Security requirement

Orchestrator

Deutsche Telekom Group

Version	4.0
Date	Jul 1, 2023
Status	Released

Publication Details

Published by
Deutsche Telekom AG
Vorstandsbereich Technology & Innovation
Chief Security Officer

Reuterstrasse 65, 53315 Bonn
Germany

File name	Document number	Document type
	3.34	Security requirement
Version	**State**	**Status**
4.0	Jul 1, 2023	Released
Contact	**Validity**	**Released by**
Telekom Security psa.telekom.de	Jul 1, 2023 - Jun 30, 2028	Stefan Pütz, Leiter SEC-T-TST

Summary
This document describes the functional security requirements used to secure Orchestrators additional to the CIS Benchmark and hardening guides which also cover implementation-oriented requirements

Table of Contents

1. Introduction

This security document has been prepared based on the general security policies of the Group.

The security requirement is used as a basis for an approval in the PSA process, among other things. It also serves as an implementation standard for units which do not participate in the PSA process. These requirements shall be taken into account from the very beginning, including during the planning and decision-making processes. When implementing these security requirements, the precedence of national, international and supranational law shall be observed.

If compliance with the described requirements can not be achieved or is only partially feasible in individual cases, risk assessments must be carried out together with a Security and/or Data Privacy Expert (in accordance with the relevant requirements) and possible alternative protective measures must be agreed.

2. Requirements

2.1. General

Req 1	TLS must be enabled.

TLS must be enabled for every component that supports it to prevent traffic sniffing, verify the identity of the server, and (for mutual TLS) verify the identity of the client.
3_50_Cryptographic Algorithms and Security Protocols Document, TLS must be used in version 1.2 or 1.3 .

Motivation: Provide privacy and data integrity between two or more communicating computer applications.

Implementation example: Orchestrator components must use TLS to communicate.

For this requirement the following threats are relevant:
- Unauthorized access or tapping of data
- Unauthorized modification of data

For this requirement the following warranty objectives are relevant:

ID: 3.34-1/4.0

Req 2	Resource policies must be used.

Resource policies LimitRanges, ResourceQuotas, and Process ID Limits restrict resource usage for namespaces, nodes, or Pods. These policies are important to reserve compute and storage space for a resource and avoid resource exhaustion.

Motivation: Avoid resource exhaustion.

Implementation example: Enforce minimum and maximum compute resources usage per Pod or Container in a namespace.
Enforce minimum and maximum storage request per PersistentVolumeClaim in a namespace.
Enforce a ratio between request and limit for a resource in a namespace.
Set default request/limit for compute resources in a namespace and automatically inject them to Containers at runtime.

ID: 3.34-2/4.0

Req 3	Role-based access control(RBAC) must be used to control access.

RBAC, enabled by default, is one method to control access to cluster resources based on the roles of individuals within an organization. RBAC can be used to restrict access for user accounts and service accounts. To check if RBAC is enabled in a cluster using kubectl, execute kubectl api-versions. The API version for .rbac.authorization.k8s.io/v1 should be listed if RBAC is enabled.
Two types of permissions can be set:

- Roles – Set permissions for particular namespaces
- ClusterRoles – Set permissions across all cluster resources regardless of namespace Both Roles and ClusterRoles can only be used to add permissions.

There are no deny rules. If a cluster is configured to use RBAC and anonymous access is disabled, the Kubernetes API server will deny permissions not explicitly allowed.

Motivation: Restrict access to resources.

ID: 3.34-3/4.0

Req 4 A centralized authentification system for the API Server must be used.

Integrating Orchestrators with third party auth providers e.g. OpenID Connect Identity (OIDC) and OAuth 2.0 providers with pluggable connectors prevents administrators having to reconfigure the API server to add or remove users. Therefore you must implement them.

Motivation: Centralising authentication and authorisation (aka Single Sign On) helps onboarding, offboarding, and consistent permissions for users.

For this requirement the following threats are relevant:
- Unauthorized access to the system

For this requirement the following warranty objectives are relevant:

ID: 3.34-4/4.0

Req 5 Management and worker nodes must be separated on different hosts.

An Orchestrator must be setup containing at least one master node and two worker nodes each on dedicated hosts.

Motivation: Master nodes are critical to the operation of the cluster. If no masters are running, or the master nodes are unable to reach a quorum, then the cluster is unable to schedule and execute applications, although the workers can run without a master node.

For this requirement the following threats are relevant:
- Unauthorized access to the system
- Unauthorized access or tapping of data
- Unauthorized modification of data
- Unauthorized use of services or resources

For this requirement the following warranty objectives are relevant:

ID: 3.34-5/4.0

Req 6 Encryption keys must be rotated.

The symmetric encryption keys that the API server uses to encrypt etcd values are not automatically rotated - they must be rotated manually at least once a year. Master access is required to do this.

Motivation: Orchestrators (e.g. Kubernetes) will rotate some certificates automatically (notably, the kubelet client and server certs) by creating new CSRs as its existing credentials expire. Orchestrators do not do this for all Keys , so this must be checked.

ID: 3.34-6/4.0

Req 7 Orchestrator Hardening Guide / CIS Benchmark must be implemented.

In case of using Kubernetes the CIS Benchmark must be implemented https://cisecurity.telekom.de/de/

- CIS Level 1 Must be implemented.
- CIS Level 2 should be implemented as far as possible.

If a level 2 Requirement would break functionality or would need disproportionate effort to implement it may be skipped.

All Benchmarks can be found here e.g. for Goolge Kubernetes Engine
For Rancher they are available here : https://docs.ranchermanager.rancher.io/pages-for-subheaders/rancher-v2.6-hardening-guides

Motivation: Establishing a secure configuration posture for the Orchestrator.

For this requirement the following threats are relevant:
- Unauthorized access to the system
- Unauthorized access or tapping of data
- Unauthorized modification of data
- Unauthorized use of services or resources

For this requirement the following warranty objectives are relevant:

ID: 3.34-7/4.0

Req 8 Network and namespace policies must be used to limit communication.

A restricted default policy must be enabled (default-deny) . Access must be whitelisted explicitly.

Motivation: By default, pods are non-isolated; they accept traffic from any source. Pods become isolated by having a NetworkPolicy that selects them. Once there is any NetworkPolicy in a namespace selecting a particular pod, that pod will reject any connections that are not allowed by any NetworkPolicy. (Other pods in the namespace that are not selected by any NetworkPolicy will continue to accept all traffic.)

Use a default policy to deny all ingress and egress traffic. Ensures unselected Pods are isolated to all namespaces except kube-system

For this requirement the following threats are relevant:
- Unauthorized access to the system
- Unauthorized access or tapping of data
- Unauthorized modification of data
- Unauthorized use of services or resources

For this requirement the following warranty objectives are relevant:

ID: 3.34-8/4.0

Req 9 Security Relevant Events must be logged / Audit logging must be enabled.

Monitoring/logging should include the following:
- API request history
- Performance metrics
- Deployments
- Operating system calls
- Protocols,
- permission changes
- Network traffic
- Pod scaling
- Volume mount actions
- Image and container modification
- Privilege changes

- Scheduled job (cronjob) creations and modifications

Motivation: Logging of orchestration events for tenants is a prerequisite for the security of tenants' workloads (e.g. who, when created a specific cloud resource, modified it or deleted it).

Implementation example: Some examples include logging of:

- creation, deletion and changes of tenants (users, projects, roles)
- creation, deletion and changes to tenant networks
- creation, deletion and other virtual machine actions
- creation, deletion, modification and reading of object storage buckets and objects

ID: 3.34-9/4.0

Req 10	Key management system must be available and used for handling secrets.

Secret management plays a crucial role in security of cloud systems. Secrets in this sense include:

- keys
- passwords
- certificates

Here is the list of use case examples for KMS:

- storage encryption
- key management for SSH access
- API communication encryption
- tenant data encryption
- certificates for tenants
- secret management for tenants

The KMS may be delivered as the part of the cloud platform or separately, but it is practically impossible to run secure cloud platform or secure tenant workloads in the cloud without KMS. Therefore, even if the KMS is not delivered as the part of the cloud platform, it is necessary that the KMS is reachable and usable in automated way (like other cloud APIs).

Motivation: KMS allows generation, storage and distribution of secrets in a proper way in cloud environment. If secrets would be stored on systems in the cloud in plaintext, they could be possibly accessed by unauthorized users.

Implementation example: A solution like HashiCorp Vault can be installed as part of the cloud infrastructure.

ID: 3.34-10/4.0

Req 11	Access to administrative interfaces, such as Kubernetes APIs, dashboards and systems must be controlled and protected by higher layer firewalls or proxies (preferably hidden behind jump host), while communication must be encrypted.

Per requirement which mandates absence of direct IP connectivity from external networks and tenant networks to cloud API endpoints, access would be necessary for management functions of applications running in the cloud and for external people and/or machines. The access to cloud API endpoints must traverse through higher layer firewalls (above L4 - TCP), WAFs or proxies.

Communication from end-clients (coming from both external and tenant networks) to firewall/WAF/proxy and from firewall/WAF/proxy to the exposed cloud API endpoints must be encrypted, including both-way certificate verification to avoid man-in-the-middle attacks. Internal cloud APIs must be naturally not reachable by tenants.

.

Specific web application checks need to be used for web-based UI (e.g. dashboards).

Strong authentication must be used for getting access to all API endpoints, according to existing security requirement documents about web applications and web services (e.g. document 3.02 - "Web Services").

Motivation: Cloud API endpoints have big attack surface with relatively low built-in protection against unexpected input and potentially high impact if breached.

Implementation example: It is preferred that all administrative communication is channeled through jump host. If an approach with jump host is not possible, strict VPN or ACLs can be used to limit the access to administrative cloud interfaces, APIs, dashboards etc. for only cloud administrators.

ID: 3.34-11/4.0

Req 12 All network connections to storage must be encrypted.

All network connections (access) to data, either from hypervisors or object storage proxies for customers, must be encrypted.

Motivation: This prevents attackers on network to obtain data in clear text.

Implementation example: Use of TLS for all connections to storage system.

ID: 3.34-12/4.0

Req 13 Tenant data must be encrypted while in transit or at rest.

Depending on the data classification and security level of the cloud infrastructure (and network connecting different cloud environments), it is necessary to:

- encrypt all the network traffic between virtual machines (or containers residing in different virtual machines)
- encrypt data which is stored on block or object storage

Application deployed in the cloud might end up running on a single hypervisor, inside same cloud environment (data center) or distributed between different cloud environments (data centers). Although connectivity for the tenant may be transparent, tenants must be aware that communication between their workloads will go through network cables on public land. Therefore, all tenant traffic which leaves a single unit of workload (virtual machine or a container which talks to other containers which may reside elsewhere), it is necessary to encrypt that traffic. Best practice is to encrypt all the traffic to make sure that application doesn't need to be re-architected in case of changes in the cloud environments or further deployment of application to multiple sites. Encryption inside the data center also offers additional protection against hackers who managed to get into the cloud infrastructure or from rogue cloud administrators, so it is still beneficial to use it even if cloud environments are connected over encrypted links.

Motivation: Not all cloud environments may be deployed in fully secure locations (e.g. "edge clouds") and despite hard disk encryption offered by the platform, attackers who managed to get access to the cloud infrastructure wouldn't be able to easily extract the data if the data stored on volumes and in object storage buckets is encrypted. Encrypted data in storage buckets with proper use of different keys for different buckets also helps to prevent accidental data leaks, either to the public or to other workloads (e.g. accidental transfer from production to the test system).

Implementation example: Use of secure communications protocols (e.g. HTTPS, TLS and SSH) for communication and using encryption for data at rest - encrypted disks or objects.

ID: 3.34-13/4.0

Req 14	Patching of the cloud infrastructure must not be delayed or blocked by tenants (neither technically nor administratively).

It is mandatory that SLA reflects the fact that no tenant can block or prevent patching of the underlaying infrastructure.

The project which wants to get onboarded on cloud even with an idea that they can delay or block patching and other necessary works of the cloud infrastructure must never be onboarded.

Motivation: Patching protects all other workloads and the cloud infrastructure, and it would be extremely unreasonable and dangerous to jeopardize security of whole cloud and all other systems just because of one tenant.

Implementation example: For certain technologies and workloads this can be achieved by live migration, and for those where there is no technical capability, it can be covered by a clause in SLA terms.

ID: 3.34-14/4.0

2.2. Pod Security

Pods are the smallest deployable Kubernetes unit and consist of one or more containers.

Req 15	Sensitive information must not be stored in pod-type YAML resources.

Sensitive information must not be stored in pod-type YAML resource (deployments, pods, sets, etc.), and sensitive configmaps and secrets must be encrypted with tools such as vault , git-crypt, sealed secrets (https://github.com/bitnami-labs/sealed-secrets) Static analysis of YAML configuration can be used to establish a baseline for runtime security. kubesec e.g. generates risk scores for resources https://kubesec.io/ https://github.com/controlplaneio/kubesec .

Motivation: Prevent secrets from being visible in yaml files.

For this requirement the following threats are relevant:
- Unauthorized access to the system
- Unauthorized access or tapping of data
- Unauthorized modification of data
- Unauthorized use of services or resources

For this requirement the following warranty objectives are relevant:

ID: 3.34-15/4.0

Req 16	Pod Security Admissions or a 3rd party admission plugin must be used limit container privileges and provide platform rules.

A restricted default policy must be enabled. A very basic setup consists of a unprivileged policy and exception policies. Resources and volumes a Pod can access must be limited.

Policies can be enforced in multiple ways:

- Pod Security Admissions https://kubernetes.io/docs/concepts/security/pod-security-admission/
- Open Policy Agent / Gatekeeper https://www.openpolicyagent.org/
- Kyverno https://kyverno.io/

Policies should limit:

Control Aspect	**Field Names**

Running of privileged containers	privileged
Usage of host namespaces	hostPID, hostIPC
Usage of host networking and ports	hostNetwork, hostPorts
Usage of volume types	volumes
Usage of the host filesystem	allowedHostPaths
Allow specific FlexVolume drivers	allowedFlexVolumes
Allocating an FSGroup that owns the pod's volumes	fsGroup
Requiring the use of a read only root file system	readOnlyRootFilesystem
The user and group IDs of the container	runAsUser, runAsGroup, supplementalGroups
Restricting escalation to root privileges	allowPrivilegeEscalation, defaultAllowPrivilegeEscalation
Linux capabilities	defaultAddCapabilities, requiredDropCapabilities, allowedCapabilities
The SELinux context of the container	seLinux
The Allowed Proc Mount types for the container	allowedProcMountTypes
The sysctl profile used by containers	forbiddenSysctls,allowedUnsafeSysctls

Motivation: Most Pods don't need privileged access or even host access, so it should be ensured that a Pod requesting such access needs to be whitelisted explicitly. By default no one should be able to request privileges above the default to avoid being vulnerable through misconfiguration or malicious content of a Container image.

Implementation example: Shared Responsibility , the Platform (Orchestrator) has to provide the utility to provide this.
An Application has to utilize this.
Most Platforms have Policies defined which Applications need to follow.

For this requirement the following threats are relevant:
- Unauthorized access to the system
- Unauthorized access or tapping of data
- Unauthorized modification of data
- Unauthorized use of services or resources

For this requirement the following warranty objectives are relevant:

ID: 3.34-16/4.0