Security requirement

# Microservice (Container based Web Service)

Deutsche Telekom Group

| | |
|---|---|
| Version | 6.0 |
| Date | Jul 1, 2024 |
| Status | In work |

# Publication Details

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

Reuterstrasse 65, 53315 Bonn
Germany

| File name | Document number | Document type |
|---|---|---|
| | 3.84 | Security requirement |
| **Version** | **State** | **Status** |
| 6.0 | Jul 1, 2024 | In work |
| **Contact** | **Validity** | **Released by** |
| Telekom Security<br>psa.telekom.de | Jul 1, 2024 - Jun 30, 2029 | Stefan Pütz, Leiter SEC-T-TST |

Summary
This requirement document replaces the use case container without GUI. The term microservice is typically used to mean a software component that provides an application programming interface (API) in the form of a REST API as a docker container. The microservice only interacts with other services or systems. If your microservice also provides parts of a web application for people, please use requirement document 3.85 "Microservice (container-based web application)". The document summarizes all requirements relevant for this microservice. This makes it possible to describe the microservice with exactly one SoC (Statement of Compliance) in an SDSK (standardized data protection and security concept).

Copyright © 2024 by Deutsche Telekom AG.
All rights reserved.

# Table of Contents


| | | |
|---|---|---|
| 1. | Introduction | 4 |
| 2. | Container | 5 |
| 3. | Application Server (Code Runtime Environment) | 7 |
| 4. | Web-Service | 12 |
| 4.1. | Web-Service: General requirements | 12 |
| 4.2. | Web-Service: Authentication and Authorization | 14 |
| 4.3. | Web-Service: Validation | 16 |
| 4.4. | Web-Service: Payload Signatures | 17 |
| 4.5. | Web-Service: TLS (Transport Layer Security) | 17 |
| 4.6. | Web-Service: REST | 22 |
| 4.6.1. | API Provider specific Requirements | 26 |
| 4.6.2. | API-Consumer specific Requirements | 28 |
| 4.6.3. | Special Case: API-Provider for Human to Machine (H2M) Interfaces | 29 |
| 5. | Logging | 31 |

# 1. Introduction

This requirement document replaces the use case container without GUI. The term microservice is typically used to mean a software component that provides an application programming interface (API) in the form of a REST API as a docker container. The microservice only interacts with other services or systems. If your microservice also provides parts of a web application for people, please use requirement document 3.85 "Microservice (container-based web application)". The document summarizes all requirements relevant for this microservice. This makes it possible to describe the microservice with exactly one SoC (Statement of Compliance) in an SDSK (standardized data protection and security concept).

# 2. Container

| 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 a 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: 3.64-1/4.0

| Req 2 | 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.

*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

ID: 3.64-2/4.0

| Req 3 | Containers must be scanned for vulnerabilities. |
|---|---|

Images must be scanned within your registry and during runtime for known vulnerabilities.

*Motivation: Find known vulnerabilities within containers.*

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: 3.64-5/4.0

| Req 4 | 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.

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

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: 3.64-6/4.0

# 3. Application Server (Code Runtime Environment)

| Req 5 | 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: 3.01-3/8.0

| Req 6 | 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: 3.01-2/8.0

| Req 7 | 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: 3.01-10/8.0

| Req 8 | Sample applications must be deleted. |
|---|---|

*Motivation: Sample application could contain vulnerabilities and provide points of attack.*

For this requirement the following threats are relevant:
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.84-8/6.0

| Req 9 | The system must be protected against overload situations. |
|---|---|

A system must have protective mechanisms that prevent overload situations as far as possible.

In particular, a partial or complete impairment of the availability of the system must be avoided.

Examples of possible protective measures are:

- Limiting the amount of memory (RAM) available per application
- Limiting the maximum sessions of a web application
- Limiting the maximum size of a dataset
- Limiting CPU resources per process
- Prioritizing processes
- Limiting the number or size of transactions by a user or from an IP address over time

Note:
A system can usually not protect itself against network-based attacks with extremely high data or packet rates, the so-called "Distributed Denial of Service" (DDoS) attacks. To defend against DDoS attacks, an upstream solution in the network layer is required.

*Motivation: Attackers can try to use up the resources of a system with targeted resource-intensive or large-volume requests, so that the system can no longer fulfill its regular tasks or intended task volumes and the availability of the services offered is effectively disrupted. Limiting the maximum resources that can be used per request made to the system is a fundamental measure to reduce the impact of such denial-of-service (DoS) attacks.*

For this requirement the following threats are relevant:
- Unauthorized use of services or resources
- Disruption of availability

ID: 3.01-12/8.0

| Req 10 | In overload situations, the system must behave in a predictable manner. |
|---|---|

Even comprehensive native protections may not be able to prevent a system from becoming overloaded in extreme situations.

It must therefore be ensured that, in overload situations, the system does not switch to a state that overrides security-relevant functions or properties of the system. Performance losses (e.g. the reduction of the throughput of legitimate network packets or the number of answered server requests per period) are usually unavoidable in overload situations, but the regular functional behavior of the system must be fundamentally preserved.

In extreme cases, this can mean that a controlled shutdown of the system is more acceptable than continued operation in the event of uncontrolled failure of the security functions and thus the loss of system protection.

*Motivation: By means of a denial-of-service attack, an attacker can try to overload a system in a targeted manner. If such a system then reacts unpredictably or fails its regular behavior, especially with regard to its security functions, this can open up an extended attack surface for the attacker on functions and data of the system and potentially endanger other linked systems.*

Implementation example: A firewall that discards its filter rules in overload situations and forwards all packets without checking would not meet the requirement. In this case, blocking all packets by shutting down the firewall would be more acceptable than failing their regular task of protecting downstream systems.

For this requirement the following threats are relevant:
- Unauthorized use of services or resources
- Disruption of availability
- Unnoticeable feasible attacks

ID: 3.01-13/8.0

| Req 11 | Use and access of functions and information with a need of protection must not be possible without successful authentication and authorization. |
|---|---|

The usage of a systems functions or access of data classified as internal or confidential must only be possible following unambiguous user identification and successful authentication on basis of the user name and at least one authentication attribute. Excepted from this are functions for public use such as those for a Web server on the Internet, via which information is made available to the public. Examples for functions which require a prior authentication are network services (like SSH, SFTP, Web services), local access via a management console, local usage of operating systems and applications. The following examples are possibilities that could be used for authentication.

- Query user name and password
- Use of cryptographic keys and certificates (e.g. as Smartcard)

This requirement must also be applied to accounts that are only used for communication between systems (M2M).

*Motivation: The authentication is necessary to doubtless identify a user because the allocated authorization, and therefore the access on data and services of the system depends on that.*

For this requirement the following threats are relevant:
- Unauthorized access to the system
- Unauthorized use of services or resources
- Denial of executed activities

ID: 3.84-11/6.0

| Req 12 | Access to the application server must be logged. |
|---|---|

The application server log must contain the following information:

- Access timestamp
- Source (IP address)
- Account (if known)
- URL
- http status code of application server response

Logging must be done considering the currently valid legal, wage and company regulations. This regulations state among others that logging of events can be done only earmarked. Logging of events for doing a work control of employees is not allowed.

*Motivation: For the analysis of security incidents it is very important to have basic information on how the attack has been carried out.*

For this requirement the following threats are relevant:
- Unnoticeable feasible attacks

ID: 3.84-12/6.0

# 4. Web-Service

## 4.1. Web-Service: General requirements

| | |
|---|---|
| Req 13 | Basic security functionality for externally accessible web services must be implemented in the DMZ and be protected with a web service gateway depending on the criticality. |

"External" in this context means outside one's own trust boundary here. If the service is offered or used on the Internet, the DMZ is necessary (mandatory). In the case of internal solutions, the degree of freedom must be weighed up over the intersection of the trust zones. From a zero trust perspective, the trust boundary is the system boundary. Depending on the situation, it is possible to extend them to the Kubernetes cluster level, for example.

A DMZ is a zone where all external traffic is terminated.
Databases and applications are located in the MZ, therefore in a zone where no direct external access is possible.

The following functions must be realized in the access and validation tier (DMZ):

- authentication and authorization for the access (i.e. is the consumer of the web service allowed for access)
- basic validation based on the given description such as WSDL, JSON schema or XSD
- pre- and postprocessing tasks (such as terminating an TLS encryption or security relevant logging)
- termination of tcp connections and preventing all direct access to web service end points in the MZ
- limit the maximum connections

To ensure these functionalities a web service gateway should always be used.

*Motivation: All measures must be taken in the DMZ to protect the services in the MZ against attacks.*

Implementation example: Using TARDIS with Stargate for DTAG's internal zones and Spacegate for the Internet is a recommended solution. Historically, SOA-BP is an equivalent solution to B2B Gatway.

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

ID: 3.02-3/8.0

| | |
|---|---|
| Req 14 | If data requiring special protection is processed in a web service, this data must be individually protected by using end-to-end mechanisms at the application level (end-to-end), such as encryption. |

Since transport encryptions are already terminated on outer layers, as well as possible payload logging and "hop by hop" communication, a pure transport encryption in most cases does not offer sufficient protection for particularly confidential data.

Examples of data requiring special protection are Medical data, Criminal records, Bank details of a person, Quarterly figures before publication and Draft contracts with high financial volume.

*Motivation: Due to the special need for protection, confidentiality protection is required for certain data even if they are encrypted for transport or transmitted via secure networks.*

Implementation example: It is recommended to use end-to-end encryption solutions such as JSON Web Encryption (JWE, defined by RFC 7516). Systems such as T-PAM Conjur, Thales Key Manager or Vault solutions of the respective business unit are available for key management.

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

ID: 3.02-4/8.0

| Req 15 | A web service must be protected from manipulation / replay attacks while on the unprotected communication path. |
|---|---|

This can be accomplished through the use of payload-signatures on the application layer, timestamps / session IDs / tokens or TLS on the transport layer, or other appropriate mechanisms (e.g., VPN).

*Motivation: Non-repudiation mechanisms are assumed to become more and more accepted in legal proceedings as well as for accounting and auditing purposes.*
*Therefore, it is highly desirable to not rely on TLS as ephemeral integrity protection mechanism on the transport layer, but rather to solve this issue more appropriately on the application layer with durable Payload signatures.*
*Due to the different time scales and goals pursued with Payload signatures, as compared to TLS integrity protection, the use of a special purpose public/private key pair for signatures becomes inevitable.*
*Payload signatures are the only mechanism that allows to prove to an external third party (e.g., a legal entity such as a court) the fact that a message was received from a particular entity.*
*The timestamp and/or message identifier can be used to counter replay attacks. The requirement to sign the entire message / body is important to counter some sophisticated attacks on the payload signature.*

Implementation example: **XML-based messages:**
XML-Signature (XML-Sig, also XML-DSig, XMLDsig) is a specification for using existing digital signatures in XML notation. It is used in many advanced web standards such as SOAP, SAML or the German online banking system FinTS.

**JSON-based services:**
JSON Web Signature (JWS, RFC 7515).

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
- Unnoticeable feasible attacks

ID: 3.02-7/8.0

| Req 16 | Only the minimal set web services must be made available to the public. |
|---|---|

In particular, when exposing an internal web service to external partners or when extending a legacy application with a web service frontend, care must be taken to only expose required Web Services.

*Motivation: By offering only the minimal required set of web services, the attack window can be reduced.*

Implementation example: On internet web service gateways, only those web services may be made available that are required.
The same applies, for example, to the outbound connections of a container namespace.

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: 3.02-8/8.0

| Req 17 | The web service must be robust against overload situations. |
|---|---|

A web service must provide security measures to deal with overload situations. In particular, partial or complete impairment of http server availability must be avoided. Potential protective measures include:

- Restricting the maximum number of HTTP sessions per IP address
- Defining the maximum size of a HTTP request
- Defining a timeout for HTTP requests

Restrictions must be implemented in consideration of the application to be protected and its characteristics. The following values may be uses as a guideline:
If the http server will not be used for uploads:

- Maximum number of HTTP sessions per IP address: 50
- Maximum size of a HTTP request: 20000 bytes
- Timeout for HTTP requests: 30 seconds

If the http server may also be used for uploads:

- Maximum number of HTTP sessions per IP address: 50
- Maximum size of a HTTP request: 10000000 bytes or, if known, maximum size of expected upload
- Timeout for HTTP requests: 60 seconds or, if known, time to complete maximum upload

*Motivation: Attackers often try to bring a web server into an overload situation by using denial-of-service (DoS) attacks. If such an attack is successfull the http server's availability or integrity may be impaired.*

For this requirement the following threats are relevant:
- Disruption of availability

For this requirement the following warranty objectives are relevant:
- Integrity
- Availability

ID: 3.84-17/6.0

## 4.2. Web-Service: Authentication and Authorization

| Req 18 | Every provider and consuments of a web service must authenticated and authorise each other when transmitting data requiring protection. |
|---|---|

The Provider of a web service must ensure that the consumer is auhorised to access this web service and to receive this data or use these functions.
The consumer must ensure that he is interacting with the correct supplier.

Authorization checks must be considered in every function. It is not allowed to rely on a special trust status due to the authentication of a device at network level or connections from e.g. the same network or namespace.

The authorization check must ensure that the consumer only receives the data he is allowed to receive.

Explanation: "data requiring protection" means data that is not "public", i.e. classified as "internal" or "confidential". Internal/confidential is the common situation within DTAG.

*Motivation: In order to prevent unpermitted usage of resources or output of data, a proper authentication and authorization is necessary.*
*Access to a weather web service that does not require protection, does not require authentication or authorization.*

Implementation example: To grant general authorizations for the access to a web service a certificate-based authentic-

ation is suitable.
To grant an authorization at data field level, JW tokens with individual "claims" are suitable, for example, in combination with filters in order to issue only those data fields that are authorized for the consumer.

The JW token or certificate must be verified by each function / container / microservice.

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: 3.02-9/8.0

| | |
|---|---|
| Req 19 | The mechanism to authenticate and authorize must rely on strong cryptographic algorithms / frameworks. |

Strong cryptographic frameworks/algorithms in this context are e.g. XML signature or TLS client certificates with adequate algorithms like SHA3- / SHA2 hashes and ECDSA / RSA signatures.
Additional examples are JW-, oAuth- and STS-tokens if transferred over tls.

*Motivation: Weak algorithms can be broken by attackers and identities can be faked.*

Implementation example: Certificates according to the certificate requirements for HTTPS:

JW Tokens:
- Hash procedure SHA 256 or higher
- no "none" algorithms ({"alg":"none"})
- ES, RS prefer algorithms over HS
- as short as possible lifetimes

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: 3.02-10/8.0

| | |
|---|---|
| Req 20 | If the authentication is based on tokens, these must be generated by the web service gateway or an approved M2M IDP. |

Using a (federated) centralized IDP (Identity Provider) as the only trusted identity provider is an essential part of the concept of an API-Security.

*Motivation: API-Security@DTAG - Princip 1: Authentication, Authorization, Identification and Access Control (Identiy Management)*
*Implementing strong and adaptive authentication mechanisms is paramount. Utilize industry-standard protocols such as OAuth 2.0 to validate the identity of M2M and human entities attempting to access APIs. The chosen mechanism should align with the required protection level of the data and the criticality of the API in our system architecture. Static API-Tokens should never be an option for internal services.*
*Coupled with authentication, robust authorization controls are essential. Provide the possibility to define and enforce fine-grained access controls to ensure that authenticated entities only access the resources and functionalities pertinent to their roles and responsibilities. This requires a meticulous mapping of user roles to specific API permissions and*

*a harmonized control plane.*
*Efficient management of short living tokens is critical for secure (DTAG-internal) API interactions. Define token lifetimes based on the sensitivity of the data and business requirements. Implement mechanisms for token revocation and renewal to maintain control over access privileges.*

Implementation example: TARDIS with its combination of gateway and IRIS.

ID: 3.02-11/8.0

## 4.3. Web-Service: Validation

| | |
|---|---|
| Req 21 | All Web Service requests and responses must be validated by the Web Service provider and consumers against a detailed specification. |

Each dataset submitted through a Web Service must match the expected data elements, expected length and/or expected range and whenever appropriate a formal specification describing the acceptable data.

If the expected data fields are not always definable, only expected data fields may be processed.

*Motivation: The detailed specification (including regular definitions) allows a much better description of the data that is expected as data elements in the web service.*

Implementation example: A complete web service description can be achieved using e.g. WSDL 2.0 or Swagger.

An example for data format descriptions are JSON schema or XML schema definitions.

An example for a formal definition are regular expressions, e.g. “([A-Za-z0-9]+[-._+&#%/=~])*[A-Za-z0-9]+@([-A-Za-z0-9]+[.])+[A-Zaz]{ 2,6}” to specify a valid e-mail address.
The base type "int" should be replaced by an application specific subtype which exactly defines the needed integer range.
Equivalently the base type "string" must be replaced by an application specific subtype which limits the string's length through "maxLength" and "minLength".

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

ID: 3.02-12/8.0

| | |
|---|---|
| Req 22 | If the web service does not contain a formal definition of the input data, black / white listings must be used to prevent illegal characters from being accepted. |

Data that is not expected from the following parts of the application may have unexpected and undesired effects.
If the input data in the DMZ is properly sanitized, such attacks are made considerably more difficult.

*Motivation: Black / whitelisting can effectively protect against certain types of attacks, such as SQL-, LDAP-, XML-, XPath-, XQuery-, code-, command injection.*
*The use of whitelisting is preferable to blacklisting because blacklists tend to become obsolete over time.*

Implementation example: For web services that are run via a web server, the open source tool mod_security is a good solution.
The OWASP offers various cheatsheets for the solution:
https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html

For this requirement the following threats are relevant:

- Unauthorized access or tapping of data
- Unauthorized modification of data
- Unauthorized use of services or resources
- Disruption of availability

ID: 3.02-13/8.0

## 4.4. Web-Service: Payload Signatures

In special use cases it might be necassary to protect the content (payload) of a request or a response with a cryptographic signature to deny changes or the deniability.

| | |
|---|---|
| Req 23 | If a web service request is protected with signatures, the signature data must not be removed from the request before it is passed on. |

The signature must be validated over the entire transmission path to guarantee the integrity and authenticity of the web service requests / responses.

*Motivation: If this signature is removed, the integrity and authenticity of the web service is no longer guaranteed.*

For this requirement the following threats are relevant:
- Unauthorized modification of data
- Unauthorized use of services or resources
- Denial of executed activities

ID: 3.02-14/8.0

## 4.5. Web-Service: TLS (Transport Layer Security)

| | |
|---|---|
| Req 24 | TLS version 1.2 or 1.3 must be used. |

User roles: Operation, Development, Integration

TLS (Transport Layer Security) is a protocol for the secure transmission of information over TCP/IP based connections and is the successor of SSL (Secure Socket Layer). TLS ensures the confidentiality, integrity and authenticity of the information or the communication partners.

TLS in version 1.2 [RFC 5246] and version 1.3 [RFC 8446] provides cipher suites with Authenticated Encryption Associated Data (AEAD). AEAD ensures the confidentiality as well as the integrity and authenticity of the transmitted information.

References:
[RFC 5246] T. Dierks, E. Rescorla: RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2, 2008
[RFC 8446] E. Rescorla: RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3, 2018

*Motivation: The current version of TLS fixes previous known security vulnerabilities and attack surfaces on the TLS protocol handshake.*

Implementation example: OpenSSL> protocol = tlsv1_3

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: 3.50-40/7.1

| Req 25 | Only Perfect Forward Secrecy (PFS) TLS-cipher suites must be used according to the tables below. |
|---|---|

User roles: Operation, Development, Integration

Cipher suites specify the cryptographic methods of a connection.

Perfect Forward Secrecy (short PFS, also Forward Secrecy) means that transmitted information cannot be decrypted afterwards, even if the long-term key of the communication partners is known.

In TLS v1.2 cipher suites are defined as follows: *TLS_AKE_WITH_Enc_Hash.*
Following, the meaning of the individual components is explained:

- *AKE (Authenticated Key Exchange)*: Key agreement mechanism with authentication for the handshake protocol.
- *Enc (Encryption)*: Encryption algorithm with mode of operation for the record protocol.
- *Hash*: Hash algorithm for HMAC used for key derivation. If *Enc* is not an AEAD encryption mechanism, HMAC is also used for integrity protection.

The following table lists the allowed cipher suites with PFS in TLS v1.2 as well as the reference specifications. The design philosophy of TLS v1.2 was followed, which is why the table contains only AEAD constructions.
Allowed cipher suites with PFS in TLS v1.2:

| Priority | Cipher Suite | Reference specification |
|---|---|---|
| HIGH | TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 | RFC 5289 |
| HIGH | TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 | RFC 5289 |
| LOW | TLS_DHE_DSS_WITH_AES_256_GCM_SHA384 | RFC 5288 |
| LOW | TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 | RFC 5288 |
| HIGH | TLS_ECDHE_ECDSA_WITH_CHACHA20POLY1305_SHA256 | RFC 7905 |
| HIGH | TLS_ECDHE_RSA_WITH_CHACHA20POLY1305_SHA256 | RFC 7905 |
| LOW | TLS_DHE_RSA_WITH_CHACHA20POLY1305_SHA256 | RFC 7905 |
| HIGH | TLS_ECDHE_ECDSA_WITH_AES_256_CCM | RFC 7251 |
| LOW | TLS_DHE_RSA_WITH_AES_256_CCM | RFC 6655 |
| HIGH | TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 | RFC 5289 |
| HIGH | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | RFC 5289 |

| | | |
|---|---|---|
| LOW | TLS_DHE_DSS_ WITH_AES_128_GCM_SHA256 | RFC 5288 |
| LOW | TLS_DHE_RSA_ WITH_AES_128_GCM_SHA256 | RFC 5288 |
| HIGH | TLS_ECDHE_ECDSA_WITH_AES_128_CCM | RFC 7251 |
| LOW | TLS_DHE_RSA_ WITH_AES_128_CCM | RFC 6655 |

Furthermore, in legacy systems the cipher suites of the following table are allowed.
Additional cipher suites with PFS in TLS v1.2 with AES-CBC:

| Priority | Cipher Suite | Reference specification |
|---|---|---|
| HIGH | TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 | RFC 5289 |
| HIGH | TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 | RFC 5289 |
| LOW | TLS_DHE_DSS_ WITH_AES_256_CBC_SHA256 | RFC 5246 |
| LOW | TLS_DHE_RSA_ WITH_AES_256_CBC_SHA256 | RFC 5246 |
| HIGH | TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 | RFC 5289 |
| HIGH | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 | RFC 5289 |
| LOW | TLS_DHE_DSS_ WITH_AES_128_CBC_SHA256 | RFC 5246 |
| LOW | TLS_DHE_RSA_ WITH_AES_128_CBC_SHA256 | RFC 5246 |

Remark on the cipher suites for TLS v1.2:
The table entries are sorted by the symmetric encryption mechanism (Enc). For the authenticated key agreement methods (AKE), mechanism based on elliptic curves (ECDHE_ECDSA) are preferred. DHE (discrete logarithm) key establishment ciphers are more vulnerable against DoS (DHeater) than ECDHE, thus ECDHE should be preferred. The "Priority" column defines which cipher suites are preferred, that is cipher suites with a priority of "HIGH" are preferable to those with "LOW".

In TLS v1.3 cipher suites are defined as follows: *TLS_AEAD_Hash.*
Following, the meaning of the individual components is explained:

- AEAD: Authenticated encryption mechanism for the record protocol.
- Hash: Hash algorithm for HMAC and HKDF in the handshake protocol.

The following table lists the allowed cipher suites with PFS in TLS v1.3 as well as the reference specifications.
Allowed cipher suites with PFS in TLS v1.3:

| Cipher suites | Reference specification |
|---|---|

| | |
|---|---|
| TLS_AES_256_GCM_SHA384 | RFC 8446 |
| TLS_CHACHA20_POLY1305_SHA256 | RFC 8446 |
| TLS_AES_128_GCM_SHA256 | RFC 8446 |
| TLS_AES_128_CCM_SHA256 | RFC 8446 |

Any cipher suites specified in the future that correspond to the requirements defined in this document can be used as well. For example, this applies for cipher suites that use a hash function from the SHA-3 family.

References:
[1] Bundesamt für Sicherheit in der Informationstechnik, TR-02102-2, version 2023-01
[2] https://ciphersuite.info/cs/?security=secure&sort=asc (last view: February 29th, 2024)
[3] https://ciphersuite.info/cs/?singlepage=true&security=recommended# (last view: February 29th, 2024)

*Motivation: The usage of modern cipher suites with Perfect Forward Secrecy protects the transport security in TLS.*

Implementation example: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

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: 3.50-41/7.1

| Req 26 | For TLS, Diffie Hellman groups according to the table below must be used. |
|---|---|

User roles: Operation, Development, Integration

The Diffie Hellman groups is used for key exchange with Perfect Forward Secrecy (PFS). Generally, a distinction is made between elliptic curve groups and finite field groups (mod p).

The following table contains the allowed Diffie Hellman groups.
Allowed Diffie Helman groups for use in TLS:

| Diffie Hellman group | IANA-No. | Referenzspezifikation |
|---|---|---|
| brainpoolP512r1 | 33 | RFC 7027 |
| secp521r1 | 25 | RFC 8422 |
| x448 | 30 | RFC 8422 |
| brainpoolP384r1 | 27 | RFC 7027 |
| secp384r1 | 24 | RFC 8422 |
| brainpoolP256r1 | 26 | RFC 7027 |
| secp256r1 | 23 | RFC 8422 |
| x25519 | 29 | RFC 8422 |
| ffdhe4096 | 258 | RFC 7919 |
| ffdhe3072 | 257 | RFC 7919 |

Remark on x448 and x25519:

x448 und x25519 are not (explicitly) recommended by BSI, but no weaknesses are known so far, and therefore they are to be classified as secure.

Remark on group 256:
Diffie Hellman group 256 (IANA-No.256) has a key length of 2048 bit [1] [2] and may only be used in legacy systems until the end of the **year 2025 [2]**. The group must be substituted by a stronger method (according to the enumeration above).

Remark on groups 256, 258 and 257:
Those groups are more vulnerable against the DHeater (DoS) attacks on server side than elliptic curve DH groups. Therefore, the brainpool and NIST (secp) groups should be preferred.

References:
[1] Bundesamt für Sicherheit in der Informationstechnik, TR-02102-2, Version 2023-01
[2] SOG-IS Crypto Evaluation Scheme: Agreed Cryptographic Mechanisms, v1.3, February 2023

*Motivation: Standardized Diffie Hellman groups use secure parameters and speed up the key exchange.*

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: 3.50-42/7.1

| | |
|---|---|
| Req 27 | For TLS, digital certificates from a certification authority with a sufficient key length and limited validity must be used. |

User roles: Operation, Development, Integration

In TLS, digital certificates are used during the TLS handshake. TLS servers must use a TLS certificate. Clients need a certificate if mutual authentication is required.

TLS certificates for web servers that are accessible from the internet must be issued by public certification authorities that are classified as trustworthy by browsers and operating systems. These can be ordered, for example, via the service “TeleSec ServerPass” (please refer https://www.telesec.de/de/serverpass ).

Regarding key lengths, validity and further configuration options, the Certificate Policy of the Certification Authority must be considered.

For web servers that are used exclusively for internal applications and are not accessible from the internet, digital certificates from a private (internal) Certification Authority can be used.

The minimal requirements according to the following table must be considered for each type of TLS certificates, this means also for client certificates:

| Algorithm family | Key length | Hash algorithm |
|---|---|---|
| Elliptic Curve | 250 bit | SHA-3, SHA-2 with an output length 256 bit |
| Digital Signature Algorithm (DSA) | 3000 bit | SHA-3, SHA-2 with an output length 256 bit |

| | | |
|---|---|---|
| RSA | 3000 bit | SHA-3, SHA-2 with an output length 256 bit |

Remarks on DSA and RSA certificates:
For DSA and RSA, key lengths smaller than 3000 bits may only be used in legacy systems [BSI TR 02102-1] until **end of the year 2025** [2] and should be substituted at the next opportunity. Because of the better performance, elliptic curve (EC-DSA) certificates shall be preferred (if supported and technically doable).
RSA-PKCS#1 v1.5 may only be used in legacy systems and should be (if feasible) substituted at the earliest opportunity [BSI TR 02102-1].

Restrictions on SHA-224/SHA-3-224:
SHA-224/SHA-3-224 may only be used in legacy systems and must be substituted by a stronger hash algorithm with an output length of at least 256 bits at the next opportunity.

The validity period of public TLS server certificates (issued by a certification authority, which issues certificates according to the specifications of the [CA/Browser Forum]) must not exceed 397 days. For other, internal TLS certificates, a validity period of 3 years should not be exceeded.

References:
[BSI TR 02102-1] Bundesamt für Sicherheit in der Informationstechnik: Cryptographic Mechanisms: Recommendations and Key Lengths, TR-02102-1, Version 2023-01
[1] Bundesamt für Sicherheit in der Informationstechnik, TR-02102-2, Version 2023-01
[2] SOG-IS Crypto Evaluation Scheme: Agreed Cryptographic Mechanisms, v1.3, February 2023
[CA/Browser Forum] https://cabforum.org/baseline-requirements-documents/ (last view: February 29th, 2024)

*Motivation: Digital certificates form the basis of the authentication and build up trust. Without sufficiently strong authentication, man-in-the-middle attacks are possible.*

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: 3.50-43/7.1

## 4.6. Web-Service: REST

| | |
|---|---|
| Req 28 | Responses from REST web services must be based on HTTP status codes. |

The HTTP protocol provides standardized codes for each status of a connection.

Further additional Information such as received data (payload) or technical details about the error may not be output but must be stored with an unique error reference in the log.
The error reference may be output.

| Kategorie | Beschreibung |
|---|---|
| 1xx: Informational | Communicates transfer protocol-level information |
| 2xx: Success | Indicates that the client's request was accepted successfully |

| | |
|---|---|
| 3xx: Redirection | Indicates that the client must take some additional action in order to complete their request |
| 4xx: Client Error | Indicates that the client seems to have erred |
| 5xx: Server Error | Indicates that the server seems to have erred |

*Motivation: The HTTP protocol already provides standardized codes for each status of a connection.*
*By using these status codes, informations requiring protection, such as error and system messages, are prevented from being included in the output.*
*The consumer of the Web service can also use these standardized codes to generically determine the cause of the error.*
*Reflecting the received data (payload) can lead to security gaps (especially in web applications).*

Implementation example: An example of a few HTTP status codes and their meaning:

| HTTP status code | Message | Description |
|---|---|---|
| 200 | OK | The request was processed successfully |
| 400 | Bad Request | The request is malformed |
| 406 | Not Acceptable | The content type requestst by the client in the accept-header is not offered by the web service |
| 413 | Payload to large | The request is larger than the server is willing or able to process |

All HTTP status codes and their usage are specified in the RFC7231.

For this requirement the following threats are relevant:
- Unauthorized access or tapping of data
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.02-19/8.0

Req 29 Functionalities of REST web services must be based on HTTP methods.

The HTTP protocol provides standardized methods that are required for all functionalities.
By using these methods a uniform and transparent authorization control is enabled, which is supported by web servers, reverse proxies and web service gateways.

*Motivation: By using standardized methods, upstream layers such as web servers, reverse proxies and web service gateways are able to perform a basic validation of the requests.*
*This also ensures a common understanding for consumers of how a web service should be used.*

Implementation example: Examples of the most common HTTP methods:

| HTTP method | Description |
|---|---|
| GET | The GET method requests transfer the current selected resource |
| HEAD | The HEAD method is identical to GET, except that the server does not send the HTTP BODY of the response |

| | |
|---|---|
| POST | The POST method requests the processing of the transmitted data |
| PUT | The PUT method requests that the state of the target resource be created or replaced |
| DELETE | The DELETE method requests the removal of the target resource |
| PATCH | The PATCH method requests a set of changes described in the request |

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

ID: 3.02-20/8.0

| Req 30 | HTTP methods that are not required must be deactivated. |
|---|---|

The TRACE/TRACK method must not be used by a productive web server. Standard requests to web servers only use GET and POST. If other methods are required, they must be processed securely.

*Motivation: HTTP TRACE could be misused by an attacker. This method allows for debugging and trace analysis of connections between the client and the web server. Other HTTP methods could also be used to obtain information about the server, or they could be directly misused by an attacker.*

For this requirement the following threats are relevant:
- Unauthorized access or tapping of data
- Unauthorized modification of data
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.84-30/6.0

| Req 31 | Information about the server in HTTP headers must be minimized. |
|---|---|

The HTTP header must not include information on the version of the web server and the modules/add-ons used.

*Motivation: Any information about the http server could allow conclusions to be drawn about security vulnerabilities.*

For this requirement the following threats are relevant:
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.84-31/6.0

| Req 32 | Confidential data must not be transferred in the URL. |
|---|---|

Confidential data may only be transferred outside the URL (e.g. HTTP-BODY, HTTP-HEADER).
When using HTTP headers, only fields that are not recorded in log files may be used.

Examples for confidential data are:
- API Keys
- Security Tokens (e.g. oAuth-, JW-Token)
- Passwords

In addition to confidential data from the perspective of information protection, confidentiality also results from the reference of the data to a person. This data must also not be transmitted in the URL.

In the context of Restful API, this leads to conflicts, as a resource name (ID, identity) can be a personal reference.

**Data protection assessment:**

If the URL is used in communication between two IT systems within DTAG, then an URL parameter with personal reference (PII parameter) is acceptable if:

- API is only exposed and managed through TARDIS.
- API is only exposed internally (Stargate).
- Only data up to max. data protection class 2 will be transmitted.
- In the URL, a**maximum of one, i.e. exactly one**personal attribute, is transmitted.

Positive example (one parameter):

- In the search request, only either the MSISDN or the customer number is transmitted in a message.

Negative examples (two parameters):

- In the search request, the combination of MSISDN + customer number would be used.
- Customer number + BKTO (anyway actually DPC 3 at TDG, because credential in the telephone service).
- MSISDN + Customer Surname.

*Motivation: All data contained in the URL appears in log files of e.g. web server, reverse proxy and web service gateway.*
*Log files are often collected at a central location or transferred to SIEM systems for monitoring.*
*In order to make this possible and at the same time protect confidential data, it must not be transmitted in the URL.*

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: 3.02-21/8.0

| Req 33 | Content types must be validated. |
|---|---|

A REST request and response must match the intended content type in the header.
Otherwise this could cause misinterpretation at the consumer / provider side and lead to code-injection / code-execution.

*Motivation: REST web services often allow multiple data formats for request and response.*
*The consumer must specify the data format of the request and the desired data format of the Web service response in the "Content-Type" and "Accept" headers.*
*The provider may only process the request if it supports the data format for the request and the response.*

Implementation example: *Reject requests with unexpected or missing content type headers with HTTP code 415 "Unsupported Media Type".*

ID: 3.02-22/8.0

| Req 34 | HTTP server information in error pages must be deleted. |
|---|---|

Default error pages must be replaced with user-defined error pages. User-defined error pages must not include ver-

sion information about the web server and the modules/addons used. Error messages must not include internal information such as internal server names, error codes, etc.

*Motivation: Any information about the http server could allow conclusions to be drawn about security vulnerabilities.*

Implementation example: Create own error pages without information about the http server product and version.

For this requirement the following threats are relevant:
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.84-34/6.0

## 4.6.1. API Provider specific Requirements

Requirements in this section are only relevant for Providers which offers an API. Systems or Services just consuming an API (i.e. consumers) mark this section as "Not Applicable" (=NA).

| Req 35 | The API-Provider must publish or offer a technical documentation. |
|---|---|

The technical Documentation should be based on public standards e.g. Open API Specification (OAS). For further explanations and information about general API standardization, specifications and documentation refer to: https://developer.telekom.de/docs/src/api_standards_conventions/2_basics or at the TM-Forum: https://www.tmforum.org/oda/open-apis

- Specification standard
- API Name
- PI Description
- API Version
- Responsible Org. Unit
- Contact Email address of the API owner
- Security Scopes (in terms of authorization) for the endpoints
- Customer Segment
- IT Application e.g. ICTO Number
- Version of the API service
- Deprecation indicator if applicable
- Endpoint Specifications
- Data Specifications
- Accepted and provided Media Types
- HTTP-Status Codes
- Error Response Object
- Differentiated Error Cases
- Data privacy class (DPC) or Information protection class

*Motivation: (Technical) Documentation: Comprehensive documentation aids in security audits and reviews, ensuring that best practices are followed throughout the development and integration process. Furthermore, it can be utilized to detect or prohibit hidden-APIs. Exemplary implementation could be OpenAPI Specification (OAS).*

Implementation example: Examples are available here: API-Catalog published by TARDIS: https://developer.telekom.de/catalog or TARDIS Rover handbook: https://developer.telekom.de/docs/src/tardis_customer_handbook/rover/

For this requirement the following threats are relevant:
- Unauthorized modification of data
- Unnoticeable feasible attacks

ID: 3.02-23/8.0

| Req 36 | APIs (M2M) must be exposed via an API gateway acting as a proxy. |
|---|---|

- Direct exposure is not permissible.
- Within the same security context of an application module, direct communication to strictly internal APIs within a trust-zone may happen but are not desirable.

In certain circumstances, an exception to this requirement may be justified.
Valid reasons for an exception are:

- Real-time or low-latency connections are necessary. If latency is critical, it can make sense to connect systems directly.
- Customer implementation: In a scenario where an API is operated for an external customer of the group, it depends on the respective contract with the customer whether the use of a central API gateway is required or optional. However, the use of a central gateway is recommended in any case.
- Data protection legislation may require end-to-end encryption. However, the software cannot be changed by the company or the customer for the purpose of implementing payload encryption, in which case a direct link between the services may be unavoidable.
- Smallest possible Trust Zone: In a scenario where microservices run in the same Kubernetes namespace and are managed by the same small team.

*Motivation:* API-Security@DTAG - Princip 2a: Isolation by Concerns – Zero Trust – Trust Boundary
*By defining trust-zones on different levels (Kubernetes, team, organization, legal-unit, data privacy rules, NatCos, etc.) development of small service APIs can be streamlined, and critical environments can be isolated against human error. By implementing isolation, an API gateway minimizes the attack surface, limiting the impact of a potential security breach to isolated areas and preventing the lateral movement of attackers. By default a new service must always be isolated and communication must be explicitly allowed through whitelisting.*
*Zero Trust is vital for APIs, as it enforces them to act as their own strict trust boundaries within segmented networks. This means that APIs must rigorously authenticate the identity of each requester, beyond just machine-to-machine interactions, scrutinizing additional context such as device health. By doing so, APIs maintain a high-security posture, assuming no inherent trust and mitigating potential internal and external threats. The effort for verifying necessary context parameters should be tailored according to the criticality of the managed data.*

Implementation example: By using TARDIS and its services, the goal can be achieved and implemented can take place. At the same time, TARDIS helps simplify the complexity and management of isolation.

For this requirement the following threats are relevant:
- Unauthorized access to the system
- Unauthorized access or tapping of data
- Unnoticeable feasible attacks

ID: 3.02-24/8.0

| Req 37 | The self-contained authentication token, forwarded by the API-Gateway needs to be verified by the provider. |
|---|---|

Following parameters need to be verified:

- Scope or identity,
- expire timestamp,
- signature and issuer,
- algorithm (not “none”)

*Motivation: The tokens form the anchor of trust and enable the provider to identify and authorize the consumer. Missing or inadequate token verification allows data to leak or misuse resources.*

Implementation example: The TARDIS documentation provides help and notes on this.
Source: https://developer.telekom.de/docs/src/tardis_customer_handbook/StarGate/#to-do-for-api-provider

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: 3.02-25/8.0

| | |
|---|---|
| Req 38 | Payload Encryption: If data privacy or information security requires it, consumers and providers must encrypt the payload end-to-end. |

The provider/consumer must ensure that the API gateways involved do not have access to the contents of the transmission (payload). This may be due to the fact that the data is subject to a certain VS-IT category or a certain data protection class and the gateways are not allowed to process this category or have access to it.
Source regarding VS-IT-Category are the NOR Operation Models: (german only): https://yam-united.telekom.com/pages/it-security-privacy-dt-it/apps/wiki/wiki/list/view/34291ced-1e0f-475b-b785-7aa8fab3f2b8

*Motivation: E2E encryption allows systems with high operating models such as DE4 or higher to be used via transport systems such as TARDIS with a lower operating model. It will also be easier to link systems with different operating models, such as DE4 to DE3, as encryption protects the higher-value data from being accessed by the low-value system.*

ID: 3.02-26/8.0

## 4.6.2. API-Consumer specific Requirements

Requirements in this section are only relevant for Consumers of an API. Systems or Services just providing an API (i.e. Provider) mark this section as "Not Applicable" (=NA).

| | |
|---|---|
| Req 39 | The consumer's identity must be managed by an approved Machine-to-Machine Identity Provider (M2M-IDP) of the API gateway infrastructure and authenticated as a token. |

*Motivation:* **API-Security@DTAG - Princip 1: Authentication, Authorization, Identification and Access Control (Identiy Management)**
*Implementing strong and adaptive authentication mechanisms is paramount. Utilize industry-standard protocols such as OAuth 2.0 to validate the identity of M2M and human entities attempting to access APIs. The chosen mechanism should align with the required protection level of the data and the criticality of the API in our system architecture. Static API-Tokens should never be an option for internal services.*
*Coupled with authentication, robust authorization controls are essential. Provide the possibility to define and enforce fine-grained access controls to ensure that authenticated entities only access the resources and functionalities pertinent to their roles and responsibilities. This requires a meticulous mapping of user roles to specific API permissions and a harmonized control plane.*
*Efficient management of short living tokens is critical for secure (DTAG-internal) API interactions. Define token lifetimes based on the sensitivity of the data and business requirements. Implement mechanisms for token revocation and renewal to maintain control over access privileges.*
*The usage of a (federated) centralized IDP as a single trusted Identity Provider is essential in the concept of an API gateway.*

Implementation example: TARDIS offers all the necessary functions and is also an approved ID provider for M2M identities. In addition, there is the possibility of IDM federation. More information for consumers can be found here: https://developer.telekom.de/docs/src/tardis_customer_handbook/StarGate/#to-do-for-api-consumer

ID: 3.02-27/8.0

| | |
|---|---|
| Req 40 | The consumer must check that he is directly connected to the API-Gateway, for example by checking the TLS certificate. |

The audit can be supported by other mechanisms, such as:

- IP address filtering,
- Security policies in the respective (Kubernetes) environment,

*Motivation: Man-in-the-middle (MiM) attacks pose a major threat to confidentiality and misuse of resources. The fundamental protection of authentication features (tokens) and secrets (credentials) lies in the TLS (Transport Layer Security) level. In order to keep complexity low, double-sided, certificate-based authentication is usually dispensed with. Only the gateway identifies itself to the consumer as an HTTPS entity. As a result, only the client can detect a MitM attack and reject the connection by checking the gateway TLS certificate.*

ID: 3.02-28/8.0

### 4.6.3. Special Case: API-Provider for Human to Machine (H2M) Interfaces

Reference to the Guiding Principles Front End: https://wiki.telekom.de/pages/viewpage.action?pageId=886833714

| | |
|---|---|
| Req 41 | APIs between browser and backend must be terminated by a backend-for-frontend (BFF). |

User facing APIs: Depending on the NatCo or PU the architecture guidelines impose the usage of a Backend for frontend (BFF) as an abstraction layer instead of an API gateway (Explanation of BFF in DTAG).
The BFF encapsulates the web session of the human and separates the backend calls on the machine to machine level.

*Motivation: On the one hand, the separation makes it possible to monitor and terminate the registration and use by humans (web session). At the same time, it ensures that access credentials/tokens are not transferred to an insecure client and that an attacker gains access to the backend systems.*

Implementation example: TARDIS offers an identity broker for the scenario with the "Sirius" service: TARDIS-Sirius ID-Broker

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: 3.02-29/8.0

| | |
|---|---|
| Req 42 | The BFF is not allowed to pass the backend M2M tokens to the user (browser). |

The backend M2M token provides access to the internal services for processing the business processes. If the role and authorization concept is implemented in the BFF, this leads to a large attack surface and the risk of data leaks or misuse of resources.

*Motivation: API Security @ DTAG section 4.4 and 4.5*

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: 3.02-30/8.0

| | |
|---|---|
| Req 43 | The BFF must use the API infrastructure (M2M) as a regular consumer or provider to interact with the backend services. |

The Backend-For-Fronted (BFF) separates the web session to the browser / human from the technical data processing in the backend. On the backend service side, it uses the API gateway infrastructure to interact with API providers or to offer its own APIs.

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
- Unnoticeable feasible attacks

ID: 3.02-31/8.0

# 5. Logging

| | |
|---|---|
| Req 44 | Events must be logged with an exact time stamp and trigger alarms depending on their criticality. The logging of users/user actions and payloads must be coordinated with data protection and data security. |

Events must be given an event identifier according to their event type.

Security relevant events must be specially marked in order to be able to trigger alarms as required.
Suitable event identifiers are, for example „info“, „error“, „warning“, „alert“, „emergency“.These event labels can be used to automatically trigger alarms and facilitate troubleshooting.
It is also possible to control deletion periods and log access authorizations based on these event labels.

*Motivation: To ensure safe operation and error analysis, a logging, monitoring and alarming concept must be created. Safety-relevant events must be immediately forwarded to a suitable company, which can analyse the problem and take countermeasures.*

Implementation example: The event identifier as well as the need for an alarm depends on the need for protection and the interface through which the web service is accessible.

A web service that can be accessed from the Internet is more likely to be subject to validation and authorization errors, so an alarm is not always appropriate. However, if this occurs with a web service that can only be accessed from trusted areas, e.g. from its own namespace, administrative special networks or M2M connections, this can be an indication of an intruder.

Likewise, the need for an alarm may depend on the frequency of the event.
An access attempt to access a resource without appropriate authorization or a validation error may be irrelevant if it occurs once. If this happens frequently, it can be an indication of a targeted attack on the application or the application's functions are not working properly.

The following table gives examples of events, event identifiers, and also the log context to be stored.

| Category | Event | Event Identifier | Log Context |
|---|---|---|---|
| Access | Access to the web service | info | Applications-, Container-ID, Authenticity, IP |
| Validation error | Incorrect data format | error, debug | Applications-, Container-ID, Authenticity, IP, Payload |
| | Incorrect encoding | error, debug | Applications-, Container-ID, Authenticity, IP, Payload |
| | incorrect daten elements | error, debug | Applications-, Container-ID, Authenticity, IP, Payload |
| | validation against formal definition fails | error, debug | Applications-, Container-ID, Authenticity, IP, Payload |
| | Incorrect range of values | error, debug | Applications-, Container-ID, Authenticity, IP, Payload |
| Output error | Datenbase / supplying systems not accessible | error, debug | Applications-, Container-ID, Authenticity, Request-ID, IP |
| | Error in the database records | error, debug | Applications-, Container-ID, Authenticity, Request-ID, IP |

| | | | |
|---|---|---|---|
| | Output incomplete | error, debug | Applications-, Container-ID, Authenticity, Request-ID, IP |
| Authorisation error | Access to resources without proper authorization | warning | Applications-, Container-ID, Authenticity, IP, Payload |
| | Signature validation error for JW Token | alert | Applications-, Container-ID, Authenticity, IP, Payload |
| | Perform actions that do not match the role / ACL | alert | Applications-, Container-ID, Authenticity, IP, Payload |
| Authentication error | Failed | warning | Applications-, Container-ID, IP |
| | Success | info | Applications-, Container-ID, IP |
| Privileged actions | Add / Delete of accounts | trace | Applications-, Container-ID, Authenticity, IP |
| | Change of privileges | trace | Applications-, Container-ID, Authenticity, IP |
| | Exporting data requiring protection | trace | Applications-, Container-ID, Authenticity, IP |
| | Import of daten | trace | Applications-, Container-ID, Authenticity, IP |
| | Change of configuration | trace | Applications-, Container-ID, Authenticity, IP |
| Runtime error | Crash / restart of the application / container | emergency | Applications-, Container-ID, Stacktrace |
| | connectivity problems | emergency | Applications-, Container-ID |
| | performance problems | alert | Applications-, Container-ID |
| | Virus detection e.g. file upload | alert | Applications-, Container-ID, Authenticity, IP |

For this requirement the following threats are relevant:
- Denial of executed activities
- Unnoticeable feasible attacks

ID: 3.02-33/8.0

| | |
|---|---|
| Req 45 | 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: 3.01-34/8.0

| | |
|---|---|
| Req 46 | For security-relevant logging data that is forwarded to the separate 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 separate log server.
- after 90 days, stored logging data must be deleted immediately on the separate 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 separate log server is not within the same operational responsibility as the source system of the loggin 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: 3.01-36/8.0

| | |
|---|---|
| Req 47 | 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.

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:** 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 4:** 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: 3.01-37/8.0