

Security requirement

Container

Deutsche Telekom Group

Version	5.0
Date	Jul 1, 2025
Status	Released

Publication Details

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

Reuterstrasse 65, 53315 Bonn
Germany

File name	Document number 3.64	Document type Security requirement
Version 5.0	State Jul 1, 2025	Status Released
Contact Telekom Security psa.telekom.de	Validity Jul 1, 2025 - Jun 30, 2030	Released by Stefan Pütz, Leiter SEC-T-TST

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

Copyright © 2025 by Deutsche Telekom AG.
All rights reserved.

Table of Contents

1.	Introduction	4
2.	Requirements	5
2.1.	General Hardening	5
2.2.	Container Linux Host Configuration	7
2.3.	Container Daemon Configuration	8
2.4.	Container Daemon Configuration Files	9
2.5.	Container Images and Build File Configuration	10
2.6.	Container Runtime Configuration	16
2.7.	Container Security Operations	21
2.8.	Registry	21
3.	Technical Baseline Security for IT/NT Systems	23
3.1.	Logging	23

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

This requirement paper outlines the essential security measures and best practices for container management and deployment.

It aims to ensure that containers are used securely, covering aspects such as container immutability, user privileges, OS configurations, etc.

The shared responsibility model for container security involves both the container runtime provider and the user (e.g., DevOps teams).

The container runtime provider is responsible for ensuring secure default configurations, implementing security mechanisms, and providing regular updates.

The user is responsible for building and maintaining secure container images, configuring container security settings, managing network and resource configurations, and ensuring compliance with security policies.

2.1. General Hardening

Req 1	Containers must be treated as immutable.
-------	--

Containers are supposed to be immutable, hence they must not be modified during runtime. Instead of patching containers while they are running, patch the image and redeploy it. You must only alter your container images by using an existing CI/CD pipeline matching the CI/CD requirements.

Motivation: You will have a fresh container after each update and in the case of a vulnerability or injection they will be cleaned during the update.

For this requirement the following threats are relevant:

- Unauthorized access to the system
- Unauthorized modification of data

ID: 3.64-1/5.0

Req 2	Production systems must be completely separated from test and development systems.
-------	--

Apart from servers, infrastructure elements such as the network and storage systems must be separated as well. A separation that is merely logical in parts is only permitted if it is not possible to bypass this logical separation and, on the other hand, it is ensured that there is no security impact on production systems when performing tests and/or development activities. A physical separation is advised. For reference systems must be decided whether these have a more productive character or being closer to a test environment.

If communication between these different system types is absolutely necessary, it can only be accomplished by employing security systems that control the entire communication. Apart from that jump hosts between these network areas should be used.

Testing activities related to security like penetration tests and network-based security scans must be possible at all times and without any impact on the production systems.

It is recommended to use clearly distinguishable IP address ranges for these different kinds of network areas.

Motivation: A sufficiently secure system status cannot be assumed in the case of test and development systems, in particular, because these are, by their nature, exposed to permanent changes. When linking system types to each other or when using the same platform, there is a risk that unauthorized parties will be able to access production systems and live data from within the test/development environment or that the stability of the production systems and thus the availability of the associated services will be put at risk. A test and development infrastructure that is completely independent from the production makes it possible to implement changes for test and development systems quickly and also during a frozen zone as they do not have an impact on the production systems. This way security updates for the production can faster be tested and accepted.

Implementation example: For an acceptance test environment, designated for security scans and penetration tests, a dedicated test environment has been chosen.

Another example is a performance test environment that is physically separated from the production, so that performance test activities do not have any negative impact on the production environment as it could be possible in the case of a shared environment, e.g., shared storage or network components.

For this requirement the following threats are relevant:

- Unauthorized access to the system
- Unauthorized access or tapping of data
- Disruption of availability
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

For this requirement the following warranty objectives are relevant:

- Availability

ID: 3.14-2/3.3

Req 3	Known vulnerabilities in the software or hardware of the system must be fixed or protected against misuse.
-------	--

Known vulnerabilities in software and hardware components must be fixed by installing available system updates from the supplier (e.g. patches, updates/upgrades). Alternatively, the use of workarounds (acute solutions that do not fix the vulnerability, but effectively prevent exploitation) is permissible. Workarounds should only be used temporarily and should be replaced by a regular system update as soon as possible in order to completely close the vulnerabilities.

Components that contain known, unrecoverable vulnerabilities must not be used in a system.

The treatment of newly discovered vulnerabilities must also be continuously ensured for the entire deployment phase of the system and implemented in the continuous operating processes of security patch management.

Motivation: The use of components without fixing contained vulnerabilities significantly increases the risk of a successful compromise. The attacker is additionally favored by the fact that, as a rule, not only detailed information on vulnerabilities that have already become known is openly available, but often also already adapted attack tools that facilitate active exploitation.

Implementation example: Following the initial installation of an operating system from an official installation medium, all currently available patches and security updates are installed.

Additional information:

The primary sources of known vulnerabilities in software/hardware are lists in the release notes as well as the security advisories from the official reporting channels of the supplier or independent CERTs. In particular, the reporting channels are sensibly integrated into continuous processes of security patch management for a system, so that newly discovered vulnerabilities can be registered promptly and led into operational remedial measures.

As a complementary measure to the detection of potentially still contained types of vulnerabilities that have in principle already become known, targeted vulnerability investigations of the system can be carried out. Particularly specialized tools such as automated vulnerability scanners are suitable for this purpose. Examples include: Tenable Nessus, Qualys Scanner Appliance.

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
- Disruption of availability
- Denial of executed activities
- Unnoticeable feasible attacks
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-10/9.0

Req 4	Software and hardware of the system must be covered by security vulnerability support from the supplier.
-------	--

Only software and hardware products for which there is security vulnerability support by the supplier may be used in a system.

Such support must include that the supplier

- continuously monitors and analyzes the product for whether it has been affected by security vulnerabilities,
- informs immediately about the type, severity and exploitability of vulnerabilities discovered in the product
- timely provides product updates or effective workarounds to remedy the vulnerabilities.

The security vulnerability support must be in place for the entire period in which the affected product remains in use.

Support phases with limited scope of services

Many suppliers optionally offer time-extended support for their products, which goes beyond the support phase intended for the general market, but is often associated with limitations. Some suppliers define their support fundamentally in increments, which may include limitations even during the final phase before the absolute end date of regular support.

If a product is used within support phases that are subject to limitations, it must be explicitly ensured that these restrictions do not affect the availability of security vulnerability support.

Open Source Software and Hardware

Open Source products are often developed by free organizations or communities; accordingly, contractually agreed security vulnerability support may not be available. In principle, it must also be ensured here that the organization/community (or a third party officially commissioned by them) operates a comprehensive security vulnerability management for the affected product, which meets the above-mentioned criteria and is considered to be reliably established.

Motivation: Hardware and software products for which there is no comprehensive security vulnerability support from the supplier pose a risk. This means that a product is not adequately checked to determine whether it is affected by further developed forms of attack or newly discovered vulnerabilities in technical implementations. Likewise, if there are existing security vulnerabilities in a product, no improvements (e.g. updates, patches) are provided. This results in a system whose weak points cannot be remedied, so that they remain exploitable by an attacker in order to compromise the system or to adversely affect it.

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
- Disruption of availability
- Denial of executed activities
- Unnoticeable feasible attacks
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-1/9.0

2.2. Container Linux Host Configuration

This section covers the best practices for hardening the Container Linux Host. It includes requirements to ensure that the host is configured securely to provide a robust foundation for running containers.

Req 5	The Container Linux Host must be hardened following best practices.
-------	---

This are security recommendations that you should follow to prepare the host machine that you plan to use for executing containerized workloads.

In order to build a solid and secure foundation for execution, the Container Linux Host must be hardened.

These recommendations may not be directly applicable for container users rather for the container runtime providers.

Here are examples of what must be considered if you are using **Docker**:

- Ensure a separate partition for containers has been created
- Ensure only trusted users are allowed to control Docker daemon
- Ensure auditing is configured for the Docker daemon
- Ensure auditing is configured for Docker files and directories - /run/containerd
- Ensure auditing is configured for Docker files and directories - /var/lib/docker
- Ensure auditing is configured for Docker files and directories - /etc/docker
- Ensure auditing is configured for Docker files and directories - docker.service
- Ensure auditing is configured for Docker files and directories - containerd.sock
- Ensure auditing is configured for Docker files and directories - docker.sock
- Ensure auditing is configured for Docker files and directories - /etc/default/docker
- Ensure auditing is configured for Docker files and directories - /etc/docker/daemon.json
- Ensure auditing is configured for Docker files and directories - /etc/containerd/config.toml
- Ensure auditing is configured for Docker files and directories - /etc/sysconfig/docker
- Ensure auditing is configured for Docker files and directories - /usr/bin/containerd
- Ensure auditing is configured for Docker files and directories - /usr/bin/containerd-shim
- Ensure auditing is configured for Docker files and directories - /usr/bin/containerd-shim-runc-v1
- Ensure auditing is configured for Docker files and directories - /usr/bin/containerd-shim-runc-v2
- Ensure auditing is configured for Docker files and directories - /usr/bin/runc

*considered if you are using **containerd** or any other runtime some docker specific files will not be available, hardening can still be achieved for non docker specific files and must be done.*

Motivation: An un-hardened host may contain vulnerabilities that attackers could exploit to gain unauthorized access, introduce malware, or compromise container workloads.

ID: 3.64-5/5.0

2.3. Container Daemon Configuration

This section outlines the requirements for hardening the Container Daemon. The settings that are under this section affect ALL container instances.

Req 6	The Container Daemon must be hardened following best practices.
-------	---

This requirement lists the recommendations that alter and secure the behavior of the Container daemon.

These recommendations may not be directly applicable for container users rather for the container runtime providers.

Here are examples of what must be considered if you are using **Docker**:

- Run the container daemon as a non-root user, if possible
- Ensure network traffic is restricted between containers on the default bridge
- Ensure the logging level is set to 'info'
- Ensure Docker is allowed to make changes to iptables
- Ensure insecure registries are not used
- Ensure aufs storage driver is not used
- Ensure TLS authentication for container daemon is configured
- Ensure the default ulimit is configured appropriately
- Enable user namespace support
- Ensure the default cgroup usage has been confirmed
- Ensure base device size is not changed until needed

- Ensure that authorization for Docker client commands is enabled
- Ensure centralized and remote logging is configured
- Ensure containers are restricted from acquiring new privileges
- Ensure live restore is enabled
- Ensure Userland Proxy is Disabled
- Ensure that a daemon-wide custom seccomp profile is applied if appropriate
- Ensure that experimental features are not implemented in production

*considered if you are using **containerd** or any other runtime some docker specific settings will not be available, hardening can still be achieved for non docker specific configurations and must be done.*

Motivation: Hardening the container daemon according to best practices reduces security risks and protects against attacks and unauthorized access.

ID: 3.64-6/5.0

2.4. Container Daemon Configuration Files

This section outlines the requirements for hardening specifically the Container Daemon configuration files. It covers Container Daemon related files and directory permissions and ownership.

For proper and secure functioning of the Daemon, files and directories, that may contain sensitive parameters must be kept protected.

Req 7 The Container Daemon configuration files must be hardened following best practices.

This requirement covers Container Daemon related files and directory permissions and ownership. Keeping the files and directories, that may contain sensitive parameters, secure is important for correct and secure functioning of the Container daemon.

These recommendations may not be directly applicable for container users rather for the container runtime providers.

Here are examples of what must be considered if you are using **Docker**:

- Ensure that the docker.service file ownership is set to root:root
- Ensure that docker.service file permissions are appropriately set
- Ensure that docker.socket file ownership is set to root:root
- Ensure that docker.socket file permissions are set to 644 or more restrictive
- Ensure that the /etc/docker directory ownership is set to root:root
- Ensure that /etc/docker directory permissions are set to 755 or more restrictively
- Ensure that registry certificate file ownership is set to root:root
- Ensure that registry certificate file permissions are set to 444 or more restrictively
- Ensure that TLS CA certificate file ownership is set to root:root
- Ensure that TLS CA certificate file permissions are set to 444 or more restrictively
- Ensure that Docker server certificate file ownership is set to root:root
- Ensure that the Docker server certificate file permissions are set to 444 or more restrictively
- Ensure that the Docker server certificate key file ownership is set to root:root
- Ensure that the Docker server certificate key file permissions are set to 400
- Ensure that the Docker socket file ownership is set to root:docker
- Ensure that the Docker socket file permissions are set to 660 or more restrictively
- Ensure that the daemon.json file ownership is set to root:root
- Ensure that daemon.json file permissions are set to 644 or more restrictive
- Ensure that the /etc/default/docker file ownership is set to root:root

- Ensure that the `/etc/default/docker` file permissions are set to 644 or more restrictively
- Ensure that the `/etc/sysconfig/docker` file permissions are set to 644 or more restrictively
- Ensure that the `/etc/sysconfig/docker` file ownership is set to root:root
- Ensure that the Containerd socket file ownership is set to root:root
- Ensure that the Containerd socket file permissions are set to 660 or more restrictively

*considered if you are using **containerd** or any other runtime some docker specific settings will not be available, hardening can still be achieved for non docker specific files and it must be done.*

Motivation: Hardening the container daemon configuration files according to best practices protects against misconfigurations and prevents potential attacks.

ID: 3.64-7/5.0

2.5. Container Images and Build File Configuration

This section addresses the security of container images and build files.

Req 8 Container base images must be hardened following best practices.

Recommendations that you should follow for container base images and build files to ensure that your containerized infrastructure is secure.

These recommendations can apply to container users and container runtime providers according to the shared responsibility model.

- Ensure that a user for the container has been created
- Ensure Content trust for Docker is Enabled
- Ensure that HEALTHCHECK instructions have been added to container images
- Ensure update instructions are not used alone in Dockerfiles
- Ensure `setuid` and `setgid` permissions are removed
- Ensure that `COPY` is used instead of `ADD` in Dockerfiles

Motivation: Hardening container base images according to best practices reduces vulnerabilities and protects deployments from attacks through compromised images.

ID: 3.64-8/5.0

Req 9 Fixed tags must be used for immutability.

Use the most specific tag available. If an image has multiple tags e.g. `:8` and `:8.0.1` or `:8.0.1-alpine`, you must prefer the last, as it is the most specific reference. Avoid using generic tags like `:latest`. Remember that specific tags might be deleted.

Since tags in the container ecosystem are generally not immutable, a tag such as `:8.0.1-alpine`—no matter how precise it may seem—can be overwritten by a different image in the future. True immutability and traceability can therefore only be ensured by specifying an image digest and pinning the image version to a specific digest.

Motivation: To avoid a specific image tag to become unavailable you must be running a trusted registry or account that is under your own control. Building images by yourself can be an advantage, because you maintain control of all components shipped with it.

For this requirement the following threats are relevant:

- Unauthorized modification of data
- Disruption of availability

Req 10 Least privileged user must be used.

When a Containerfile doesn't specify a user, it defaults to executing the container using the root user. When that namespace is then mapped to the root user in the running container, it means that the container potentially has root access on the Docker host. The container must not run using the default root user, privileges must be dropped and a specific user must be used.

Motivation: Prevent containers running as the default root user.

For this requirement the following threats are relevant:

- Unauthorized access to the system
- Unauthorized modification of data
- Unauthorized use of services or resources

Req 11 Container-specific host OS must be used.

Container-specific host operating systems must be used instead of general-purpose OSs. When using a container-specific host OS, attack surfaces are typically much smaller than they would be with a general-purpose host OS, so there are fewer opportunities to attack and compromise a container-specific host OS like e.g.: RedHat Atomic or Core OS.

If not possible to use a Container specific Host OS it must be at least hardened according to Linux/Windows SoCs.

Motivation: Container-specific OSs reduce the attack surfaces because of the minimalistic approach to run exclusively containers.

Implementation example: Use a minimal Container Images.

There are also minimal images available in DT <https://devsecops-tools.pages.devops.telekom.de/user-documentation/docs/mcc/baseimages/> using them already takes care of some hardening according to SoCs and helps to start with a solid foundation.

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

Req 12 Unnecessary packages must be avoided.

Each package added to a container image increases the potential attack surface. By minimizing the number of packages, you reduce the number of vulnerabilities that could be exploited by attackers.

Motivation: Avoiding unnecessary packages reduces the attack surface and minimizes potential security risks.

Req 13 Containers must be scanned for vulnerabilities.

Images must be scanned within your registry for known vulnerabilities. This also must be done during runtime.

Motivation: Find known vulnerabilities within containers.

Implementation example: Can be fulfilled with : <https://dtsp.telekom-dienste.de/login?redirect=%2F>

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

ID: 3.64-13/5.0

Req 14	Secret management must be provided.
--------	-------------------------------------

Secrets must be stored outside of images and provided dynamically at runtime as needed.

Motivation: Most orchestrators, such as Docker Swarm and Kubernetes, include native management of secrets. These orchestrators not only provide secure storage of secrets and 'just in time' injection to containers, but also make it much simpler to integrate secret management into the build and deployment processes. For example, an organization could use these tools to securely provision the database connection string into a web application container. The orchestrator can ensure that only the web application container has access to this secret, that it is not persisted to disk, and that anytime the web app is deployed, the secret is provisioned into it.

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

ID: 3.64-14/5.0

Req 15	Unnecessary services must be disabled.
--------	--

After the installation of systems and software products, supplier-preset, local or network-accessible services are often active that are not required for the operation and functionality of the specific system in the intended operating environment.

However, in principle only the services actually required may be active on a system.

Accordingly, all services that are not required on a system must be completely disabled immediately after installation. It must be ensured that these services remain disabled even after the system is restarted.

Motivation: Active services that are not required unnecessarily increase the attack surface of a system and, as a direct consequence, the risk of a successful compromise. This risk can be further increased if - as is often observed with services that are not required - a targeted examination and optimization of the configuration with regard to security does not take place sufficiently.

For this requirement the following threats are relevant:

- Unauthorized use of services or resources
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-5/9.0

Req 16	The software used must be obtained from trusted sources and checked for integrity.
--------	--

The software used on the system must be obtained from trusted sources and checked for integrity before installation. This requirement applies to all types of software:

- Firmware and microcode for hardware components
- Operating systems
- Software Libraries
- Application Software
- Pre-integrated application solutions, such as software appliances or containers

as well as other software that may be used.

Trusted Sources

Trusted sources are generally considered to be:

- the official distribution and supply channels of the supplier
- third party distributors, provided they are authorized by the supplier and are a legitimate part of the supplier's delivery channels
- internet downloads, if they are made from official provisioning servers of the supplier or authorized distributors
 - (1) If the provisioning server offers various forms of downloads, those protected by encryption or cryptographic signatures must be preferred to those without such protection.
 - (2) If the provisioning server secures the transport layer using cryptographic protocols (e.g. HTTPS, sftp), the associated server certificates or server keys/fingerprints must be validated with each download to confirm the identity of the provisioning server; if validation fails, the download must be cancelled and the provisioning server has to be considered an untrusted source.

Integrity Check

The integrity check is intended to ensure that the received software is free of manipulation and malware infection. If available, the mechanisms implemented by the supplier must be used for checking.

Valid mechanisms are:

- physical seals or permanently applied certificates of authenticity (if the software is provided on physical media)
- comparison of cryptographic hash values (e.g. SHA256, SHA512) of the received software against target values, which the supplier provides separately
- verification of cryptographic signatures (e.g. GPG, certificates) with which the supplier provides its software

In addition, a check of the software using an anti-virus or anti-malware scanner is recommended (if the vendor has not implemented any of the aforementioned integrity protection mechanisms for its software, this verification is mandatory).

Extended integrity checking when pulling software from public registries

Public registries allow developers to make any of their own software projects available for use. The range includes projects from well-known companies with controlled development processes, as well as from smaller providers or amateur developers.

Examples of such registries are:

- Code registries (e.g. GitHub, Bitbucket, SourceForge, Python Package Index)
- Container registries (e.g. Docker Hub)

Software from public registries must undergo an extended integrity check before deployment.

In addition to the integrity check components described in the previous section, the extended check is intended to explicitly ensure that the software actually performs its function as described, does not contain inherent security risks such as intentionally implemented malware features, and is not affected by known security vulnerabilities. If the software has direct dependencies on third-party software projects (dependencies are very typical in open source software), which must also be obtained and installed for the use of the software, these must be included in the extended integrity check.

Methods for an extended integrity check can be, for example:

- Strict validation of project/package names (avoidance of confusion with deliberately imitated malicious software projects)

- dynamic code analysis / structured functional checks in a test environment
- static code analysis using a linter (e.g. Splint, JSLint, pylint)
- Examination using a security vulnerability scanner (e.g. Qualys, Nessus)
- Examination using a container security scanner (e.g. JFrog Xray, Harbor, Clair, Docker Scan)
- Examination using an SCA (Software Composition Analysis) tool or dependency scanner (e.g. OWASP Dependency Check, Snyk)

The test methods must be selected and appropriately combined according to the exact form of software delivery (source code, binaries/artifacts, containers).

Motivation: Software supply chains contain various attack vectors. An attacker can start at various points to manipulate software or introduce his own routines and damage or control the target environment in which the software is subsequently used. The attack can occur on the transport or transmission path or on the provisioning source itself. Accordingly, an attack is facilitated if software is not obtained from official and controlled sources or if an integrity check is omitted.

There is a particular risk for software obtained from public registries, as these are open to anyone for the provision of software projects. Perfidious attack methods are known, in which the attacker first provides completely inconspicuous, functional software for a while and as soon as it has established itself and found a certain spread, deliberately hidden malicious code is integrated in future versions. Other methods rely on similar-sounding project names for widely used existing projects or overruling version numbers to inject manipulated software into any solutions based on them.

Implementation example: Obtain the software via the official delivery channels of the supplier. Upon receipt of the software, immediately check for integrity using cryptographic checksums, as provided by the supplier, as well as scan for any infections by known malware using anti-malware / anti-virus scanners. Storage of the tested software on an internal, protected file storage and further use (e.g. rollout to the target systems) only from there.

For this requirement the following threats are relevant:

- Unauthorized modification of data
- Unnoticeable feasible attacks
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-2/9.0

Req 17 Only required software may be used on the system.

In the installation routines for software provided by the supplier, individual components of the software are often preselected as standard installations, which are not necessary for the operation and function of a specific system. This also includes parts of software that are installed as application examples (e.g. default web pages, sample databases, test data), but are typically not used afterwards.

Such components must be specifically deselected (not installed) during the installation of the system or - if deselection during installation is not possible - removed immediately afterwards.

In principle, no software may be used that is not required for the operation, maintenance or function of the system.

Motivation: Vulnerabilities in a system's software are gateways for attackers. By uninstalling unnecessary components, the potential attack surfaces can be significantly reduced.

For this requirement the following threats are relevant:

- Unauthorized use of services or resources
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-3/9.0

Req 18 The accessibility of activated services must be restricted.

In principle, a service provided must be completely deactivated on all interfaces of the system through which accessib-

ility of the service is not required for the proper operation of the system. The deactivation is primarily to be implemented by a corresponding configuration of the service or operating system. In cases where the available configuration options do not allow deactivation on individual interfaces, a local filter ("Host Firewall") may instead be used on the system to block access to the service via unnecessary interfaces.

The accessibility of a service via the required interfaces must also be restricted to legitimate communication partners. The restriction must be implemented by a corresponding configuration of the service or operating system or by means of a local filter ("Host Firewall"). Alternatively, this task may be outsourced to a network-side filter element, provided that the system is located in a separate network segment and communication with this segment is only possible via the network-side filter element.

Motivation: By deactivating services on interfaces through which accessibility is not necessary, as well as by restricting possible communication partners, the attack surface offered by a system can be greatly reduced.

Implementation example: An SNMP service used to monitor a system is enabled exclusively on the dedicated management network interface of the system. A firewall also regulates that only the legitimate monitoring system of the infrastructure environment can reach this service.

For this requirement the following threats are relevant:

- Unauthorized use of services or resources
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-6/9.0

Req 19	Features that are not required in the software and hardware used must be deactivated.
--------	---

During the initial installation of software, features may have been activated by default that are not necessary for the operation and functionality of the specific system. Features are usually an integral part of the software that cannot be deleted or uninstalled individually.

Such features must be disabled immediately after the initial installation through the software's configuration settings, so that they remain permanently disabled even after the system is rebooted.

Even before delivery or during initial commissioning, features may have been activated by default in the hardware that are not required for the purpose of the specific system. Such functions, for example unnecessary interfaces, must also be permanently deactivated immediately after initial commissioning.

Motivation: A system's hardware or software often contains enabled features that are not being used. Such features can be an unnecessary target for manipulation. Furthermore, there is a potential that unauthorized access to areas or data of the system can be created.

Implementation example: [Example 1]

Deactivation of debugging functions in the software that are used in the event of fault analysis, but do not have to be active during normal operation.

[Example 2]

Disabling unused network interfaces of a server.

For this requirement the following threats are relevant:

- Unauthorized use of services or resources
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-4/9.0

Req 20	The permissions for users and applications must be limited to the extent necessary to fulfill their tasks.
--------	--

The permissions on a system must be restricted to such an extent that a user can only access data and use functions that he needs in the context of his work. Appropriate permissions must also be assigned for access to files that are part

of the operating system or applications or that are generated by the same (e.g. configuration and logging files).

In addition to access to data, applications and their components must also be executed with the lowest possible permissions. Applications should not be run with administrator or system privileges.

Motivation: If a user is granted too far-reaching permissions on a system, he can access data and applications to an extent that is not necessary for the fulfillment of the assigned tasks. This creates an unnecessarily increased risk in the event of abuse, in particular if the user or his user account is compromised by an attacker.

Applications with too far-reaching permissions can be misused by an attacker to gain or expand unauthorized access to data and system areas in need of protection.

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
- Denial of executed activities

ID: 3.01-23/9.0

2.6. Container Runtime Configuration

This section focuses on the security of the container runtime configuration.

Req 21	Deployments/Applications must specify their Security Context.
--------	---

The following must be defined within the Security Context:

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

The following restrictions must be defined in the Security Context to ensure the security of Kubernetes workloads:

Prohibition of Privileged Containers

privileged: false must be set in the Security Context.

This prevents containers from running with root privileges on the host.

Prohibition of Host PID Namespace Usage

hostPID: false must be set in the Pod spec.

Prevents containers from viewing or manipulating processes on the host system.

Prohibition of Host IPC Namespace Usage

hostIPC: false must be set in the Pod spec.

Prevents the use of host inter-process communication mechanisms.

Prohibition of Host Network Namespace Usage

hostNetwork: false must be set in the Pod spec.

Prevents direct access to the host's network interface.

Prohibition of Privilege Escalation

allowPrivilegeEscalation: false must be set in the Security Context.

Prevents container processes from expanding their privileges.

Prohibition of Root Containers

runAsNonRoot: true must be set in the Security Context.

Enforces containers to run as non-root users.

Prohibition of NET_RAW Capability

The NET_RAW capability must explicitly not be included in the list of capabilities.add in the Security Context.

This capability enables the creation of raw sockets and should be removed for security reasons.

Prohibition of Additional Capabilities

No additional capabilities should be added through capabilities.add in the Security Context.
Limits the possibilities for container escape attacks.

Removal of All Capabilities

capabilities.drop: ["ALL"] must be set in the Security Context.
Removes all Linux capabilities by default and increases security.

Prohibition of Windows HostProcess Containers

For Windows containers, windowsOptions.hostProcess: false must be set.
Prevents containers from interacting directly with the Windows host.

Prohibition of HostPath Volumes

No hostPath volumes should be defined in the Pod spec.
Prevents direct access to the host filesystem.

Prohibition of Host Ports Usage

No hostPort definitions should be present in the container spec.
Prevents containers from binding directly to host ports.

Enforcement of ReadOnly Filesystems

readOnlyRootFilesystem: true must be set in the Security Context.
Prevents write access to the container filesystem.

Motivation: These requirements implement multiple layers of protection against container escape vulnerabilities, which could otherwise allow malicious containers to break out and access the host system.

Principle of Least Privilege: By restricting capabilities, access rights, and namespaces, containers operate with only the minimum privileges necessary to function, reducing the attack surface significantly.

Prevention of Privilege Escalation: These controls prevent containers from gaining additional privileges during runtime, which is a common technique used in container escape attacks.

Host System Protection: By isolating containers from host resources (network, IPC, PID namespaces), the host system is protected from direct manipulation by containerized applications.

Implementation example: <h1>apiVersion: apps/v1 </h1>
kind: Deployment
metadata:
 name: compliant-nginx
 labels:
 app: compliant-nginx
spec:
 replicas: 3 # Multiple replicas for high availability
 selector:
 matchLabels:
 app: compliant-nginx
 template:
 metadata:
 labels:
 app: compliant-nginx
 annotations:
 # SECURITY: Applies seccomp profile to restrict syscalls, reducing attack surface
 seccomp.security.alpha.kubernetes.io/pod: "runtime/default"
 # SECURITY: Applies AppArmor profile to restrict file access and capabilities
 container.apparmor.security.beta.kubernetes.io/compliant-nginx: "runtime/default"
 spec:
 securityContext:
 # SECURITY: Prevents container from running as root user
 runAsNonRoot: true
 # SECURITY: Specifies non-root user ID to run container processes
 runAsUser: 101
 # SECURITY: Sets group ID for container processes

```

runAsGroup: 101
# SECURITY: Sets group ID for mounted volumes
fsGroup: 101
# SECURITY: Applies seccomp profile at pod level (newer API)
seccompProfile:
  type: RuntimeDefault
containers:
- name: compliant-nginx
  # SECURITY: Uses specific version instead of "latest" to ensure consistency and proper security testing
  image: nginx:1.25.3-alpine
  # SECURITY: Avoids automatically pulling potentially compromised new images
  imagePullPolicy: IfNotPresent
  securityContext:
    # SECURITY: Prevents processes from gaining additional privileges
    allowPrivilegeEscalation: false
    # SECURITY: Prevents modification of container filesystem during runtime
    readOnlyRootFilesystem: true
    capabilities:
      # SECURITY: Drops all Linux capabilities by default
      drop: ["ALL"]
    # SECURITY: Explicitly disables privileged mode which would give host-level access
    privileged: false
  resources:
    # SECURITY: Sets resource limits to prevent resource exhaustion attacks
    limits:
      cpu: "500m"
      memory: "256Mi"
    requests:
      cpu: "100m"
      memory: "128Mi"
  ports:
    # SECURITY: Uses non-privileged port (>1024) to minimize required capabilities
    - containerPort: 8080
      name: http
  volumeMounts:
    # SECURITY: Mounts required writable directories as isolated volumes since root filesystem is read-only
    - name: tmp-volume
      mountPath: /tmp
    - name: nginx-cache
      mountPath: /var/cache/nginx
    - name: nginx-run
      mountPath: /var/run
    # SECURITY: Mounts configuration as read-only to prevent runtime modification
    - name: nginx-conf
      mountPath: /etc/nginx/conf.d
      readOnly: true
    # SECURITY: Implements health probes to ensure container is functioning properly
  livenessProbe:
    httpGet:
      path: /
      port: http
    initialDelaySeconds: 10
    periodSeconds: 20
  readinessProbe:
    httpGet:
      path: /
      port: http
    initialDelaySeconds: 5
    periodSeconds: 10
  volumes:
    # SECURITY: Uses ephemeral volumes instead of persistent storage for writable directories
    - name: tmp-volume

```

```

emptyDir: {}
- name: nginx-cache
emptyDir: {}
- name: nginx-run
emptyDir: {}
# SECURITY: Uses ConfigMap for configuration instead of hardcoded values
- name: nginx-conf
configMap:
  name: compliant-nginx-config
# SECURITY: Prevents pod from accessing host network interfaces
hostNetwork: false
# SECURITY: Prevents pod from viewing or manipulating host processes
hostPID: false
# SECURITY: Prevents pod from accessing host IPC mechanisms
hostIPC: false
# SECURITY: Prevents automatic mounting of service account token to limit access to Kubernetes API
automountServiceAccountToken: false

```

ID: 3.34-32/5.0

Req 22 Container runtime configuration must be hardened following best practices.

Runtime parameters can be supplied that have security consequences that could compromise the host and the containers running on it. It is therefore very important to verify the way in which containers are started, and which parameters are associated with them.

Container must ensure the following:

- Ensure swarm mode is not Enabled, if not needed
- Ensure that, if applicable, an AppArmor Profile is enabled
- Ensure that, if applicable, SELinux security options are set
- Ensure that Linux kernel capabilities are restricted within containers
- Ensure that privileged containers are not used
- Ensure sensitive host system directories are not mounted on containers
- Ensure sshd is not run within containers
- Ensure privileged ports are not mapped within containers
- Ensure that only needed ports are open on the container
- Ensure that the host's network namespace is not shared
- Ensure that the memory usage for containers is limited
- Ensure that CPU priority is set appropriately on containers
- Ensure that the container's root filesystem is mounted as read only
- Ensure that incoming container traffic is bound to a specific host interface
- Ensure that the 'on-failure' container restart policy is set to '5'
- Ensure that the host's process namespace is not shared
- Ensure that the host's IPC namespace is not shared
- Ensure that host devices are not directly exposed to containers
- Ensure that the default ulimit is overwritten at runtime if needed
- Ensure mount propagation mode is not set to shared
- Ensure that the host's UTS namespace is not shared
- Ensure the default seccomp profile is not Disabled
- Ensure that docker exec commands are not used with the privileged option
- Ensure that docker exec commands are not used with the user=root option
- Ensure that cgroup usage is confirmed

- Ensure that the container is restricted from acquiring additional privileges
- Ensure that container health is checked at runtime
- Ensure that Docker commands always make use of the latest version of their image
- Ensure that the PIDs cgroup limit is used
- Ensure that Docker's default bridge "docker0" is not used
- Ensure that the host's user namespaces are not shared
- Ensure that the Docker socket is not mounted inside any containers
- Ensure containers have a readonly Filesystem

*considered if you are using **containerd** or any other runtime some docker specific files will not be available, hardening can still be achieved for non docker specific configuration and must be done.*

Motivation: Hardening container runtime configuration according to best practices protects against misconfigurations and reduces potential attack surfaces.

ID: 3.64-22/5.0

Req 23 Namespace isolation must be ensured.

Resources a container can interact with must be limited. This includes file systems, network interfaces, interprocess communications, host names, user information, and processes. Ensure that apps and processes inside a container only see the physical and virtual resources allocated to that container. Namespace isolation provides each container with its own networking stack, including unique interfaces and IP addresses.

Motivation: Make containers more isolated

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

ID: 3.64-23/5.0

Req 24 Security zones must be established.

Containers that are assigned to different security zones (e.g. DMZ-Zone, Application/Database- Zone) must be deployed on different hosts/host pools. A pool refers to a group of at least one or more VMs. In case of using Container as a Service (CaaS) platforms, these pools can be shared for several applications. For critical applications dedicated hosts/host pools must be used. If the only exposed service is an ingress controller (e.g. nginx) hardened regarding Security guidelines you can implement this by using only container specific measures e.g. Pod/Network Security Policies.

Motivation: Segmenting containers by purpose, sensitivity, and threat posture provides additional defense in depth. By grouping containers in this manner, it's more difficult for an attacker who compromises one of the groups to expand that compromise to other groups.

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

ID: 3.64-24/5.0

Req 25 Linux kernel capabilities must be restricted within containers.

By default containers start with a restricted set of Linux kernel capabilities. This means that any process can be granted the required capabilities instead of giving it root access. Using Linux kernel capabilities, processes in general do not need to run as the root user.

Rationale:

You must remove all capabilities not required for the correct function of the container. Specifically, in the default capability set provided by Docker, the NET_RAW capability should be removed if not explicitly required, as it can give an attacker with access to a container the ability to create spoofed network traffic

Overview of Docker capabilities : <https://docs.docker.com/engine/security/#linux-kernel-capabilities>

Motivation: The Linux kernel is able to narrow down the privileges of the Root user into distinct units referred to as capabilities.

Minimize attack surface of a container by restricting it to only the capabilities it needs to function.

Implementation example: Example for Docker :

```
$ docker run --cap-drop ALL
```

Now add specific privileges to the container with the **-cap-add** argument.

ID: 3.64-25/5.0

2.7. Container Security Operations

Req 26 Image sprawl must be avoided

Each additional image increases the potential attack surface. By limiting the number of images, you minimize the number of vulnerabilities that could be exploited by attackers.

Motivation: Avoiding image sprawl reduces the attack surface and simplifies the management and security of the container environment.

ID: 3.64-26/5.0

Req 27 Container sprawl must be avoided.

Each additional container increases the potential attack surface. By limiting the number of containers, you minimize the number of vulnerabilities that could be exploited by attackers.

Motivation: Avoiding container sprawl simplifies management, reduces the attack surface, and minimizes security risks.

ID: 3.64-27/5.0

2.8. Registry

This section outlines the requirements for container registries.

Req 28 Registries must have user management in place.

For any registry, there must be a multi tenancy support, either done by the use of dedicated environments or by user management on the same instance.

Motivation: Separate and manage access to registries.

ID: 3.64-28/5.0

Req 29 Images from external registries must be downloaded to an internal registry first.

Images from external registries must be downloaded to an internal registry first so they can be scanned for vulnerabilities there. Within DT you can use the Magenta Trusted Registry <https://mtr.devops.telekom.de/>

Motivation: Ongoing, continuously updated, centralized reporting and monitoring of image compliance/vulnerabilities.

For this requirement the following threats are relevant:

- Unauthorized access to the system
- Unauthorized modification of data

ID: 3.64-29/5.0

Req 30 Connections to registries must be encrypted.

Registries must be configured to only be reachable from development tools, orchestrators and container runtimes over encrypted channels.

Motivation: Key goal is to ensure that all data pushed to and pulled from a registry occurs between trusted endpoints and is encrypted in transit.

For this requirement the following threats are relevant:

- Unauthorized access to the system
- Unauthorized access or tapping of data
- Unauthorized modification of data

ID: 3.64-30/5.0

Req 31 Sufficient authentication and authorization must be used.

Any write access to a registry must require authentication. For example, only allow developers to push images to the specific repositories they are responsible for, rather than being able to update any repository. Registries also provide an opportunity to apply context-aware authorization controls to actions. For example, you can configure your continuous integration processes to allow images to be signed by the authorized personnel and pushed to a registry only after they have passed a vulnerability scan and compliance assessment.

Motivation: Ensure that only images from trusted entities can be added to registries.

For this requirement the following threats are relevant:

- Unauthorized access to the system
- Unauthorized access or tapping of data
- Unauthorized modification of data

ID: 3.64-31/5.0

3. Technical Baseline Security for IT/NT Systems

3.1. Logging

Req 32 The system clock must be synchronized to an accurate reference time (Time Standard).

A time reference source must be used which provides a time signal based on the Coordinated Universal Time ("UTC" = "Universal Time Coordinated").

Please Note: The UTC-synchronized system time may be transformed to local time using a corresponding timezone configuration setup for any output of time information, as long as this timezone adjustment is fully accountable.

Systems belonging to the same security domain must synchronize to one and the same time reference source.

Motivation: Reference time synchronization may be a technical prerequisite for many time-dependent mechanisms, for example: Validation of Certificates; Authentication. It is also much-needed to generate exact timestamps for logged events, since without the often required time-related correlation in case of a Security Incident or during a Problem Analysis cannot be achieved.

Implementation example: some valid time reference sources:

- trustworthy NTP ("NetworkTimeProtocol") Server on the IP network
- DCF77 radio signal received via a physically connected receiver
- GPS radio signal received via a physically connected receiver

For this requirement the following threats are relevant:

- Disruption of availability
- Denial of executed activities
- Unnoticeable feasible attacks

ID: 3.01-32/9.0

Req 33 Security-related events must be logged with an exact timestamp and a unique system reference.

Security-relevant are any events that may have an impact on the information security (confidentiality, integrity, availability) of a system or indicate its misuse. Systems must log the occurrence of security-related events.

In order to enable the assignment and evaluation of these events, they must be logged together with a unique system name (e.g. host name, IP or MAC address) and the exact time of occurrence ("time stamp"). At a minimum, the timestamp of a logged event must include the following information:

- date of the event (Year, Month, Day)
- time of the event (Hours, Minutes, Seconds)
- Timezone, those information belongs to

The security relevance of specific events depends on the type and function of the system as well as the data processed in it. Therefore, it is not possible to specify a uniform list of events that are relevant to security. Relevant events must be identified according to the system. The following list contains typical security-related events and is accordingly to be understood as a basic guide for the identification of security-relevant events for a particular system.

Security-related Event	Information to log to the event
Failed login attempts	• affected user account • number of failed attempts • source of access (IP address, client-ID/client-name, local console, ...)

System access via user accounts (including system accounts or technical user accounts)	• affected user account • Access start/end time (login/logout) • source of access (IP address, client-ID/client-name, local console, ...)
Modifications of user accounts (including system accounts or technical user accounts)	• user account used to make the modification • user account to which the modification was made • type of modification made (create, activate, deactivate, delete, password reset, ...)
Changes to role assignments/permissions of user accounts (including system accounts or technical user accounts)	• user account used to make the modification • user account to which the modification was made • name of the affected role, type of the affected permission • type of modification made (role/permission added, role/permission removed, ...)
Use/initiation of critical system functions	• user account used to make the use • name/type of system function used
Access to sensitive data processed in the system	• user account used to access the data • Name of the affected data field (the contents of the data field should not be logged to prevent data leakage via log files) • type of access (read, modify, delete, ...)
Abnormal increase in the utilization of system resources, such as memory, CPU, network, or disk access, over an extended period of time	• threshold of the affected system resource • duration of exceeding the threshold • threshold exceedance Depending on the individual system, sensible threshold values must be defined.
Other events that may provide evidence of attempted or successful attacks against the system *)	• individually depending on the type of event

*) This requires identifying the most important forms of attack that the system can typically expect and ensuring that events relevant to their detection are actually logged by the system. As a rule, this can only be guaranteed by the developer/manufacturer of the system.

When logging, the applicable legal and operational regulations must be observed. The latter also include agreements that have been made with the company's social partners. Following these regulations logging of events is only allowed for a defined use case. Logging of events for doing a work control of employees is not allowed.

In addition - as for any data that is processed by a system - an appropriate protection requirement must also be taken into account and implemented for logging data; this applies to storage, transmission and access. In particular, if the logging data contains real data, the same protection requirements must be taken into account that is also used for the regular processing of this real data within the source system.

Motivation: Logging security-relevant events is a basic requirement for detecting ongoing attacks as well as attacks that have already occurred. This is the only way in which suitable measures can be taken to maintain or restore system security. Logging data could be used as evidence to take legal steps against attackers.

For this requirement the following threats are relevant:

- Denial of executed activities
- Unnoticeable feasible attacks

ID: 3.01-33/9.0

Req 34	Applicable retention and deletion periods must be observed for security-relevant logging data that is recorded locally.
--------	---

From an IT security perspective, local storage of security-relevant logging data on a system is not mandatory. Since the local storage can be damaged in the event of system malfunctions or manipulated by a successful attacker, it can only be used to a limited extent for security-related or forensic analyses. Accordingly, it is relevant for IT security that logging data is forwarded to a separate log server.

Local storage can nevertheless take place; for example, if local storage is initially indispensable when generating the logging data due to technical processes or if there are justified operational interests in also keeping logging data available locally.

The following basic rules must be taken into account when storing logging data locally:

- Security-related logging data must be retained for a period of 90 days.
(*This requirement only applies if no additional forwarding to a separate log server is implemented on the system and the logging data is therefore only recorded locally.*)
- After 90 days, stored logging data must be deleted immediately.

Deviances

Different retention periods and deletion periods may exist due to legal or regulatory requirements (especially in connection with personal data) or may be defined by contractual agreements. In these cases, the applicable periods must be agreed individually with a Project Security Manager (PSM) / Data Privacy Advisor (DPA) or are specified by them.

Motivation: Logging data is an immensely important IT security tool for preventing, detecting and clearing up system faults, security and data privacy incidents. On the other hand, the recording of logging data, like any other data processing, is also subject to legal and regulatory requirements. Accordingly, guidelines must be adhered to that reconcile the two.

Implementation example: Taking into account the current legal situation and applicable data privacy regulations, the following deletion periods for locally stored security-relevant logging data are implemented on an exemplary telecommunications system:

- Standard System Logs: Deletion after 90 days at the latest
- Logging of public IP addresses: Deletion (or anonymization) after 7 days at the latest
- Logging of the assignment of dynamic public IP addresses by the telecommunication solution: Deletion after 7 days at the latest
- Logging of non-billing-relevant call detail records: Deletion after 7 days at the latest
- Logging of the content of e-mail and SMS: Deletion after 24 hours at the latest
- Logging of the domain queries handled by the DNS server of the telecommunications solution: Deletion after 24 hours at the latest

For this requirement the following threats are relevant:

- Unauthorized access or tapping of data
- Denial of executed activities
- Unnoticeable feasible attacks

ID: 3.01-34/9.0

Req 35	Security-relevant logging data must be forwarded to a log server immediately after it has been generated.
--------	---

The system's security-relevant logging data must be forwarded to a log server immediately after it has been generated.

For the purposes of the requirement, the log server must meet the following criteria:

- The log server must be dedicated to the storage of logging data. It must not depict any other functions or services that are not related to this task. A permissible exception is the combination of a log server with a SIEM (Security Information and Event Management) solution.
- The log server must be separate from the system. It must not be operated directly on the system as an additional module.
- The log server must be centrally available for the operational environment, to which the system belongs.

To transmit the logging data to the log server, standard protocols specialized for this purpose (e.g. Syslog or SNMPv3) should preferably be used

Motivation: If logging data is stored only locally, it can be manipulated in the event of a successful compromise of the system by an attacker in order to obfuscate their attack and actions taken on the system. For this reason, the forwarding of logging data to a log server must take place very soon after the occurrence of a security-relevant event in the source system. This means that the logging data generated is separated from the source system and is available for analysis at any time, regardless of the source system state. The use of a central log server for an operational environment ensures that the logs of all IT/NT Systems in this operational environment can be accessed quickly and uniformly and that the access itself can be optimally integrated into the incident processes of the environment.

For this requirement the following threats are relevant:

- Unauthorized modification of data
- Disruption of availability
- Denial of executed activities
- Unnoticeable feasible attacks

ID: 3.01-35/9.0

Req 36	For security-relevant logging data that is forwarded to the log server, compliance with the applicable retention and deletion periods must be ensured.
--------	--

The following basic rules must be taken into account:

- security-related logging data must be retained for a period of 90 days on the log server.
- after 90 days, stored logging data must be deleted immediately on the log server.

Deviances

Different retention periods and deletion periods may exist due to legal or regulatory requirements (especially in connection with personal data) or may be defined by contractual agreements. In these cases, the applicable periods must be agreed individually with a Project Security Manager (PSM) / Data Privacy Advisor (DSB) or are specified by them.

Log server under the responsibility of a third party

If the selected log server is not within the same operational responsibility as the source system of the logging data, it must be ensured that the responsible operator of the log server is aware of the valid parameters for the logging data to be received and that they are adhered to in accordance with the regulations mentioned here.

Motivation: Logging data is an immensely important IT security tool for preventing, detecting and clearing up system faults, security and data privacy incidents. On the other hand, the recording of logging data, like any other data processing, is also subject to legal and regulatory requirements. Accordingly, guidelines must be adhered to that reconcile the two.

Implementation example: Taking into account the current legal situation and applicable data privacy regulations, the following deletion periods for forwarded security-relevant logging data from an exemplary telecommunications system are implemented on the separate log server:

- Standard System Logs: Deletion after 90 days at the latest
- Logging of public IP addresses: Deletion (or anonymization) after 7 days at the latest
- Logging of the assignment of dynamic public IP addresses by the telecommunication solution: Deletion after 7

days at the latest

- Logging of non-billing-relevant call detail records: Deletion after 7 days at the latest
- Logging of the content of e-mail and SMS: Deletion after 24 hours at the latest
- Logging of the domain queries handled by the DNS server of the telecommunications solution: Deletion after 24 hours at the latest

For this requirement the following threats are relevant:

- Unauthorized access or tapping of data
- Denial of executed activities
- Unnoticeable feasible attacks

ID: 3.01-36/9.0

Req 37	Security-relevant logging data must be provided to a security incident detection and response solution in near-real-time.
--------	---

An acceptable detection and response solution for the purposes of this requirement must meet the following criteria:

- There is a continuous analysis and correlation of the logging data provided by connected IT/NT Systems.
- The identification of threatening activities and ongoing attacks takes place on the basis of a continuously updated detection pattern database
- Integrated alerting processes trigger the initiation of countermeasures in the event of a find.
- Security experts are actively involved in the operation of the solution as well as the evaluation of alarms

Such a solution is not the same as a log server in terms of its function. Accordingly, the provision to a detection and response solution must always be in addition to an existing forwarding to a log server. An exception exists if the log server used provides the logging data to a detection and response solution immediately after receipt. In this case, it is sufficient for the system to forward the logging data only to the log server.

The logging data must be provided to the detection and response solution within 5 minutes ("near-real-time") from the time it is generated in the system.

The specific technical interfaces and protocols to be supported for deployment are determined by the detection and response solution available in the system's operating environment as well as the technical basis of the system itself.

Motivation: IT/NT systems are constantly exposed to security threats and attacks. By means of a central analysis and the correlation of recorded system events of entire IT landscapes in almost real time, current threats and ongoing attacks can be detected even beyond system boundaries and appropriate countermeasures can be initiated promptly. The prompt delivery of logging data to the detection and response solution is essential for its effectiveness.

Implementation example: **Example 1:** A system based on a classic server collects security-relevant logging data in the operating system's local syslog or Windows Event Log. Logging data is continuously forwarded to a Security Information and Event Management (SIEM) for further analysis, intrusion detection, and alarm handling.

Example 2: An Endpoint Detection and Response (EDR) agent is running on a system. The EDR agent continuously forwards the security-related logging data it collects on the system to the central backend of the EDR solution, where further analysis, intrusion detection, and alarm handling takes place

Example 3: Network components provide events to a SIEM that make it possible to detect suspicious communication behavior such as network scans, unauthorized data flows between network segments, or active connections or connection attempts to suspicious destinations on the Internet. In particular, gateway routers from the CNDTAG provide "Netflow" logs, while forward proxies report access to domains and/or IP addresses and firewalls log blocked connection attempts.

Example 4: Systems designed on the basis of cloud-native functions generate logging data directly from these functions using the cloud's own logging and monitoring infrastructure. In such a system, the generation of security-relevant logging data is specifically activated in the corresponding configuration options of the cloud-native functions used.

The resulting logging data is continuously transferred to a SIEM (Security Information and Event Management) for further analysis, intrusion detection and alarm handling. Some cloud providers already provide some attack pattern detection in the cloud's own monitoring infrastructure and generate specific cloud alerts from it. These are also transferred to the SIEM.

Example 5: A system is based on a SaaS ("Software as a Service") application from a cloud provider. Typically, in the SaaS model, the cloud provider is responsible for all operational aspects of the application, including security management. The cloud customer itself has no influence on the underlying technical infrastructure or the operation of the application. Based on contractual agreements, recognized certificates, and documented self-disclosures from the cloud provider, it is formally verified that the cloud provider has implemented a security incident detection and response solution. Some cloud providers also generate dedicated logging data about security-relevant activities within the separate customer tenants of their SaaS application (e.g. changes to configuration settings, permission assignments or user accounts within the tenants) and make them available to cloud customers. These so-called audit logs are continuously transferred to a SIEM (Security Information and Event Management) for further analysis, attack detection and alarm handling.

For this requirement the following threats are relevant:

- Unnoticeable feasible attacks

ID: 3.01-37/9.0