Security requirement

Microservice (Container based Web Service)

Deutsche Telekom Group

Version	5.0
Date	Dec 1, 2023
Status	Released

Publication Details

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

Reuterstrasse 65, 53315 Bonn
Germany

File name	Document number	Document type
	3.84	Security requirement

Version	State	Status
5.0	Dec 1, 2023	Released

Contact	Validity	Released by
Telekom Security psa.telekom.de	Dec 1, 2023 - Nov 30, 2028	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).

Table of Contents

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

For this requirement the following warranty objectives are relevant:

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

For this requirement the following warranty objectives are relevant:

ID: 3.64-2/4.0

Req 3	Unnecessary packages must be avoided.

Containers must only have the essentials needed to run the intended application. An image must only contain a single piece of functionality for an application.

Motivation: This avoids running not needed software within containers to lower the attack surface.

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

For this requirement the following warranty objectives are relevant:

ID: 3.64-3/4.0

Req 4	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

For this requirement the following warranty objectives are relevant:

ID: 3.64-5/4.0

Req 5	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

For this requirement the following warranty objectives are relevant:

ID: 3.64-6/4.0

3. Application Server (Code Runtime Environment)

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.

Suitable 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

For this requirement the following warranty objectives are relevant:

ID: 3.01-2/7.0

Req 7	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

For this requirement the following warranty objectives are relevant:

ID: 3.84-7/5.0

Req 8	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

For this requirement the following warranty objectives are relevant:

ID: 3.84-8/5.0

Req 9	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

For this requirement the following warranty objectives are relevant:

ID: 3.84-9/5.0

4. Web-Service

4.1. Web-Service: General requirements

Req 10	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.

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.

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

For this requirement the following warranty objectives are relevant:

ID: 3.02-3/7.0

Req 11	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 XML 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.

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

For this requirement the following warranty objectives are relevant:

ID: 3.02-4/7.0

Req 12	A web service must be protected from manipulation / replay attacks while on the unprotected communication path.

This can be accomplished through the use of XML-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 XML signatures.
Due to the different time scales and goals pursued with XML signatures, as compared to TLS integrity protection, the use of a special purpose public/private key pair for signatures becomes inevitable.
XML 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 XML message / SOAP Body is important to counter some sophisticated attacks on the XML-signature.

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

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

ID: 3.02-5/7.0

Req 13	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

For this requirement the following warranty objectives are relevant:

ID: 3.02-6/7.0

Req 14	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: 3.84-14/5.0

4.2. Web-Service: Authentication and Authorization

Req 15	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.

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 authentication 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

For this requirement the following warranty objectives are relevant:

ID: 3.02-7/7.0

Req 16	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

For this requirement the following warranty objectives are relevant:

ID: 3.02-8/7.0

4.3. Web-Service: Validation

Req 17	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

For this requirement the following warranty objectives are relevant:

ID: 3.02-9/7.0

Req 18	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.

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

For this requirement the following warranty objectives are relevant:

ID: 3.02-10/7.0

4.4. Web-Service: REST

Req 19	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

For this requirement the following warranty objectives are relevant:

ID: 3.02-11/7.0

Req 20 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

For this requirement the following warranty objectives are relevant:

ID: 3.02-12/7.0

Req 21	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

For this requirement the following warranty objectives are relevant:

ID: 3.84-21/5.0

Req 22	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

For this requirement the following warranty objectives are relevant:

ID: 3.84-22/5.0

Req 23	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

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

For this requirement the following warranty objectives are relevant:

ID: 3.02-13/7.0

Req 24	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: 3.02-14/7.0

Req 25	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 version 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

For this requirement the following warranty objectives are relevant:

ID: 3.84-25/5.0

4.5. 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 26	If a web service request is protected with signatures, the signature data must not be removed from the request by an intermediate processor.

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

For this requirement the following warranty objectives are relevant:

ID: 3.02-15/7.0

4.6. Web-Service: TLS (Transport Layer Security)

Req 27	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

For this requirement the following warranty objectives are relevant:

ID: 3.50-40/7.0

Req 28	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_1 28_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_2 56_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_1 28_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, i.e. 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
[3] https://ciphersuite.info/cs/?singlepage=true&security=recommended#

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

For this requirement the following warranty objectives are relevant:

ID: 3.50-41/7.0

Req 29	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

For this requirement the following warranty objectives are relevant:

ID: 3.50-42/7.0

Req 30	For TLS, digital certificates from an appropriate 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 an appropriate 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/

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

For this requirement the following warranty objectives are relevant:

ID: 3.50-43/7.0

5. Logging

Req 31	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

For this requirement the following warranty objectives are relevant:

ID: 3.02-21/7.0

Req 32	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

For this requirement the following warranty objectives are relevant:

ID: 3.01-34/7.0

Req 33	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

For this requirement the following warranty objectives are relevant:

ID: 3.01-36/7.0

Req 34	The system must provide logging data that is required to detect the system-specific relevant forms of attack in a SIEM.

The forms of attack that are typically to be expected for the present system must be systematically analyzed and identified.
The MITRE Attack Matrix (https://attack.mitre.org) can be used as a structured guide during such an identification.

It must be ensured that the system generates appropriate logging data on events that are or may be related to these identified forms of attack and that can be used to detect an attack that is taking place.

The logging data must be sent to a SIEM immediately after the system event occurs.
SIEM (Security Information & Event Management) solutions collect event log data from various source systems, correlate it and evaluate it automatically in real time in order to detect anomalous activities such as ongoing attacks on IT/NT systems and to be able to initiate alarms or countermeasures.
The immediate receipt of system events is therefore absolutely crucial for the SIEM to fulfill its protective functions.

Note:
The immediate need to connect a system to a SIEM is specifically regulated by the separate "Operation" security re-

quirements catalogs.
If the present system does not fall under this need, the requirement may be answered as "not applicable".

Motivation: A SIEM as an automated detection system for attacks can only be effective if it continuously receives sufficient and, above all, system-specific relevant event messages from the infrastructures and systems to be monitored. General standard event messages may not be sufficient to achieve an adequate level of detection and only allow rudimentary attack detections.

Implementation example: An example system allows end users to log in using a username and password. One of the typical forms of attack for this system would be to try to discover and take over user accounts with weak or frequently used passwords by means of automated password testing (dictionary or brute force attack). The example system is configured to record every failed login event in system protocols ("logs"). By routing this logging data in parallel to a SIEM, the SIEM can detect in real time that an attack is obviously taking place, alert it and thus enable immediate countermeasures.

ID: 3.01-37/7.0

6. Technical Baseline Security for IT/NT Systems

6.1. Basic System Hardening

Req 35	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

For this requirement the following warranty objectives are relevant:

ID: 3.01-10/7.0