anonymous.
Check if they are used and review the permissions associated with the binding using the commands in the Audit section above or refer to GKE documentation.
Strongly consider replacing unsafe bindings with an authenticated, user-defined group.
Where possible, bind to non-default, user-defined groups with least-privilege roles.
If there are any unsafe bindings to the user `system:anonymous`, proceed to delete them after consideration for cluster operations with only necessary, safer bindings.

```
kubectl delete clusterrolebinding
[CLUSTER_ROLE_BINDING_NAME]
kubectl delete rolebinding
[ROLE_BINDING_NAME]
--namespace
[ROLE_BINDING_NAMESPACE]
```

Default Value:

No `clusterrolebindings` nor `rolebindings` with user `system:anonymous`.

References:

1. https://kubernetes.io/docs/reference/access-authn-authz/rbac/#discovery-roles

4.1.9 Avoid non-default bindings to system:unauthenticated (Automated)

Profile Applicability:

- Level 1

Description:

Avoid non-default `ClusterRoleBindings` and `RoleBindings` with the group `system:unauthenticated`, except the `ClusterRoleBinding` `system:public-info-viewer`.

Rationale:

Kubernetes assigns the group `system:unauthenticated` to API server requests that have no authentication information provided. Binding a role to this group gives any unauthenticated user the permissions granted by that role and is strongly discouraged.

Impact:

Unauthenticated users will have privileges and permissions associated with roles associated with the configured bindings.

Care should be taken before removing any non-default `clusterrolebindings` or `rolebindings` from the environment to ensure they were not required for operation of the cluster. Leverage a more specific and authenticated user for cluster operations.

Audit:

Both `CusterRoleBindings` and `RoleBindings` should be audited. Use the following command to confirm there are no non-default `ClusterRoleBindings` to group `system:unauthenticated`:

```
$ kubectl get clusterrolebindings -o json    | jq -r '["Name"], ["-----"],
(.items[] | select((.subjects | length) > 0) | select(any(.subjects[]; .name
== "system:unauthenticated")) | [.metadata.namespace, .metadata.name]) |
@tsv'
```

Only the following default `ClusterRoleBinding` should be displayed:

```
Name
-----
        system:public-info-viewer
```

If any non-default bindings exist, review their permissions with the following command and reassess their privilege.

```
$ kubectl get clusterrolebinding [CLUSTER_ROLE_BINDING_NAME] -o json \
    | jq ' .roleRef.name +" " + .roleRef.kind' \
    | sed -e 's/"//g' \
    | xargs -l bash -c 'kubectl get $1 $0 -o yaml'
```

Confirm that there are no RoleBindings including the `system:unauthenticated` group:

```
$ kubectl get rolebindings -A -o json \
  | jq -r '["Namespace", "Name"], ["---------", "-----"], (.items[] |
select((.subjects | length) > 0) | select(any(.subjects[]; .name ==
"system:unauthenticated")) | [.metadata.namespace, .metadata.name]) | @tsv'
```

There should be no `RoleBindings` listed.
If any bindings exist, review their permissions with the following command and reassess their privilege.

```
$ kubectl get rolebinding [ROLE_BINDING_NAME] --namespace
[ROLE_BINDING_NAMESPACE] -o json \
    | jq ' .roleRef.name +" " + .roleRef.kind' \
    | sed -e 's/"//g' \
    | xargs -l bash -c 'kubectl get $1 $0 -o yaml --namespace
[ROLE_BINDING_NAMESPACE]'
```

Remediation:

Identify all non-default `clusterrolebindings` and `rolebindings` to the group `system:unauthenticated`. Check if they are used and review the permissions associated with the binding using the commands in the Audit section above or refer to GKE documentation.
Strongly consider replacing non-default, unsafe bindings with an authenticated, user-defined group. Where possible, bind to non-default, user-defined groups with least-privilege roles.
If there are any non-default, unsafe bindings to the group `system:unauthenticated`, proceed to delete them after consideration for cluster operations with only necessary, safer bindings.

```
kubectl delete clusterrolebinding
[CLUSTER_ROLE_BINDING_NAME]
kubectl delete rolebinding
[ROLE_BINDING_NAME]
--
namespace
[ROLE_BINDING_NAMESPACE]
```

Default Value:

`ClusterRoleBindings` with group s`ystem:unauthenticated`:

- `system:public-info-viewer`

No `RoleBindings` with the group `system:unauthenticated`.

References:

1. https://kubernetes.io/docs/reference/access-authn-authz/rbac/#discovery-roles

4.1.10 Avoid non-default bindings to system:authenticated (Automated)

Profile Applicability:

- Level 1

Description:

Avoid non-default `ClusterRoleBindings` and `RoleBindings` with the group `system:authenticated`, except the `ClusterRoleBindings` `system:basic-user`, `system:discovery`, and `system:public-info-viewer`.

Google's approach to authentication is to make authenticating to Google Cloud and GKE as simple and secure as possible without adding complex configuration steps. The group `system:authenticated` includes all users with a Google account, which includes all Gmail accounts. Consider your authorization controls with this extended group scope when granting permissions. Thus, group `system:authenticated` is not recommended for non-default use.

Rationale:

GKE assigns the group `system:authenticated` to API server requests made by any user who is signed in with a Google Account, including all Gmail accounts. In practice, this isn't meaningfully different from `system:unauthenticated` because anyone can create a Google Account.

Binding a role to the group `system:authenticated` gives any user with a Google Account, including all Gmail accounts, the permissions granted by that role and is strongly discouraged.

Impact:

Authenticated users in group `system:authenticated` should be treated similarly to users in `system:unauthenticated`, having privileges and permissions associated with roles associated with the configured bindings.

Care should be taken before removing any non-default `clusterrolebindings` or `rolebindings` from the environment to ensure they were not required for operation of the cluster. Leverage a more specific and authenticated user for cluster operations.

Audit:

Use the following command to confirm there are no non-default `ClusterRoleBindings` to `system:authenticated`:

```
$ kubectl get clusterrolebindings -o json    | jq -r '["Name"], ["-----"],
(.items[] | select((.subjects | length) > 0) | select(any(.subjects[]; .name
== "system:unauthenticated")) | [.metadata.namespace, .metadata.name]) |
@tsv'
```

Only the following default `ClusterRoleBindings` should be displayed:

```
Name
-----
        system:basic-user
        system:discovery
        system:public-info-viewer
```

If any non-default bindings exist, review their permissions with the following command and reassess their privilege.

```
$ kubectl get clusterrolebinding CLUSTER_ROLE_BINDING_NAME -o json \
    | jq ' .roleRef.name +" " + .roleRef.kind' \
    | sed -e 's/"//g' \
    | xargs -l bash -c 'kubectl get $1 $0 -o yaml'
```

Confirm that there are no `RoleBindings` including the `system:authenticated` group:

```
$ kubectl get rolebindings -A -o json \
  | jq -r '["Namespace", "Name"], ["---------", "-----"], (.items[] |
select((.subjects | length) > 0) | select(any(.subjects[]; .name ==
"system:unauthenticated")) | [.metadata.namespace, .metadata.name]) | @tsv'
```

There should be no `RoleBindings` listed.
If any bindings exist, review their permissions with the following command and reassess their privilege.

```
$ kubectl get rolebinding [ROLE_BINDING_NAME] --namespace
[ROLE_BINDING_NAMESPACE] -o json \
    | jq ' .roleRef.name +" " + .roleRef.kind' \
    | sed -e 's/"//g' \
    | xargs -l bash -c 'kubectl get $1 $0 -o yaml --namespace
[ROLE_BINDING_NAMESPACE]'
```

Remediation:

Identify all non-default `clusterrolebindings` and `rolebindings` to the group `system:authenticated`. Check if they are used and review the permissions associated with the binding using the commands in the Audit section above or refer to GKE documentation.
Strongly consider replacing non-default, unsafe bindings with an authenticated, user-defined group. Where possible, bind to non-default, user-defined groups with least-privilege roles.
If there are any non-default, unsafe bindings to the group `system:authenticated`, proceed to delete them after consideration for cluster operations with only necessary, safer bindings.

```
kubectl delete clusterrolebinding
[CLUSTER_ROLE_BINDING_NAME]
kubectl delete rolebinding
[ROLE_BINDING_NAME]
--namespace
[ROLE_BINDING_NAMESPACE]
```

Default Value:

`ClusterRoleBindings` with group `system:authenticated`:

- `system:basic-user`
- `system:discovery`

No `RoleBindings` with the group `system:authenticated`.

References:

1. https://kubernetes.io/docs/reference/access-authn-authz/rbac/#discovery-roles

4.2 Pod Security Standards

4.2.1 Ensure that the cluster enforces Pod Security Standard Baseline profile or stricter for all namespaces. (Manual)

Profile Applicability:

- Level 1

Description:

The Pod Security Standard Baseline profile defines a baseline for container security. You can enforce this by using the built-in Pod Security Admission controller.

Rationale:

Without an active mechanism to enforce the Pod Security Standard Baseline profile, it is not possible to limit the use of containers with access to underlying cluster nodes, via mechanisms like privileged containers, or the use of hostPath volume mounts.

Audit:

Run the following command to list the namespaces that don't have the the baseline policy enforced.

```
diff \
<(kubectl get namespace -l pod-security.kubernetes.io/enforce=baseline -o
jsonpath='{range .items[*]}{.metadata.name}{"\n"}') \
<(kubectl get namespace -o jsonpath='{range
.items[*]}{.metadata.name}{"\n"}')
```

Remediation:

Ensure that Pod Security Admission is in place for every namespace which contains user workloads.
Run the following command to enforce the Baseline profile in a namespace:
`kubectl label namespace <namespace-name> pod-security.kubernetes.io/enforce=baseline`

Default Value:

By default, Pod Security Admission is enabled but no policies are in place.

References:

1. https://kubernetes.io/docs/concepts/security/pod-security-admission
2. https://kubernetes.io/docs/concepts/security/pod-security-standards
3. https://cloud.google.com/kubernetes-engine/docs/concepts/about-security-posture-dashboard

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.7 Use Standard Hardening Configuration Templates for Application Infrastructure Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.		●	●
v7	5.1 Establish Secure Configurations Maintain documented, standard security configuration standards for all authorized operating systems and software.	●	●	●
v7	5.2 Maintain Secure Images Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.		●	●

4.3 Network Policies and CNI

4.3.1 Ensure that all Namespaces have Network Policies defined (Automated)

Profile Applicability:

- Level 2

Description:

Use network policies to isolate traffic in the cluster network.

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.

Impact:

Once network policies are in use within a given namespace, traffic not explicitly allowed by a network policy will be denied. As such it is important to ensure that, when introducing network policies, legitimate traffic is not blocked.

Audit:

Run the below command and review the `NetworkPolicy` objects created in the cluster.

```
kubectl get networkpolicy --all-namespaces

ensure that each namespace defined in the cluster has at least one Network
Policy.
```

Remediation:

Follow the documentation and create `NetworkPolicy` objects as needed.
See: https://cloud.google.com/kubernetes-engine/docs/how-to/network-policy#creating_a_network_policy for more information.

Default Value:

By default, network policies are not created.

References:

1. https://cloud.google.com/kubernetes-engine/docs/how-to/network-policy#creating_a_network_policy
2. https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/
3. https://cloud.google.com/kubernetes-engine/docs/concepts/network-overview

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	13.4 Perform Traffic Filtering Between Network Segments Perform traffic filtering between network segments, where appropriate.		●	●
v7	14.1 Segment the Network Based on Sensitivity Segment the network based on the label or classification level of the information stored on the servers, locate all sensitive information on separated Virtual Local Area Networks (VLANs).		●	●
v7	14.2 Enable Firewall Filtering Between VLANs Enable firewall filtering between VLANs to ensure that only authorized systems are able to communicate with other systems necessary to fulfill their specific responsibilities.		●	●

4.4 Secrets Management

4.4.1 Consider external secret storage (Manual)

Profile Applicability:

- Level 2

Description:

Consider the use of an external secrets storage and management system instead of using Kubernetes Secrets directly, if more complex secret management is required. Ensure the solution requires authentication to access secrets, has auditing of access to and use of secrets, and encrypts secrets. Some solutions also make it easier to rotate secrets.

Rationale:

Kubernetes supports secrets as first-class objects, but care needs to be taken to ensure that access to secrets is carefully limited. Using an external secrets provider can ease the management of access to secrets, especially where secrests are used across both Kubernetes and non-Kubernetes environments.

Impact:

None

Audit:

Review your secrets management implementation.

Remediation:

Refer to the secrets management options offered by the cloud service provider or a third-party secrets management solution.

Default Value:

By default, no external secret management is configured.

References:

1. https://kubernetes.io/docs/concepts/configuration/secret/
2. https://cloud.google.com/secret-manager/docs/overview

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3 Data Protection Develop processes and technical controls to identify, classify, securely handle, retain, and dispose of data.			
v7	13 Data Protection Data Protection			

4.5 Extensible Admission Control

4.5.1 Configure Image Provenance using ImagePolicyWebhook admission controller (Manual)

Profile Applicability:

- Level 2

Description:

Configure Image Provenance for the deployment.

Rationale:

Kubernetes supports plugging in provenance rules to accept or reject the images in deployments. Rules can be configured to ensure that only approved images are deployed in the cluster.

Impact:

Regular maintenance for the provenance configuration should be carried out, based on container image updates.

Audit:

Review the pod definitions in the cluster and verify that image provenance is configured as appropriate.

Remediation:

Follow the Kubernetes documentation and setup image provenance.

Default Value:

By default, image provenance is not set.

References:

1. https://kubernetes.io/docs/concepts/containers/images/
2. https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.6 Securely Manage Enterprise Assets and Software Securely manage enterprise assets and software. Example implementations include managing configuration through version-controlled-infrastructure-as-code and accessing administrative interfaces over secure network protocols, such as Secure Shell (SSH) and Hypertext Transfer Protocol Secure (HTTPS). Do not use insecure management protocols, such as Telnet (Teletype Network) and HTTP, unless operationally essential.	●	●	●
v7	18 Application Software Security Application Software Security			

4.6 General Policies

These policies relate to general cluster management topics, like namespace best practices and policies applied to pod objects in the cluster.

4.6.1 Create administrative boundaries between resources using namespaces (Manual)

Profile Applicability:

- Level 1

Description:

Use namespaces to isolate your Kubernetes objects.

Rationale:

Limiting the scope of user permissions can reduce the impact of mistakes or malicious activities. A Kubernetes namespace allows you to partition created resources into logically named groups. Resources created in one namespace can be hidden from other namespaces. By default, each resource created by a user in Kubernetes cluster runs in a default namespace, called `default`. You can create additional namespaces and attach resources and users to them. You can use Kubernetes Authorization plugins to create policies that segregate access to namespace resources between different users.

Impact:

You need to switch between namespaces for administration.

Audit:

Run the below command and review the namespaces created in the cluster.

```
kubectl get namespaces
```

Ensure that these namespaces are the ones you need and are adequately administered as per your requirements.

Remediation:

Follow the documentation and create namespaces for objects in your deployment as you need them.

Default Value:

By default, Kubernetes starts with two initial namespaces:

1. `default` - The default namespace for objects with no other namespace
2. `kube-system` - The namespace for objects created by the Kubernetes system
3. `kube-node-lease` - Namespace used for node heartbeats
4. `kube-public` - Namespace used for public information in a cluster

References:

1. https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/#viewing-namespaces
2. http://blog.kubernetes.io/2016/08/security-best-practices-kubernetes-deployment.html
3. https://github.com/kubernetes/enhancements/tree/master/keps/sig-node/589-efficient-node-heartbeats

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	13 Network Monitoring and Defense Operate processes and tooling to establish and maintain comprehensive network monitoring and defense against security threats across the enterprise's network infrastructure and user base.			
v7	12 Boundary Defense Boundary Defense			

4.6.2 Ensure that the seccomp profile is set to RuntimeDefault in the pod definitions (Automated)

Profile Applicability:

- Level 2

Description:

Enable `RuntimeDefault` seccomp profile in the pod definitions.

Rationale:

Seccomp (secure computing mode) is used to restrict the set of system calls applications can make, allowing cluster administrators greater control over the security of workloads running in the cluster. Kubernetes disables seccomp profiles by default for historical reasons. It should be enabled to ensure that the workloads have restricted actions available within the container.

Impact:

If the `RuntimeDefault` seccomp profile is too restrictive for you, you would have to create/manage your own `Localhost` seccomp profiles.

Audit:

Review the pod definitions output for all namespaces in the cluster with the command below.

```
kubectl get pods --all-namespaces -o json | jq -r '.items[] |
select(.metadata.annotations."seccomp.security.alpha.kubernetes.io/pod" ==
"runtime/default" or .spec.securityContext.seccompProfile.type ==
"RuntimeDefault") | {namespace: .metadata.namespace, name: .metadata.name,
seccompProfile: .spec.securityContext.seccompProfile.type}'
```

Remediation:

Use security context to enable the `RuntimeDefault` seccomp profile in your pod definitions. An example is as below:

```
{
  "namespace": "kube-system",
  "name": "metrics-server-v0.7.0-dbcc8ddf6-gz7d4",
  "seccompProfile": "RuntimeDefault"
}
```

Default Value:

By default, seccomp profile is set to `unconfined` which means that no seccomp profiles are enabled.

References:

1. https://kubernetes.io/docs/tutorials/security/seccomp/
2. https://cloud.google.com/kubernetes-engine/docs/concepts/seccomp-in-gke

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.7 Use Standard Hardening Configuration Templates for Application Infrastructure Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.		●	●
v7	5.2 Maintain Secure Images Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.		●	●

4.6.3 Apply Security Context to Pods and Containers (Manual)

Profile Applicability:

- Level 2

Description:

Apply Security Context to Pods and Containers

Rationale:

A security context defines the operating system security settings (uid, gid, capabilities, SELinux role, etc..) applied to a container. When designing containers and pods, make sure that the security context is configured for pods, containers, and volumes. A security context is a property defined in the deployment yaml. It controls the security parameters that will be assigned to the pod/container/volume. There are two levels of security context: pod level security context, and container level security context.

Impact:

If you incorrectly apply security contexts, there may be issues running the pods.

Audit:

Review the pod definitions in the cluster and verify that the security contexts have been defined as appropriate.

Remediation:

Follow the Kubernetes documentation and apply security contexts to your pods. For a suggested list of security contexts, you may refer to the CIS Google Container-Optimized OS Benchmark.

Default Value:

By default, no security contexts are automatically applied to pods.

References:

1. https://kubernetes.io/docs/concepts/workloads/pods/
2. https://kubernetes.io/docs/concepts/containers/
3. https://kubernetes.io/docs/tasks/configure-pod-container/security-context/
4. https://learn.cisecurity.org/benchmarks

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4 Secure Configuration of Enterprise Assets and Software Establish and maintain the secure configuration of enterprise assets (end-user devices, including portable and mobile; network devices; non-computing/IoT devices; and servers) and software (operating systems and applications).			
v7	5.1 Establish Secure Configurations Maintain documented, standard security configuration standards for all authorized operating systems and software.	●	●	●

4.6.4 The default namespace should not be used (Automated)

Profile Applicability:

- Level 2

Description:

Kubernetes provides a default namespace, where objects are placed if no namespace is specified for them. Placing objects in this namespace makes application of RBAC and other controls more difficult.

Rationale:

Resources in a Kubernetes cluster should be segregated by namespace, to allow for security controls to be applied at that level and to make it easier to manage resources.

Impact:

None

Audit:

Run this command to list objects in default namespace

```
kubectl get $(kubectl api-resources --verbs=list --namespaced=true -o name |
paste -sd, -) --ignore-not-found -n default
```

The only entries there should be system managed resources such as the `kubernetes` service
OR

```
kubectl get pods -n default
```

Returning `No resources found in default namespace.`

Remediation:

Ensure that namespaces are created to allow for appropriate segregation of Kubernetes resources and that all new resources are created in a specific namespace.

Default Value:

Unless a namespace is specific on object creation, the `default` namespace will be used

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	12.2 Establish and Maintain a Secure Network Architecture Establish and maintain a secure network architecture. A secure network architecture must address segmentation, least privilege, and availability, at a minimum.		●	●
v7	2.10 Physically or Logically Segregate High Risk Applications Physically or logically segregated systems should be used to isolate and run software that is required for business operations but incur higher risk for the organization.			●

5 Managed services

This section consists of security recommendations for the direct configuration of GKE AutoPilot managed service components, namely, Google Kubernetes Engine (GKE) AutoPilot mode. These recommendations are directly applicable for features which exist only as part of a managed service.

5.1 Image Registry and Image Scanning

This section contains recommendations relating to container image registries and securing images in those registries, such as Google Container Registry (GCR).

5.1.1 Ensure Image Vulnerability Scanning is enabled (Automated)

Profile Applicability:

- Level 2

Description:

Note: GCR is now deprecated, being superseded by Artifact Registry starting 15th May 2024. Runtime Vulnerability scanning is available via GKE Security Posture

Scan images stored in Google Container Registry (GCR) or Artifact Registry (AR) for vulnerabilities.

Rationale:

Vulnerabilities in software packages can be exploited by malicious users to obtain unauthorized access to local cloud resources. GCR Container Analysis API or Artifact Registry Container Scanning API allow images stored in GCR or AR respectively to be scanned for known vulnerabilities.

Impact:

None.

Audit:

For Images Hosted in GCR:

Using Google Cloud Console:

1. Go to GCR by visiting https://console.cloud.google.com/gcr
2. Select Settings and check if `Vulnerability scanning` is Enabled.

Using Command Line:

```
gcloud services list --enabled
```

Ensure that the `Container Registry API` and `Container Analysis API` are listed in the output.

For Images Hosted in AR:

Using Google Cloud Console:

1. Go to AR by visiting https://console.cloud.google.com/artifacts
2. Select Settings and check if `Vulnerability scanning` is Enabled.

Using Command Line:

```
gcloud services list --enabled
```

Ensure that `Container Scanning API` and `Artifact Registry API` are listed in the output.

Remediation:

For Images Hosted in GCR:

Using Google Cloud Console

1. Go to GCR by visiting: https://console.cloud.google.com/gcr
2. Select Settings and, under the Vulnerability Scanning heading, click the TURN ON button.

Using Command Line

```
gcloud services enable containeranalysis.googleapis.com
```

For Images Hosted in AR:

Using Google Cloud Console

1. Go to GCR by visiting: https://console.cloud.google.com/artifacts
2. Select Settings and, under the Vulnerability Scanning heading, click the ENABLE button.

Using Command Line

```
gcloud services enable containerscanning.googleapis.com
```

Default Value:

By default, GCR Container Analysis and AR Container Scanning are disabled.

References:

1. https://cloud.google.com/artifact-registry/docs/analysis
2. https://cloud.google.com/artifact-analysis/docs/os-overview
3. https://console.cloud.google.com/marketplace/product/google/containerregistry.googleapis.com
4. https://cloud.google.com/kubernetes-engine/docs/concepts/about-configuration-scanning
5. https://containersecurity.googleapis.com

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	7.6 Perform Automated Vulnerability Scans of Externally-Exposed Enterprise Assets Perform automated vulnerability scans of externally-exposed enterprise assets using a SCAP-compliant vulnerability scanning tool. Perform scans on a monthly, or more frequent, basis.		●	●
v7	3 Continuous Vulnerability Management Continuous Vulnerability Management			
v7	3.1 Run Automated Vulnerability Scanning Tools Utilize an up-to-date SCAP-compliant vulnerability scanning tool to automatically scan all systems on the network on a weekly or more frequent basis to identify all potential vulnerabilities on the organization's systems.		●	●
v7	3.2 Perform Authenticated Vulnerability Scanning Perform authenticated vulnerability scanning with agents running locally on each system or with remote scanners that are configured with elevated rights on the system being tested.		●	●

5.1.2 Minimize user access to Container Image repositories (Manual)

Profile Applicability:

- Level 2

Description:

Note: GCR is now deprecated, see the references for more details.

Restrict user access to GCR or AR, limiting interaction with build images to only authorized personnel and service accounts.

Rationale:

Weak access control to GCR or AR may allow malicious users to replace built images with vulnerable or back-doored containers.

Impact:

Care should be taken not to remove access to GCR or AR for accounts that require this for their operation. Any account granted the Storage Object Viewer role at the project level can view all objects stored in GCS for the project.

Audit:

For Images Hosted in AR:

1. Go to Artifacts Browser by visiting https://console.cloud.google.com/artifacts
2. From the list of artifacts select each repository with format `Docker`
3. Under the Permissions tab, review the roles for each member and ensure only authorized users have the Artifact Registry Administrator, Artifact Registry Reader, Artifact Registry Repository Administrator and Artifact Registry Writer roles.

Users may have permissions to use Service Accounts and thus Users could inherit privileges on the AR repositories. To check the accounts that could do this:

1. Go to IAM by visiting https://console.cloud.google.com/iam-admin/iam
2. Apply the filter `Role: Service Account User`.

Note that other privileged project level roles will have the ability to write and modify AR repositories. Consult the GCP CIS benchmark and IAM documentation for further reference.
Using Command Line:

```
gcloud artifacts repositories get-iam-policy <repository-name> --location
<repository-location>
```

The output of the command will return roles associated with the AR repository and which members have those roles.

For Images Hosted in GCR:

Using Google Cloud Console:
GCR bucket permissions

1. Go to Storage Browser by visiting https://console.cloud.google.com/storage/browser
2. From the list of storage buckets, select `artifacts.<project_id>.appspot.com` for the GCR bucket
3. Under the Permissions tab, review the roles for each member and ensure only authorized users have the Storage Admin, Storage Object Admin, Storage Object Creator, Storage Legacy Bucket Owner, Storage Legacy Bucket Writer and Storage Legacy Object Owner roles.

Users may have permissions to use Service Accounts and thus Users could inherit privileges on the GCR Bucket. To check the accounts that could do this:

1. Go to IAM by visiting https://console.cloud.google.com/iam-admin/iam
2. Apply the filter `Role: Service Account User`.

Note that other privileged project level roles will have the ability to write and modify objects and the GCR bucket. Consult the GCP CIS benchmark and IAM documentation for further reference.
Using Command Line:
To check GCR bucket specific permissions

```
gsutil iam get gs://artifacts.<project_id>.appspot.com
```

The output of the command will return roles associated with the GCR bucket and which members have those roles.
Additionally, run the following to identify users and service accounts that hold privileged roles at the project level, and thus inherit these privileges within the GCR bucket:

```
gcloud projects get-iam-policy <project_id> \
--flatten="bindings[].members" \
--format='table(bindings.members,bindings.role)' \
--filter="bindings.role:roles/storage.admin OR
bindings.role:roles/storage.objectAdmin OR
bindings.role:roles/storage.objectCreator OR
bindings.role:roles/storage.legacyBucketOwner OR
bindings.role:roles/storage.legacyBucketWriter OR
bindings.role:roles/storage.legacyObjectOwner"
```

The output from the command lists the service accounts that have create/modify permissions.
Users may have permissions to use Service Accounts and thus Users could inherit privileges on the GCR Bucket. To check the accounts that could do this:

```
gcloud projects get-iam-policy <project_id>  \
--flatten="bindings[].members" \
--format='table(bindings.members)' \
--filter="bindings.role:roles/iam.serviceAccountUser"
```

Note that other privileged project level roles will have the ability to write and modify objects and the GCR bucket. Consult the GCP CIS benchmark and IAM documentation for further reference.

Remediation:

For Images Hosted in AR:

Using Google Cloud Console:

1. Go to Artifacts Browser by visiting https://console.cloud.google.com/artifacts
2. From the list of artifacts select each repository with format `Docker`
3. Under the Permissions tab, modify the roles for each member and ensure only authorized users have the Artifact Registry Administrator, Artifact Registry Reader, Artifact Registry Repository Administrator and Artifact Registry Writer roles.

Using Command Line:

```
gcloud artifacts repositories set-iam-policy <repository-name> <path-to-
policy-file> --location <repository-location>
```

To learn how to configure policy files see: https://cloud.google.com/artifact-registry/docs/access-control#grant

For Images Hosted in GCR:

Using Google Cloud Console:
To modify roles granted at the GCR bucket level:

1. Go to Storage Browser by visiting: https://console.cloud.google.com/storage/browser.
2. From the list of storage buckets, select `artifacts.<project_id>.appspot.com` for the GCR bucket
3. Under the Permissions tab, modify permissions of the identified member via the drop-down role menu and change the Role to `Storage Object Viewer` for read-only access.

For a User or Service account with Project level permissions inherited by the GCR bucket, or the `Service Account User Role`:

1. Go to IAM by visiting: https://console.cloud.google.com/iam-admin/iam
2. Find the User or Service account to be modified and click on the corresponding pencil icon.
3. Remove the `create`/`modify` role (`Storage Admin` / `Storage Object Admin` / `Storage Object Creator` / `Service Account User`) on the user or service account.
4. If required add the `Storage Object Viewer` role - note with caution that this permits the account to view all objects stored in GCS for the project.

Using Command Line:
To change roles at the GCR bucket level:
Firstly, run the following if read permissions are required:

```
gsutil iam ch <type>:<email_address>:objectViewer
gs://artifacts.<project_id>.appspot.com
```

Then remove the excessively privileged role (`Storage Admin` / `Storage Object Admin` / `Storage Object Creator`) using:

```
gsutil iam ch -d <type>:<email_address>:<role>
gs://artifacts.<project_id>.appspot.com
```

where:

- `<type>` can be one of the following:
  - `user`, if the `<email_address>` is a Google account.
  - `serviceAccount`, if `<email_address>` specifies a Service account.
  - `<email_address>` can be one of the following:
    - a Google account (for example, `someone@example.com`).
    - a Cloud IAM service account.

To modify roles defined at the project level and subsequently inherited within the GCR bucket, or the Service Account User role, extract the IAM policy file, modify it accordingly and apply it using:

```
gcloud projects set-iam-policy <project_id> <policy_file>
```

Default Value:

By default, GCR is disabled and access controls are set during initialisation.

References:

1. https://cloud.google.com/container-registry/docs/
2. https://cloud.google.com/kubernetes-engine/docs/how-to/service-accounts
3. https://cloud.google.com/kubernetes-engine/docs/how-to/iam
4. https://cloud.google.com/artifact-registry/docs/access-control#grant

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 Configure Data Access Control Lists Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.	●	●	●
v7	14.6 Protect Information through Access Control Lists Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.	●	●	●

5.1.3 Minimize cluster access to read-only for Container Image repositories (Manual)

Profile Applicability:

- Level 2

Description:

Note: GCR is now deprecated, see the references for more details.

Configure the Cluster Service Account with Artifact Registry Viewer Role to only allow read-only access to AR repositories. Configure the Cluster Service Account with Storage Object Viewer Role to only allow read-only access to GCR.

Rationale:

The Cluster Service Account does not require administrative access to GCR or AR, only requiring pull access to containers to deploy onto GKE. Restricting permissions follows the principles of least privilege and prevents credentials from being abused beyond the required role.

Impact:

A separate dedicated service account may be required for use by build servers and other robot users pushing or managing container images.

Any account granted the Storage Object Viewer role at the project level can view all objects stored in GCS for the project.

Audit:

For Images Hosted in AR:

Using Google Cloud Console

1. Go to Artifacts Browser by visiting https://console.cloud.google.com/artifacts
2. From the list of repositories, for each repository with Format Docker
3. Under the Permissions tab, review the role for GKE Service account and ensure that only the Artifact Registry Viewer role is set.

Using Command Line:

```
gcloud artifacts repositories get-iam-policy <repository-name> --location
<repository-location>
```

The output of the command will return roles associated with the AR repository. If listed, ensure the GKE Service account is set to `"role": "roles/artifactregistry.reader"`.

For Images Hosted in GCR:

Using Google Cloud Console

1. Go to Storage Browser by visiting https://console.cloud.google.com/storage/browser
2. From the list of storage buckets, select `artifacts.<project_id>.appspot.com` for the GCR bucket
3. Under the Permissions tab, review the role for GKE Service account and ensure that only the Storage Object Viewer role is set.

Using Command Line
GCR bucket permissions

```
gsutil iam get gs://artifacts.<project_id>.appspot.com
```

The output of the command will return roles associated with the GCR bucket. If listed, ensure the GKE Service account is set to `"role": "roles/storage.objectViewer"`. If the GKE Service Account has project level permissions that are inherited within the bucket, ensure that these are not privileged:

```
gcloud projects get-iam-policy <project_id> \
--flatten="bindings[].members" \
--format='table(bindings.members,bindings.role)' \
--filter="bindings.role:roles/storage.admin OR
bindings.role:roles/storage.objectAdmin OR
bindings.role:roles/storage.objectCreator OR
bindings.role:roles/storage.legacyBucketOwner OR
bindings.role:roles/storage.legacyBucketWriter OR
bindings.role:roles/storage.legacyObjectOwner"
```

Your GKE Service Account should not be output when this command is run.

Remediation:

For Images Hosted in AR:

Using Google Cloud Console:

1. Go to Artifacts Browser by visiting https://console.cloud.google.com/artifacts
2. From the list of repositories, for each repository with Format Docker
3. Under the Permissions tab, modify the permissions for GKE Service account and ensure that only the Artifact Registry Viewer role is set.

Using Command Line:
Add artifactregistry.reader role

```
gcloud artifacts repositories add-iam-policy-binding <repository> \
--location=<repository-location> \
--member='serviceAccount:<email-address>' \
--role='roles/artifactregistry.reader'
```

Remove any roles other than `artifactregistry.reader`

```
gcloud artifacts repositories remove-iam-policy-binding <repository> \
--location <repository-location> \
--member='serviceAccount:<email-address>' \
--role='<role-name>'
```

For Images Hosted in GCR:

Using Google Cloud Console:
For an account explicitly granted access to the bucket:

1. Go to Storage Browser by visiting: https://console.cloud.google.com/storage/browser.
2. From the list of storage buckets, select `artifacts.<project_id>.appspot.com` for the GCR bucket.
3. Under the Permissions tab, modify permissions of the identified GKE Service Account via the drop-down role menu and change to the Role to `Storage Object Viewer` for read-only access.

For an account that inherits access to the bucket through Project level permissions:

1. Go to IAM console by visiting: https://console.cloud.google.com/iam-admin.
2. From the list of accounts, identify the required service account and select the corresponding pencil icon.
3. Remove the `Storage Admin` / `Storage Object Admin` / `Storage Object Creator` roles.
4. Add the `Storage Object Viewer` role - note with caution that this permits the account to view all objects stored in GCS for the project.
5. Click `SAVE`.

Using Command Line:
For an account explicitly granted to the bucket:
Firstly add read access to the Kubernetes Service Account:

```
gsutil iam ch <type>:<email_address>:objectViewer
gs://artifacts.<project_id>.appspot.com
```

where:

- `<type>` can be one of the following:
  - `user`, if the `<email_address>` is a Google account.
  - `serviceAccount`, if `<email_address>` specifies a Service account.

- `<email_address>` can be one of the following:
  - a Google account (for example, `someone@example.com`).
  - a Cloud IAM service account.

Then remove the excessively privileged role (`Storage Admin` / `Storage Object Admin` / `Storage Object Creator`) using:

```
gsutil iam ch -d <type>:<email_address>:<role>
gs://artifacts.<project_id>.appspot.com
```

For an account that inherits access to the GCR Bucket through Project level permissions, modify the Projects IAM policy file accordingly, then upload it using:

```
gcloud projects set-iam-policy <project_id> <policy_file>
```

Default Value:

The default permissions for the cluster Service account is dependent on the initial configuration and IAM policy.

References:

1. https://cloud.google.com/container-registry/docs/
2. https://cloud.google.com/kubernetes-engine/docs/how-to/service-accounts
3. https://cloud.google.com/kubernetes-engine/docs/how-to/iam

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 Configure Data Access Control Lists Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.	●	●	●
v7	3.2 Perform Authenticated Vulnerability Scanning Perform authenticated vulnerability scanning with agents running locally on each system or with remote scanners that are configured with elevated rights on the system being tested.		●	●

5.1.4 Ensure only trusted container images are used (Automated)

Profile Applicability:

- Level 2

Description:

Use Binary Authorization to allowlist (whitelist) only approved container registries.

Rationale:

Allowing unrestricted access to external container registries provides the opportunity for malicious or unapproved containers to be deployed into the cluster. Ensuring only trusted container images are used reduces this risk.

Also see recommendation 5.10.4.

Impact:

All container images to be deployed to the cluster must be hosted within an approved container image registry. If public registries are not on the allowlist, a process for bringing commonly used container images into an approved private registry and keeping them up to date will be required.

Audit:

Using Google Cloud Console:
Check that Binary Authorization is enabled for the GKE cluster:

1. Go to Kubernetes Engine by visiting: https://console.cloud.google.com/kubernetes/list
2. Click on the cluster and on the Details pane, ensure that Binary Authorization is set to 'Enabled'.

Then assess the contents of the policy:

1. Go to Binary Authorization by visiting: https://console.cloud.google.com/security/binary-authorization
2. Ensure the project default rule is not set to 'Allow all images' under Policy deployment rules.
3. Review the list of 'Images exempt from policy' for unauthorized container registries.

Using Command Line:
Check that Binary Authorization is enabled for the GKE cluster:

```
gcloud container clusters describe <cluster_name> --zone <compute_zone> --
format json | jq .binaryAuthorization
```

This will return the following if Binary Authorization is enabled:

```
{
  "enabled": true
}
```

Then assess the contents of the policy:

```
gcloud container binauthz policy export > current-policy.yaml
```

Ensure that the current policy is not configured to allow all images (`evaluationMode: ALWAYS_ALLOW`).
Review the list of `admissionWhitelistPatterns` for unauthorized container registries.

```
cat current-policy.yaml
admissionWhitelistPatterns:
...
defaultAdmissionRule:
  evaluationMode: ALWAYS_ALLOW
```

Remediation:

Using Google Cloud Console:

1. Go to Binary Authorization by visiting: https://console.cloud.google.com/security/binary-authorization
2. Enable Binary Authorization API (if disabled).
3. Go to Kubernetes Engine by visiting: https://console.cloud.google.com/kubernetes/list.
4. Select Kubernetes cluster for which Binary Authorization is disabled.
5. Within the `Details` pane, under the `Security` heading, click on the pencil icon called `Edit binary authorization`.
6. Ensure that `Enable Binary Authorization` is checked.
7. Click `SAVE CHANGES`.
8. Return to the Binary Authorization by visiting: https://console.cloud.google.com/security/binary-authorization.
9. Set an appropriate policy for the cluster and enter the approved container registries under Image paths.

Using Command Line:
Update the cluster to enable Binary Authorization:

```
gcloud container cluster update <cluster_name> --enable-binauthz
```

Create a Binary Authorization Policy using the Binary Authorization Policy Reference: https://cloud.google.com/binary-authorization/docs/policy-yaml-reference for guidance.
Import the policy file into Binary Authorization:

```
gcloud container binauthz policy import <yaml_policy>
```

Default Value:

By default, Binary Authorization is disabled along with container registry allowlisting.

References:

1. https://cloud.google.com/binary-authorization/docs/policy-yaml-reference
2. https://cloud.google.com/binary-authorization/docs/setting-up

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.5 Allowlist Authorized Software Use technical controls, such as application allowlisting, to ensure that only authorized software can execute or be accessed. Reassess bi-annually, or more frequently.		●	●
v7	5.2 Maintain Secure Images Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.		●	●
v7	5.3 Securely Store Master Images Store the master images and templates on securely configured servers, validated with integrity monitoring tools, to ensure that only authorized changes to the images are possible.		●	●

5.2 Identity and Access Management (IAM)

This section contains recommendations relating to using Cloud IAM with GKE.

5.2.1 Ensure GKE clusters are not running using the Compute Engine default service account (Automated)

Profile Applicability:

- Level 2

Description:

Create and use minimally privileged Service accounts to run GKE clusters instead of using the Compute Engine default Service account. Unnecessary permissions could be abused in the case of a node compromise.

Rationale:

A GCP service account (as distinct from a Kubernetes ServiceAccount) is an identity that an instance or an application can be used to run GCP API requests. This identity is used to identify virtual machine instances to other Google Cloud Platform services. By default, Kubernetes Engine nodes use the Compute Engine default service account. This account has broad access by default, as defined by access scopes, making it useful to a wide variety of applications on the VM, but it has more permissions than are required to run your Kubernetes Engine cluster.

A minimally privileged service account should be created and used to run the Kubernetes Engine cluster instead of using the Compute Engine default service account, and create separate service accounts for each Kubernetes Workload (See recommendation 5.2.2).

Kubernetes Engine requires, at a minimum, the node service account to have the `monitoring.viewer`, `monitoring.metricWriter`, and `logging.logWriter` roles. Additional roles may need to be added for the nodes to pull images from GCR.

Impact:

Instances are automatically granted the https://www.googleapis.com/auth/cloud-platform scope to allow full access to all Google Cloud APIs. This is so that the IAM permissions of the instance are completely determined by the IAM roles of the Service account. Thus if Kubernetes workloads were using cluster access scopes to perform actions using Google APIs, they may no longer be able to, if not permitted by the permissions of the Service account. To remediate, follow recommendation 5.2.2.

The Service account roles listed here are the minimum required to run the cluster. Additional roles may be required to pull from a private instance of Google Container Registry (GCR).

Audit:

Using Google Cloud Console:

1. Go to Kubernetes Engine by visiting https://console.cloud.google.com/kubernetes/list
2. Select the cluster under test and under Security ensure Service account is not set to default.

To check the permissions allocated to the service account are the minimum required for cluster operation:

1. Go to IAM by visiting https://console.cloud.google.com/iam-admin/iam
2. From the list of Service accounts, ensure each cluster Service account has only the following roles:

- Logs Writer
- Monitoring Metric Writer
- Monitoring Viewer

Using Command line:
To check which Service account is set for an existing cluster, run the following command:

```
gcloud container clusters describe $CLUSTER_NAME --zone $COMPUTE_ZONE --
format json | jq '.nodeConfig.serviceAccount'
```

The output of the above command will return default if default Service account is used for Project access.
To check that the permissions allocated to the service account are the minimum required for cluster operation:

```
gcloud projects get-iam-policy <project_id> \
  --flatten="bindings[].members" \
  --format='table(bindings.role)' \
  --filter="bindings.members:<service_account>"
```

Review the output to ensure that the service account only has the roles required to run the cluster:

- roles/logging.logWriter
- roles/monitoring.metricWriter
- roles/monitoring.viewer

Remediation:

Using Google Cloud Console:
To create a minimally privileged service account:

1. Go to Service Accounts by visiting: https://console.cloud.google.com/iam-admin/serviceaccounts.
2. Click on CREATE SERVICE ACCOUNT.
3. Enter Service Account Details.

4. Click CREATE AND CONTINUE.
5. Within Service Account permissions add the following roles:
    - Logs Writer.
    - Monitoring Metric Writer.
    - `Monitoring Viewer.
6. Click CONTINUE.
7. Grant users access to this service account and create keys as required.
8. Click DONE.

Note: A new cluster will need to be created specifying the minimally privileged service account, and workloads will need to be migrated to the new cluster and the old cluster deleted.
Using Command Line:
To create a minimally privileged service account:

```
gcloud iam service-accounts create <node_sa_name> --display-name "GKE Node
Service Account"
export NODE_SA_EMAIL=gcloud iam service-accounts list --format='value(email)'
--filter='displayName:GKE Node Service Account'
```

Grant the following roles to the service account:

```
export PROJECT_ID=gcloud config get-value project
gcloud projects add-iam-policy-binding <project_id> --member
serviceAccount:<node_sa_email> --role roles/monitoring.metricWriter
gcloud projects add-iam-policy-binding <project_id> --member
serviceAccount:<node_sa_email> --role roles/monitoring.viewer
gcloud projects add-iam-policy-binding <project_id> --member
serviceAccount:<node_sa_email> --role roles/logging.logWriter
```

Note: A new cluster will need to be created specifying the minimally privileged service account, and workloads will need to be migrated to the new cluster and the old cluster deleted.

Default Value:

By default, nodes use the Compute Engine default service account when you create a new cluster.

References:

1. https://cloud.google.com/compute/docs/access/service-accounts#compute_engine_default_service_account

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.7 Manage Default Accounts on Enterprise Assets and Software Manage default accounts on enterprise assets and software, such as root, administrator, and other pre-configured vendor accounts. Example implementations can include: disabling default accounts or making them unusable.	●	●	●
v7	4.3 Ensure the Use of Dedicated Administrative Accounts Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.	●	●	●

5.3 Cloud Key Management Service (Cloud KMS)

This section contains recommendations relating to using Cloud KMS with GKE.

5.3.1 Ensure Kubernetes Secrets are encrypted using keys managed in Cloud KMS (Automated)

Profile Applicability:

- Level 2

Description:

Encrypt Kubernetes secrets, stored in etcd, at the application-layer using a customer-managed key in Cloud KMS.

Rationale:

By default, GKE encrypts customer content stored at rest, including Secrets. GKE handles and manages this default encryption for you without any additional action on your part.

Application-layer Secrets Encryption provides an additional layer of security for sensitive data, such as user defined Secrets and Secrets required for the operation of the cluster, such as service account keys, which are all stored in etcd.

Using this functionality, you can use a key, that you manage in Cloud KMS, to encrypt data at the application layer. This protects against attackers in the event that they manage to gain access to etcd.

Impact:

To use the Cloud KMS CryptoKey to protect etcd in the cluster, the 'Kubernetes Engine Service Agent' Service account must hold the 'Cloud KMS CryptoKey Encrypter/Decrypter' role.

Audit:

Using Google Cloud Console:

1. Go to Kubernetes Engine by visiting https://console.cloud.google.com/kubernetes/list
2. From the list of clusters, click on each cluster to bring up the Details pane, and ensure Application-layer Secrets Encryption is set to 'Enabled'.

Using Command Line:

```
gcloud container clusters describe $CLUSTER_NAME --zone $COMPUTE_ZONE --
format json | jq '.databaseEncryption'
```

If configured correctly, the output from the command returns a response containing the following detail:

```
keyName=projects/<key_project_id>/locations/<location>/keyRings/<ring_name>/c
ryptoKeys/<key_name>]
state=ENCRYPTED

{
  "currentState": "CURRENT_STATE_ENCRYPTED",
  "keyName": "projects/<key_project_id>/locations/us-
central1/keyRings/<ring_name>/cryptoKeys/<key_name>",
  "state": "ENCRYPTED"
}
```

Remediation:

To enable Application-layer Secrets Encryption, several configuration items are required. These include:

- A key ring
- A key
- A GKE service account with `Cloud KMS CryptoKey Encrypter/Decrypter` role

Once these are created, Application-layer Secrets Encryption can be enabled on an existing or new cluster.
Using Google Cloud Console:
To create a key

1. Go to Cloud KMS by visiting https://console.cloud.google.com/security/kms.
2. Select `CREATE KEY RING`.
3. Enter a Key ring name and the region where the keys will be stored.
4. Click `CREATE`.
5. Enter a Key name and appropriate rotation period within the Create key pane.
6. Click `CREATE`.

To enable on a new cluster

1. Go to Kubernetes Engine by visiting: https://console.cloud.google.com/kubernetes/list.
2. Click `CREATE CLUSTER`, and choose the required cluster mode.
3. Within the `Security` heading, under `CLUSTER`, check `Encrypt secrets at the application layer` checkbox.
4. Select the kms key as the customer-managed key and, if prompted, grant permissions to the GKE Service account.
5. Click `CREATE`.

To enable on an existing cluster

1. Go to Kubernetes Engine by visiting: https://console.cloud.google.com/kubernetes/list.

2. Select the cluster to be updated.
3. Under the Details pane, within the Security heading, click on the pencil named Application-layer secrets encryption.
4. Enable `Encrypt secrets at the application layer` and choose a kms key.
5. Click `SAVE CHANGES`.

Using Command Line:
To create a key:
Create a key ring:

```
gcloud kms keyrings create <ring_name> --location <location> --project
<key_project_id>
```

Create a key:

```
gcloud kms keys create <key_name> --location <location> --keyring <ring_name>
--purpose encryption --project <key_project_id>
```

Grant the Kubernetes Engine Service Agent service account the `Cloud KMS CryptoKey Encrypter/Decrypter` role:

```
gcloud kms keys add-iam-policy-binding <key_name> --location <location> --
keyring <ring_name> --member serviceAccount:<service_account_name> --role
roles/cloudkms.cryptoKeyEncrypterDecrypter --project <key_project_id>
```

To create a new cluster with Application-layer Secrets Encryption:

```
gcloud container clusters create <cluster_name> --cluster-version=latest --
zone <zone> --database-encryption-key
projects/<key_project_id>/locations/<location>/keyRings/<ring_name>/cryptoKey
s/<key_name> --project <cluster_project_id>
```

To enable on an existing cluster:

```
gcloud container clusters update <cluster_name> --zone <zone> --database-
encryption-key
projects/<key_project_id>/locations/<location>/keyRings/<ring_name>/cryptoKey
s/<key_name> --project <cluster_project_id>
```

Default Value:

By default, Application-layer Secrets Encryption is disabled.

References:

1. https://cloud.google.com/kubernetes-engine/docs/how-to/encrypting-secrets

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.11 Encrypt Sensitive Data at Rest Encrypt sensitive data at rest on servers, applications, and databases containing sensitive data. Storage-layer encryption, also known as server-side encryption, meets the minimum requirement of this Safeguard. Additional encryption methods may include application-layer encryption, also known as client-side encryption, where access to the data storage device(s) does not permit access to the plain-text data.		●	●
v7	14.8 Encrypt Sensitive Information at Rest Encrypt all sensitive information at rest using a tool that requires a secondary authentication mechanism not integrated into the operating system, in order to access the information.			●

5.4 Cluster Networking

This section contains recommendations relating to network security configurations in GKE.

5.4.1 Enable VPC Flow Logs and Intranode Visibility (Automated)

Profile Applicability:

- Level 1

Description:

Enable VPC Flow Logs and Intranode Visibility to see pod-level traffic, even for traffic within a worker node.

Rationale:

Enabling Intranode Visibility makes intranode pod to pod traffic visible to the networking fabric. With this feature, VPC Flow Logs or other VPC features can be used for intranode traffic.

Impact:

Enabling it on existing cluster causes the cluster master and the cluster nodes to restart, which might cause disruption.

Audit:

Using Google Cloud Console:

1. Go to Kubernetes Engine by visiting: https://console.cloud.google.com/kubernetes/list
2. Select the desired cluster, and under the `Cluster` section, make sure that `Intranode visibility` is set to `Enabled`.

Using Command Line:
Run this command:

```
gcloud container clusters describe $CLUSTER_NAME --zone $COMPUTE_ZONE --
format json | jq '.networkConfig.enableIntraNodeVisibility'
```

The result should return `true` if Intranode Visibility is Enabled.

Remediation:

Enable Intranode Visibility:
Using Google Cloud Console:

1. Go to Kubernetes Engine by visiting: https://console.cloud.google.com/kubernetes/list.
2. Select Kubernetes clusters for which intranode visibility is disabled.
3. Within the `Details` pane, under the `Network` section, click on the pencil icon named `Edit intranode visibility`.
4. Check the box next to `Enable Intranode visibility`.

5. Click SAVE CHANGES.

Using Command Line:
To enable intranode visibility on an existing cluster, run the following command:

```
gcloud container clusters update <cluster_name> --enable-intra-node-visibility
```

Enable VPC Flow Logs:
Using Google Cloud Console:

1. Go to Kubernetes Engine by visiting: https://console.cloud.google.com/kubernetes/list.
2. Select Kubernetes clusters for which VPC Flow Logs are disabled.
3. Select Nodes tab.
4. Select Node Pool without VPC Flow Logs enabled.
5. Select an Instance Group within the node pool.
6. Select an Instance Group Member.
7. Select the Subnetwork under Network Interfaces.
8. Click on EDIT.
9. Set Flow logs to On.
10. Click SAVE.

Using Command Line:

1. Find the subnetwork name associated with the cluster.

```
gcloud container clusters describe <cluster_name> --region <cluster_region> --format json | jq '.subnetwork'
```

2. Update the subnetwork to enable VPC Flow Logs.

```
gcloud compute networks subnets update <subnet_name> --enable-flow-logs
```

Default Value:

By default, Intranode Visibility is disabled.

References:

1. https://cloud.google.com/kubernetes-engine/docs/how-to/intranode-visibility
2. https://cloud.google.com/vpc/docs/using-flow-logs

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.		●	●
v7	6.3 Enable Detailed Logging Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.		●	●

5.4.2 Ensure Control Plane Authorized Networks is Enabled (Automated)

Profile Applicability:

- Level 2

Description:

Enable Control Plane Authorized Networks to restrict access to the cluster's control plane to only an allowlist of authorized IPs.

Rationale:

Authorized networks are a way of specifying a restricted range of IP addresses that are permitted to access your cluster's control plane. Kubernetes Engine uses both Transport Layer Security (TLS) and authentication to provide secure access to your cluster's control plane from the public internet. This provides you the flexibility to administer your cluster from anywhere; however, you might want to further restrict access to a set of IP addresses that you control. You can set this restriction by specifying an authorized network.

Control Plane Authorized Networks blocks untrusted IP addresses. Google Cloud Platform IPs (such as traffic from Compute Engine VMs) can reach your master through HTTPS provided that they have the necessary Kubernetes credentials.

Restricting access to an authorized network can provide additional security benefits for your container cluster, including:

- Better protection from outsider attacks: Authorized networks provide an additional layer of security by limiting external, non-GCP access to a specific set of addresses you designate, such as those that originate from your premises. This helps protect access to your cluster in the case of a vulnerability in the cluster's authentication or authorization mechanism.
- Better protection from insider attacks: Authorized networks help protect your cluster from accidental leaks of master certificates from your company's premises. Leaked certificates used from outside GCP and outside the authorized IP ranges (for example, from addresses outside your company) are still denied access.

Impact:

When implementing Control Plane Authorized Networks, be careful to ensure all desired networks are on the allowlist to prevent inadvertently blocking external access to your cluster's control plane.

Audit:

Using Google Cloud Console:

1. Go to Kubernetes Engine by visiting: https://console.cloud.google.com/kubernetes/list
2. From the list of clusters, click on the cluster to open the Details page and make sure 'Control plane authorized networks' is set to 'Enabled'.

Using Command Line:
To check Master Authorized Networks status for an existing cluster, run the following command;

```
gcloud container clusters describe $CLUSTER_NAME --zone $COMPUTE_ZONE --
format json | jq '.masterAuthorizedNetworksConfig'
```

The output should include

```
{
  "enabled": true,
  "gcpPublicCidrsAccessEnabled": true
}
```

if Control Plane Authorized Networks is enabled.

Remediation:

Using Google Cloud Console:

1. Go to Kubernetes Engine by visiting https://console.cloud.google.com/kubernetes/list
2. Select Kubernetes clusters for which Control Plane Authorized Networks is disabled
3. Within the Details pane, under the Networking heading, click on the pencil icon named Edit control plane authorised networks.
4. Check the box next to Enable control plane authorised networks.
5. Click SAVE CHANGES.

Using Command Line:
To enable Control Plane Authorized Networks for an existing cluster, run the following sample command changing the IP range for fit your network:

```
gcloud container clusters update $CLUSTER_NAME --region $REGION --enable-
master-authorized-networks --master-authorized-networks 192.168.1.0/24
```

Along with this, you can list authorized networks using the `--master-authorized-networks` flag which contains a list of up to 20 external networks that are allowed to connect to your cluster's control plane through HTTPS. You provide these networks as a comma-separated list of addresses in CIDR notation (such as `90.90.100.0/24`).

Default Value:

By default, Control Plane Authorized Networks is disabled.

References:

1. https://cloud.google.com/kubernetes-engine/docs/how-to/authorized-networks

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 Configure Data Access Control Lists Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.	●	●	●
v7	14.6 Protect Information through Access Control Lists Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.	●	●	●

5.4.3 Ensure clusters are created with Private Endpoint Enabled and Public Access Disabled (Automated)

Profile Applicability:

- Level 2

Description:

Disable access to the Kubernetes API from outside the node network if it is not required.

Rationale:

In a private cluster, the master node has two endpoints, a private and public endpoint. The private endpoint is the internal IP address of the master, behind an internal load balancer in the master's VPC network. Nodes communicate with the master using the private endpoint. The public endpoint enables the Kubernetes API to be accessed from outside the master's VPC network.

Although Kubernetes API requires an authorized token to perform sensitive actions, a vulnerability could potentially expose the Kubernetes publically with unrestricted access. Additionally, an attacker may be able to identify the current cluster and Kubernetes API version and determine whether it is vulnerable to an attack. Unless required, disabling public endpoint will help prevent such threats, and require the attacker to be on the master's VPC network to perform any attack on the Kubernetes API.

Impact:

To enable a Private Endpoint, the cluster has to also be configured with private nodes, a private master IP range and IP aliasing enabled.

If the Private Endpoint flag `--enable-private-endpoint` is passed to the gcloud CLI, or the external IP address undefined in the Google Cloud Console during cluster creation, then all access from a public IP address is prohibited.

Audit:

Using Google Cloud Console:

1. Go to Kubernetes Engine by visiting https://console.cloud.google.com/kubernetes/list
2. Select the required cluster, and within the Details pane, make sure the 'Endpoint' does not have a public IP address.

Using Command Line:
Run this command:

```
gcloud container clusters describe $CLUSTER_NAME --zone $COMPUTE_ZONE --
format json | jq '.privateClusterConfig.enablePrivateEndpoint'
```

The output of the above command returns `true` if a Private Endpoint is enabled with Public Access disabled.

Remediation:

Once a cluster is created without enabling Private Endpoint only, it cannot be remediated. Rather, the cluster must be recreated.
Using Google Cloud Console:

1. Go to Kubernetes Engine by visiting https://console.cloud.google.com/kubernetes/list
2. Click CREATE CLUSTER, and choose CONFIGURE for the Standard mode cluster.
3. Configure the cluster as required then click Networking under CLUSTER in the navigation pane.
4. Under IPv4 network access, click the Private cluster radio button.
5. Uncheck the Access control plane using its external IP address checkbox.
6. In the Control plane IP range textbox, provide an IP range for the control plane.
7. Configure the other settings as required, and click CREATE.

Using Command Line:
Create a cluster with a Private Endpoint enabled and Public Access disabled by including the `--enable-private-endpoint` flag within the cluster create command:

```
gcloud container clusters create-auto <cluster_name> --location $LOCATION --
enable-private-endpoint
```

Setting this flag also requires the setting of `--enable-private-nodes` and `--enable-master-authorized-networks`.

Default Value:

By default, the Private Endpoint is disabled.

References:

1. https://cloud.google.com/kubernetes-engine/docs/how-to/private-clusters

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.4 Implement and Manage a Firewall on Servers Implement and manage a firewall on servers, where supported. Example implementations include a virtual firewall, operating system firewall, or a third-party firewall agent.	●	●	●

Controls Version	Control	IG 1	IG 2	IG 3
v7	12 Boundary Defense Boundary Defense			

5.4.4 Ensure clusters are created with Private Nodes (Automated)

Profile Applicability:

- Level 2

Description:

Private Nodes are nodes with no public IP addresses. Disable public IP addresses for cluster nodes, so that they only have private IP addresses.

Rationale:

Disabling public IP addresses on cluster nodes restricts access to only internal networks, forcing attackers to obtain local network access before attempting to compromise the underlying Kubernetes hosts.

Impact:

To enable Private Nodes, the cluster has to also be configured with a private master IP range and IP Aliasing enabled.

Private Nodes do not have outbound access to the public internet. If you want to provide outbound Internet access for your private nodes, you can use Cloud NAT or you can manage your own NAT gateway.

To access Google Cloud APIs and services from private nodes, Private Google Access needs to be set on Kubernetes Engine Cluster Subnets.

Audit:

Using Google Cloud Console:

1. Go to Kubernetes Engine by visiting: https://console.cloud.google.com/kubernetes/list.
2. Select the desired cluster, and within the Details pane, make sure `Private Cluster` is set to `Enabled`.

Using Command Line:
Run this command:

```
gcloud container clusters describe $CLUSTER_NAME --zone $COMPUTE_ZONE --
format json | jq '.privateClusterConfig.enablePrivateNodes'
```

The output of the above command returns `true` if Private Nodes is enabled.

Remediation:

Once a cluster is created without enabling Private Nodes, it cannot be remediated. Rather the cluster must be recreated.
Using Google Cloud Console:

1. Go to Kubernetes Engine by visiting: https://console.cloud.google.com/kubernetes/list.
2. Click CREATE CLUSTER.
3. Configure the cluster as required then click Networking under CLUSTER in the navigation pane.
4. Under IPv4 network access, click the Private cluster radio button.
5. Configure the other settings as required, and click CREATE.

Using Command Line:
To create a cluster with Private Nodes enabled, include the `--enable-private-nodes` flag within the cluster create command:

```
gcloud container clusters create <cluster_name> --location $LOCATION --
enable-private-nodes
```

Default Value:

By default, Private Nodes are disabled.

References:

1. https://cloud.google.com/kubernetes-engine/docs/how-to/private-clusters

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.4 Implement and Manage a Firewall on Servers Implement and manage a firewall on servers, where supported. Example implementations include a virtual firewall, operating system firewall, or a third-party firewall agent.	●	●	●
v7	12 Boundary Defense Boundary Defense			

5.4.5 Ensure use of Google-managed SSL Certificates (Automated)

Profile Applicability:

- Level 2

Description:

Encrypt traffic to HTTPS load balancers using Google-managed SSL certificates.

Rationale:

Encrypting traffic between users and the Kubernetes workload is fundamental to protecting data sent over the web.

Google-managed SSL Certificates are provisioned, renewed, and managed for domain names. This is only available for HTTPS load balancers created using Ingress Resources, and not TCP/UDP load balancers created using Service of `type:LoadBalancer`.

Impact:

Google-managed SSL Certificates are less flexible than certificates that are self obtained and managed. Managed certificates support a single, non-wildcard domain. Self-managed certificates can support wildcards and multiple subject alternative names (SANs).

Audit:

Using Command Line:
Identify if there are any workloads exposed publicly using Services of `type:LoadBalancer`:

```
kubectl get svc -A -o json | jq '.items[] |
select(.spec.type=="LoadBalancer")'
```

Consider using ingresses instead of these services in order to use Google managed SSL certificates.
For the ingresses within the cluster, run the following command:

```
kubectl get ingress -A -o json | jq .items[] | jq '{name: .metadata.name,
annotations: .metadata.annotations, namespace: .metadata.namespace, status:
.status}'
```

The above command should return the name of the ingress, namespace, annotations and status. Check that the following annotation is present to ensure managed certificates are referenced.

```
"annotations": {
    ...
    "networking.gke.io/managed-certificates": "<example_certificate>"
  },
```

For completeness, run the following command to ensure that the managed certificate resource exists:

```
kubectl get managedcertificates -A
```

The above command returns a list of managed certificates for which `<example_certificate>` should exist within the same namespace as the ingress.

Remediation:

If services of `type:LoadBalancer` are discovered, consider replacing the Service with an Ingress.
To configure the Ingress and use Google-managed SSL certificates, follow the instructions as listed at: https://cloud.google.com/kubernetes-engine/docs/how-to/managed-certs.

Default Value:

By default, Google-managed SSL Certificates are not created when an Ingress resource is defined.

References:

1. https://cloud.google.com/kubernetes-engine/docs/how-to/managed-certs
2. https://cloud.google.com/kubernetes-engine/docs/concepts/ingress

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).		●	●
v7	14.4 Encrypt All Sensitive Information in Transit Encrypt all sensitive information in transit.		●	●

5.5 Authentication and Authorization

This section contains recommendations relating to authentication and authorization in GKE.

5.5.1 Manage Kubernetes RBAC users with Google Groups for GKE (Manual)

Profile Applicability:

- Level 2

Description:

Cluster Administrators should leverage G Suite Groups and Cloud IAM to assign Kubernetes user roles to a collection of users, instead of to individual emails using only Cloud IAM.

Rationale:

On- and off-boarding users is often difficult to automate and prone to error. Using a single source of truth for user permissions via G Suite Groups reduces the number of locations that an individual must be off-boarded from, and prevents users gaining unique permissions sets that increase the cost of audit.

Impact:

When migrating to using security groups, an audit of `RoleBindings` and `ClusterRoleBindings` is required to ensure all users of the cluster are managed using the new groups and not individually.

When managing `RoleBindings` and `ClusterRoleBindings`, be wary of inadvertently removing bindings required by service accounts.

Audit:

Using G Suite Admin Console and Google Cloud Console

1. Navigate to manage G Suite Groups in the Google Admin console at: https://admin.google.com/dashboard
2. Ensure there is a group named `gke-security-groups@[yourdomain.com]`. The group must be named exactly `gke-security-groups`.
3. Ensure only further groups (not individual users) are included in the `gke-security-groups` group as members.
4. Go to the Kubernetes Engine by visiting: https://console.cloud.google.com/kubernetes/list.
5. From the list of clusters, click on the desired cluster. In the `Details` pane, make sure `Google Groups for RBAC` is set to `Enabled`.

Remediation:

Follow the G Suite Groups instructions at: https://cloud.google.com/kubernetes-engine/docs/how-to/role-based-access-control#google-groups-for-gke.
Then, create a cluster with:

```
gcloud container clusters create <cluster_name> --security-group
<security_group_name>
```

Finally create `Roles`, `ClusterRoles`, `RoleBindings`, and `ClusterRoleBindings` that reference the G Suite Groups.

Default Value:

Google Groups for GKE is disabled by default.

References:

1. https://cloud.google.com/kubernetes-engine/docs/how-to/google-groups-rbac
2. https://cloud.google.com/kubernetes-engine/docs/how-to/role-based-access-control

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	6.8 Define and Maintain Role-Based Access Control Define and maintain role-based access control, through determining and documenting the access rights necessary for each role within the enterprise to successfully carry out its assigned duties. Perform access control reviews of enterprise assets to validate that all privileges are authorized, on a recurring schedule at a minimum annually, or more frequently.			●
v7	16.2 Configure Centralized Point of Authentication Configure access for all accounts through as few centralized points of authentication as possible, including network, security, and cloud systems.		●	●

5.6 Storage

This section contains recommendations relating to security-related configurations for storage in GKE.

5.6.1 Enable Customer-Managed Encryption Keys (CMEK) for GKE Persistent Disks (PD) (Manual)

Profile Applicability:

- Level 2

Description:

Use Customer-Managed Encryption Keys (CMEK) to encrypt dynamically-provisioned attached Google Compute Engine Persistent Disks (PDs) using keys managed within Cloud Key Management Service (Cloud KMS).

Rationale:

GCE persistent disks are encrypted at rest by default using envelope encryption with keys managed by Google. For additional protection, users can manage the Key Encryption Keys using Cloud KMS.

Impact:

Encryption of dynamically-provisioned attached disks requires the use of the self-provisioned Compute Engine Persistent Disk CSI Driver v0.5.1 or higher.

If CMEK is being configured with a regional cluster, the cluster must run GKE 1.14 or higher.

Audit:

Using Google Cloud Console:

1. Go to Compute Engine Disks by visiting: https://console.cloud.google.com/compute/disks
2. Select each disk used by the cluster, and ensure the `Encryption Type` is listed as `Customer Managed`.

Using Command Line:
Identify the Persistent Volumes Used by the cluster:

```
kubectl get pv -o json | jq '.items[].metadata.name'
```

For each volume used, check that it is encrypted using a customer managed key by running the following command:

```
gcloud compute disks describe <pv_name> --zone <compute_zone> --format json |
jq '.diskEncryptionKey.kmsKeyName'
```

This returns null (`{ }`) if a customer-managed encryption key is not used to encrypt the disk.

Remediation:

This cannot be remediated by updating an existing cluster. The node pool must either be recreated or a new cluster created.
Using Google Cloud Console:
This is not possible using Google Cloud Console.
Using Command Line:
Follow the instructions detailed at: https://cloud.google.com/kubernetes-engine/docs/how-to/using-cmek.

Default Value:

Persistent disks are encrypted at rest by default, but are not encrypted using Customer-Managed Encryption Keys by default. By default, the Compute Engine Persistent Disk CSI Driver is not provisioned within the cluster.

References:

1. https://cloud.google.com/kubernetes-engine/docs/how-to/using-cmek
2. https://cloud.google.com/compute/docs/disks/customer-managed-encryption
3. https://cloud.google.com/security/encryption-at-rest/default-encryption/
4. https://cloud.google.com/kubernetes-engine/docs/concepts/persistent-volumes
5. https://cloud.google.com/sdk/gcloud/reference/container/node-pools/create

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.11 Encrypt Sensitive Data at Rest Encrypt sensitive data at rest on servers, applications, and databases containing sensitive data. Storage-layer encryption, also known as server-side encryption, meets the minimum requirement of this Safeguard. Additional encryption methods may include application-layer encryption, also known as client-side encryption, where access to the data storage device(s) does not permit access to the plain-text data.		●	●
v7	14.8 Encrypt Sensitive Information at Rest Encrypt all sensitive information at rest using a tool that requires a secondary authentication mechanism not integrated into the operating system, in order to access the information.			●

5.7 Other Cluster Configurations

This section contains recommendations relating to any remaining security-related cluster configurations in GKE.

5.7.1 Enable Security Posture (Manual)

Profile Applicability:

- Level 2

Description:

Rationale:

The security posture dashboard provides insights about your workload security posture at the runtime phase of the software delivery life-cycle.

Impact:

GKE security posture configuration auditing checks your workloads against a set of defined best practices. Each configuration check has its own impact or risk. Learn more about the checks: https://cloud.google.com/kubernetes-engine/docs/concepts/about-configuration-scanning

Example: The host namespace check identifies pods that share host namespaces. Pods that share host namespaces allow Pod processes to communicate with host processes and gather host information, which could lead to a container escape

Audit:

Check the SecurityPostureConfig on your cluster:
gcloud container clusters --location describe
securityPostureConfig: mode: BASIC

Remediation:

Enable security posture via the UI, gCloud or API. https://cloud.google.com/kubernetes-engine/docs/how-to/protect-workload-configuration

Default Value:

GKE security posture has multiple features. Not all are on by default. Configuration auditing is enabled by default for new standard and autopilot clusters.

securityPostureConfig: mode: BASIC

References:

1. https://cloud.google.com/kubernetes-engine/docs/concepts/about-security-posture-dashboard

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.4 Utilize Automated Software Inventory Tools Utilize software inventory tools, when possible, throughout the enterprise to automate the discovery and documentation of installed software.		●	●
v7	5.5 Implement Automated Configuration Monitoring Systems Utilize a Security Content Automation Protocol (SCAP) compliant configuration monitoring system to verify all security configuration elements, catalog approved exceptions, and alert when unauthorized changes occur.		●	●

Appendix: Summary Table

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
1	**Control Plane Components**		
2	**Control Plane Configuration**		
3	**Worker Nodes**		
4	**Policies**		
4.1	**RBAC and Service Accounts**		
4.1.1	Ensure that the cluster-admin role is only used where required (Automated)	☐	☐
4.1.2	Minimize access to secrets (Automated)	☐	☐
4.1.3	Minimize wildcard use in Roles and ClusterRoles (Automated)	☐	☐
4.1.4	Ensure that default service accounts are not actively used (Automated)	☐	☐
4.1.5	Ensure that Service Account Tokens are only mounted where necessary (Automated)	☐	☐
4.1.6	Avoid use of system:masters group (Automated)	☐	☐
4.1.7	Limit use of the Bind, Impersonate and Escalate permissions in the Kubernetes cluster (Manual)	☐	☐
4.1.8	Avoid bindings to system:anonymous (Automated)	☐	☐
4.1.9	Avoid non-default bindings to system:unauthenticated (Automated)	☐	☐
4.1.10	Avoid non-default bindings to system:authenticated (Automated)	☐	☐
4.2	**Pod Security Standards**		

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
4.2.1	Ensure that the cluster enforces Pod Security Standard Baseline profile or stricter for all namespaces. (Manual)	☐	☐
4.3	**Network Policies and CNI**		
4.3.1	Ensure that all Namespaces have Network Policies defined (Automated)	☐	☐
4.4	**Secrets Management**		
4.4.1	Consider external secret storage (Manual)	☐	☐
4.5	**Extensible Admission Control**		
4.5.1	Configure Image Provenance using ImagePolicyWebhook admission controller (Manual)	☐	☐
4.6	**General Policies**		
4.6.1	Create administrative boundaries between resources using namespaces (Manual)	☐	☐
4.6.2	Ensure that the seccomp profile is set to RuntimeDefault in the pod definitions (Automated)	☐	☐
4.6.3	Apply Security Context to Pods and Containers (Manual)	☐	☐
4.6.4	The default namespace should not be used (Automated)	☐	☐
5	**Managed services**		
5.1	**Image Registry and Image Scanning**		
5.1.1	Ensure Image Vulnerability Scanning is enabled (Automated)	☐	☐
5.1.2	Minimize user access to Container Image repositories (Manual)	☐	☐
5.1.3	Minimize cluster access to read-only for Container Image repositories (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
5.1.4	Ensure only trusted container images are used (Automated)	☐	☐
5.2	**Identity and Access Management (IAM)**		
5.2.1	Ensure GKE clusters are not running using the Compute Engine default service account (Automated)	☐	☐
5.3	**Cloud Key Management Service (Cloud KMS)**		
5.3.1	Ensure Kubernetes Secrets are encrypted using keys managed in Cloud KMS (Automated)	☐	☐
5.4	**Cluster Networking**		
5.4.1	Enable VPC Flow Logs and Intranode Visibility (Automated)	☐	☐
5.4.2	Ensure Control Plane Authorized Networks is Enabled (Automated)	☐	☐
5.4.3	Ensure clusters are created with Private Endpoint Enabled and Public Access Disabled (Automated)	☐	☐
5.4.4	Ensure clusters are created with Private Nodes (Automated)	☐	☐
5.4.5	Ensure use of Google-managed SSL Certificates (Automated)	☐	☐
5.5	**Authentication and Authorization**		
5.5.1	Manage Kubernetes RBAC users with Google Groups for GKE (Manual)	☐	☐
5.6	**Storage**		
5.6.1	Enable Customer-Managed Encryption Keys (CMEK) for GKE Persistent Disks (PD) (Manual)	☐	☐
5.7	**Other Cluster Configurations**		

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
5.7.1	Enable Security Posture (Manual)	☐	☐

Appendix: CIS Controls v7 IG 1 Mapped Recommendations

Recommendation		Set Correctly	
		Yes	No
4.1.1	Ensure that the cluster-admin role is only used where required	☐	☐
4.1.4	Ensure that default service accounts are not actively used	☐	☐
4.2.1	Ensure that the cluster enforces Pod Security Standard Baseline profile or stricter for all namespaces.	☐	☐
4.6.3	Apply Security Context to Pods and Containers	☐	☐
5.1.2	Minimize user access to Container Image repositories	☐	☐
5.2.1	Ensure GKE clusters are not running using the Compute Engine default service account	☐	☐
5.4.2	Ensure Control Plane Authorized Networks is Enabled	☐	☐

Appendix: CIS Controls v7 IG 2 Mapped Recommendations

Recommendation		Set Correctly	
		Yes	No
4.1.1	Ensure that the cluster-admin role is only used where required	☐	☐
4.1.2	Minimize access to secrets	☐	☐
4.1.3	Minimize wildcard use in Roles and ClusterRoles	☐	☐
4.1.4	Ensure that default service accounts are not actively used	☐	☐
4.2.1	Ensure that the cluster enforces Pod Security Standard Baseline profile or stricter for all namespaces.	☐	☐
4.3.1	Ensure that all Namespaces have Network Policies defined	☐	☐
4.6.2	Ensure that the seccomp profile is set to RuntimeDefault in the pod definitions	☐	☐
4.6.3	Apply Security Context to Pods and Containers	☐	☐
5.1.1	Ensure Image Vulnerability Scanning is enabled	☐	☐
5.1.2	Minimize user access to Container Image repositories	☐	☐
5.1.3	Minimize cluster access to read-only for Container Image repositories	☐	☐
5.1.4	Ensure only trusted container images are used	☐	☐
5.2.1	Ensure GKE clusters are not running using the Compute Engine default service account	☐	☐
5.4.1	Enable VPC Flow Logs and Intranode Visibility	☐	☐
5.4.2	Ensure Control Plane Authorized Networks is Enabled	☐	☐
5.4.5	Ensure use of Google-managed SSL Certificates	☐	☐
5.5.1	Manage Kubernetes RBAC users with Google Groups for GKE	☐	☐
5.7.1	Enable Security Posture	☐	☐

Appendix: CIS Controls v7 IG 3 Mapped Recommendations

Recommendation		Set Correctly	
		Yes	No
4.1.1	Ensure that the cluster-admin role is only used where required	☐	☐
4.1.2	Minimize access to secrets	☐	☐
4.1.3	Minimize wildcard use in Roles and ClusterRoles	☐	☐
4.1.4	Ensure that default service accounts are not actively used	☐	☐
4.1.5	Ensure that Service Account Tokens are only mounted where necessary	☐	☐
4.2.1	Ensure that the cluster enforces Pod Security Standard Baseline profile or stricter for all namespaces.	☐	☐
4.3.1	Ensure that all Namespaces have Network Policies defined	☐	☐
4.6.2	Ensure that the seccomp profile is set to RuntimeDefault in the pod definitions	☐	☐
4.6.3	Apply Security Context to Pods and Containers	☐	☐
4.6.4	The default namespace should not be used	☐	☐
5.1.1	Ensure Image Vulnerability Scanning is enabled	☐	☐
5.1.2	Minimize user access to Container Image repositories	☐	☐
5.1.3	Minimize cluster access to read-only for Container Image repositories	☐	☐
5.1.4	Ensure only trusted container images are used	☐	☐
5.2.1	Ensure GKE clusters are not running using the Compute Engine default service account	☐	☐
5.3.1	Ensure Kubernetes Secrets are encrypted using keys managed in Cloud KMS	☐	☐
5.4.1	Enable VPC Flow Logs and Intranode Visibility	☐	☐
5.4.2	Ensure Control Plane Authorized Networks is Enabled	☐	☐
5.4.5	Ensure use of Google-managed SSL Certificates	☐	☐
5.5.1	Manage Kubernetes RBAC users with Google Groups for GKE	☐	☐

Recommendation		Set Correctly	
		Yes	No
5.6.1	Enable Customer-Managed Encryption Keys (CMEK) for GKE Persistent Disks (PD)	☐	☐
5.7.1	Enable Security Posture	☐	☐

Appendix: CIS Controls v7 Unmapped Recommendations

Recommendation		Set Correctly	
		Yes	No
4.1.8	Avoid bindings to system:anonymous	☐	☐
4.1.9	Avoid non-default bindings to system:unauthenticated	☐	☐
4.1.10	Avoid non-default bindings to system:authenticated	☐	☐

Appendix: CIS Controls v8 IG 1 Mapped Recommendations

Recommendation		Set Correctly	
		Yes	No
4.1.1	Ensure that the cluster-admin role is only used where required	☐	☐
4.1.2	Minimize access to secrets	☐	☐
4.1.3	Minimize wildcard use in Roles and ClusterRoles	☐	☐
4.1.4	Ensure that default service accounts are not actively used	☐	☐
4.1.6	Avoid use of system:masters group	☐	☐
4.1.7	Limit use of the Bind, Impersonate and Escalate permissions in the Kubernetes cluster	☐	☐
4.5.1	Configure Image Provenance using ImagePolicyWebhook admission controller	☐	☐
5.1.2	Minimize user access to Container Image repositories	☐	☐
5.1.3	Minimize cluster access to read-only for Container Image repositories	☐	☐
5.2.1	Ensure GKE clusters are not running using the Compute Engine default service account	☐	☐
5.4.2	Ensure Control Plane Authorized Networks is Enabled	☐	☐
5.4.3	Ensure clusters are created with Private Endpoint Enabled and Public Access Disabled	☐	☐
5.4.4	Ensure clusters are created with Private Nodes	☐	☐

Appendix: CIS Controls v8 IG 2 Mapped Recommendations

Recommendation		Set Correctly	
		Yes	No
4.1.1	Ensure that the cluster-admin role is only used where required	☐	☐
4.1.2	Minimize access to secrets	☐	☐
4.1.3	Minimize wildcard use in Roles and ClusterRoles	☐	☐
4.1.4	Ensure that default service accounts are not actively used	☐	☐
4.1.5	Ensure that Service Account Tokens are only mounted where necessary	☐	☐
4.1.6	Avoid use of system:masters group	☐	☐
4.1.7	Limit use of the Bind, Impersonate and Escalate permissions in the Kubernetes cluster	☐	☐
4.2.1	Ensure that the cluster enforces Pod Security Standard Baseline profile or stricter for all namespaces.	☐	☐
4.3.1	Ensure that all Namespaces have Network Policies defined	☐	☐
4.5.1	Configure Image Provenance using ImagePolicyWebhook admission controller	☐	☐
4.6.2	Ensure that the seccomp profile is set to RuntimeDefault in the pod definitions	☐	☐
4.6.4	The default namespace should not be used	☐	☐
5.1.1	Ensure Image Vulnerability Scanning is enabled	☐	☐
5.1.2	Minimize user access to Container Image repositories	☐	☐
5.1.3	Minimize cluster access to read-only for Container Image repositories	☐	☐
5.1.4	Ensure only trusted container images are used	☐	☐
5.2.1	Ensure GKE clusters are not running using the Compute Engine default service account	☐	☐
5.3.1	Ensure Kubernetes Secrets are encrypted using keys managed in Cloud KMS	☐	☐
5.4.1	Enable VPC Flow Logs and Intranode Visibility	☐	☐

Recommendation		Set Correctly	
		Yes	No
5.4.2	Ensure Control Plane Authorized Networks is Enabled	☐	☐
5.4.3	Ensure clusters are created with Private Endpoint Enabled and Public Access Disabled	☐	☐
5.4.4	Ensure clusters are created with Private Nodes	☐	☐
5.4.5	Ensure use of Google-managed SSL Certificates	☐	☐
5.6.1	Enable Customer-Managed Encryption Keys (CMEK) for GKE Persistent Disks (PD)	☐	☐
5.7.1	Enable Security Posture	☐	☐

Appendix: CIS Controls v8 IG 3 Mapped Recommendations

Recommendation		Set Correctly	
		Yes	No
4.1.1	Ensure that the cluster-admin role is only used where required	☐	☐
4.1.2	Minimize access to secrets	☐	☐
4.1.3	Minimize wildcard use in Roles and ClusterRoles	☐	☐
4.1.4	Ensure that default service accounts are not actively used	☐	☐
4.1.5	Ensure that Service Account Tokens are only mounted where necessary	☐	☐
4.1.6	Avoid use of system:masters group	☐	☐
4.1.7	Limit use of the Bind, Impersonate and Escalate permissions in the Kubernetes cluster	☐	☐
4.2.1	Ensure that the cluster enforces Pod Security Standard Baseline profile or stricter for all namespaces.	☐	☐
4.3.1	Ensure that all Namespaces have Network Policies defined	☐	☐
4.5.1	Configure Image Provenance using ImagePolicyWebhook admission controller	☐	☐
4.6.2	Ensure that the seccomp profile is set to RuntimeDefault in the pod definitions	☐	☐
4.6.4	The default namespace should not be used	☐	☐
5.1.1	Ensure Image Vulnerability Scanning is enabled	☐	☐
5.1.2	Minimize user access to Container Image repositories	☐	☐
5.1.3	Minimize cluster access to read-only for Container Image repositories	☐	☐
5.1.4	Ensure only trusted container images are used	☐	☐
5.2.1	Ensure GKE clusters are not running using the Compute Engine default service account	☐	☐
5.3.1	Ensure Kubernetes Secrets are encrypted using keys managed in Cloud KMS	☐	☐
5.4.1	Enable VPC Flow Logs and Intranode Visibility	☐	☐

Recommendation		Set Correctly	
		Yes	No
5.4.2	Ensure Control Plane Authorized Networks is Enabled	☐	☐
5.4.3	Ensure clusters are created with Private Endpoint Enabled and Public Access Disabled	☐	☐
5.4.4	Ensure clusters are created with Private Nodes	☐	☐
5.4.5	Ensure use of Google-managed SSL Certificates	☐	☐
5.5.1	Manage Kubernetes RBAC users with Google Groups for GKE	☐	☐
5.6.1	Enable Customer-Managed Encryption Keys (CMEK) for GKE Persistent Disks (PD)	☐	☐
5.7.1	Enable Security Posture	☐	☐

Appendix: CIS Controls v8 Unmapped Recommendations

Recommendation		Set Correctly	
		Yes	No
4.1.8	Avoid bindings to system:anonymous	☐	☐
4.1.9	Avoid non-default bindings to system:unauthenticated	☐	☐
4.1.10	Avoid non-default bindings to system:authenticated	☐	☐

Appendix: Change History

Date	Version	Changes for this version