Security requirement

Web Applications

Deutsche Telekom Group

Version	7.0
Date	Jul 1, 2024
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
	3.06	Security requirement
Version	**State**	**Status**
7.0	Jul 1, 2024	Released
Contact	**Validity**	**Released by**
Telekom Security psa.telekom.de	Jul 1, 2024 - Jun 30, 2029	Stefan Pütz, Leiter SEC-T-TST

Summary
This document was created on the basis of the general security policies of the group and defines the requirements for securely implementing web applications. The requirements described in this document shall be met to ensure that a web application cannot be easily misused by attackers.

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

Table of Contents

1.	Introduction	4
2.	System Hardening	5
3.	System Update	9
4.	Protection of Data and Information	11
5.	Protection of Availability and Integrity	16
6.	Authentication and Authorization	29
7.	Protecting Sessions	34
8.	Authentication Parameter Password	43
9.	Content Management Systems (CMS)	51
10.	Logging	53

1. Introduction

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

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

2. System Hardening

Req 1	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 2	Features that are not required in the software and hardware used must be deactivated.

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

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

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

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

Implementation example: [Example 1]
Deactivation of debugging functions in the software that are used in the event of fault analysis, but do not have to be active during normal operation.

[Example 2]
Disabling unused network interfaces of a server.

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

ID: 3.01-4/8.0

Req 3	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 4	The web application must not integrate or execute external resources or functionalities from untrusted sources. If the web application integrates external resources, their integrity must be protected by means of Subresource Integrity (SRI).

SRI must be used for all supported subresources, at least for JavaScript and CSS.

For a subresource integrated using SRI (for example, a JavaScript file hosted on an external Content Delivery Network (CDN)), the hash value of the valid file is specified in the "integrity" attribute:

```
<script src="https://example.com/example.js"
integrity="sha384-R4/ztc4ZlRqWuvf6RX5yb/v90qNGx6fS48N0tRxiGgDVJCp2T"
crossorigin="anonymous"></script>
```

The browser will download the script, calculate the hash value for the downloaded file and execute the file, if both hash values match.

For SRI to work, the server hosting the resources must support Cross-Origin Resource Sharing (CORS).

By loading untrusted resources within an iframe and setting the attribute "sandbox" with a restrictive configuration, the risk of such content can at least be reduced, among other things because these resources cannot access the DOM of the actual page then.

Motivation: When using SRI, an embedded script is only executed by the browser if the hash of the downloaded script and the hash specified in advance in the "integrity" attribute match. Manipulated scripts, for example, due to a compromised CDN, are thus not executed (if the browser used supports SRI). This reduces the risk of integrated external resources.

For this requirement the following threats are relevant:

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

ID: 3.06-4/7.0

3. System Update

Req 5	Software and hardware of the system must be covered by security vulnerability support from the supplier.

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

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

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

Support phases with limited scope of services
Many suppliers optionally offer time-extended support for their products, which goes beyond the support phase intended for the general market, but is often associated with limitations. Some suppliers define their support fundamentally in increments, which may include limitations even during the final phase before the absolute end date of regular support.
If a product is used within support phases that are subject to limitations, it must be explicitly ensured that these restrictions do not affect the availability of security vulnerability support.

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

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

For this requirement the following threats are relevant:
- Unauthorized access to the system
- Unauthorized access or tapping of data
- Unauthorized modification of data
- Unauthorized use of services or resources
- Disruption of availability
- Denial of executed activities
- Unnoticeable feasible attacks
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-1/8.0

Req 6	The web application must not use or presuppose any client-side technologies that have reached the status "end of life" or "end of support".

This includes in particular Flash, Shockwave, ActiveX and Java applets.

Motivation: Also, obsolete or insecure technologies which are used only in the browser can be exploited by attackers in order to compromise the client, but also in order to compromise the security of the web application itself.

For this requirement the following threats are relevant:

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

ID: 3.06-6/7.0

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

4. Protection of Data and Information

Req 8	Stored data in need of protection must be protected against unauthorized access, modification and deletion.

The need for protection of stored data depends on its classification (e.g. according to applicable legal data privacy requirements, regulatory requirements, contractual obligations), the potential damage in the event of its misuse, and other relevant factors (e.g. the location of storage). The nature and extent of protective measures must be appropriately chosen.
Stored authentication attributes such as passwords, private keys, tokens or certificates etc. are generally considered to be in need of protection. Data that determines the functionality and security-relevant behavior of a system (e.g. system configuration files, operating systems and kernels, drivers) are also considered to be fundamentally in need of protection.

Compliance with the protection objectives of confidentiality, integrity and availability must be consistently guaranteed for stored data in need of protection. This also applies during only short-term storage (e.g. when storing in a web cache or in a temporary folder within a data processing chain).

Basically, access to data in need of protection in a system must be fully regulated on the basis of technically implemented authorization assignments and controls.
If such technical access control alone is no longer sufficient to ensure the necessary protection requirements of stored data, or if its effectiveness cannot be consistently ensured, additional cryptographic methods (e.g. encryption, signing, hashing) must be implemented. Cryptographic methods used for the storage of data must be recognised and effective for this purpose and must not have any known vulnerabilities.

Motivation: The storage of data on a system without adequate protection enables an attacker to view, use, disseminate, modify or destroy it without authorization. This potentially opens up additional attack vectors on the immediate and connected other systems and can lead to significant failures, loss of control and damage as well as resulting penalties and loss of reputation towards customers and business partners.

Implementation example: [Example 1]
A system exports data for transport to mobile media. Since the system's technical access control at the file permission level no longer applies as soon as the mobile media is removed from the system, additional measures must be taken to protect the data. Before the system writes the data to the mobile media, it is encrypted accordingly using a suitable algorithm. The associated encryption key is exchanged on a separate channel so that the data can be decrypted and processed again in the legitimate target system. An attacker who takes possession of the mobile media, on the other hand, has no access to the data.

[Example 2]
Only cryptographic hashes of passwords generated with a secure password hashing method are stored in the local user database of a system. For the system, these hashes are sufficient to authenticate users when they log on to the system. However, if an attacker can copy the user database, he does not immediately come into possession of plaintext passwords with which he could log on to the system on behalf of the users.

[Example 3]
On a system, the configuration files of the Web server can only be written by the legitimate admin in which corresponding permissions have been set in the file system. The access control of the operating system kernel thus denies all other users of the system to make changes to the configuration files of the web server; including the web server service account itself, which also reduces the attack surface from the outside in case of vulnerabilities in the web server.

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

ID: 3.01-14/8.0

Req 9	Data in need of protection must be protected against unauthorized access and modification during transmission.

The need for protection of data to be transmitted depends on its classification (e.g. according to applicable legal data privacy requirements, regulatory requirements, contractual obligations), the potential damage in the event of its misuse, and other relevant factors (e.g. transmission via public networks). The nature and extent of the protective measures must be appropriately chosen.
Authentication attributes such as passwords or tokens etc. are generally considered to be in need of protection. Data that determines the functionality and security-relevant behavior of a system (e.g. updates & patches, configuration parameters, remote maintenance, control via APIs) are also considered to be fundamentally in need of protection.

Compliance with the protection objectives of confidentiality and integrity must be consistently guaranteed during the transmission of data in need of protection.

As a rule, this requires the implementation of cryptographic methods (e.g. encryption, signatures, Hashes).
Cryptographic methods may

- be applied directly to the data before transmission, which can make subsequent transmission acceptable even via insecure channels
- be used on the transmission channel to create a secure channel and protect any kind of data passing through it
- or be implemented as a combination of both.

Cryptographic methods used for the transmission of data must be recognized and effective for this purpose and must have no known vulnerabilities.

Motivation: The transmission of data without adequate protection enables an attacker to intercept, use, disseminate, modify or remove it from transmission without authorization. This potentially opens up further attack vectors on the immediate target systems as well as connected other systems and can lead to significant failures, loss of control and damage as well as resulting penalty claims and reputational losses towards customers and business partners.

Implementation example: [Example 1]
Confidential documents are encrypted before they are sent by e-mail to the customer.

[Example 2]
An administrator configures a new cloud application over the Internet. Access is via a TLS-encrypted connection ("https").

[Example 3]
A system obtains automatic software updates from an update server. The update server delivers the software updates cryptographically signed. The system can thus validate the received software updates and reliably rule out that they have been manipulated during transmission.

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

ID: 3.01-15/8.0

Req 10	TLS with server authentication and encryption must be used for the web application. In doing so, all content of the web application must be transmitted in encrypted form.

This includes even possibly uncritical content such as JavaScript, style sheets, images, etc.

Motivation: TLS is the usual mechanism for web applications in order to ensure the confidentiality of communications

and the authenticity of the application or server.
The behavior of the various browsers is very inconsistent when it comes to processing mixed HTTP and HTTPS resources. The resulting vulnerabilities, for example man-in-the-middle or surf jacking attacks, are prevented by full encryption.

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

ID: 3.06-10/7.0

Req 11	The web application must use HTTP Strict Transport Security (HSTS) to force all future connections from the browser to be established in encrypted form only. For this purpose, the web application must set the HTTP response header “Strict-Transport-Security”.

Motivation: If a browser receives the HSTS header from a web application, the browser will communicate with this web application only in encrypted form for the length of time specified in the header. If the user subsequently calls up the web application with an HTTP address, the browser automatically changes this to the corresponding HTTPS address. If the encrypted transmission leads to an error code of any kind, the browser stops the connection with an error message. This includes certificate errors that a user can then no longer ignore.
First and foremost, HSTS addresses the problem that many users often do not type in, for example, https://www.anwendung.de as an address, but instead anwendung.de or www.anwendung.de or call up the application from their bookmarks in unencrypted form. Usually, a redirect from HTTP to HTTPS follows. However, this initial unencrypted request including the redirect already allows attackers to perform man-in-the-middle attacks. If a web application sets the HSTS header, this initial unencrypted request is avoided. Furthermore, the browser does not call up any other unencrypted resources if, for example, mixed HTTP and HTTPS resources were accidentally set in the web application.

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

ID: 3.06-11/7.0

Req 12	The web application must not use URL parameters or other fields that are captured in log files to transmit data with need of protection.

Motivation: In the case of the GET method, parameters are transmitted in the URL and are usually logged by web servers and clients. If the web application contains links to other web applications, it is also possible for the URL parameter to be transmitted to these other web applications via the HTTP header "Referer" and then appear in log files there. This part of the problem may also be addressed (for most browsers) by using the HTTP response header "Referrer-Policy“ (for example, with the value "no-referrer“). Moreover, it cannot be ruled out that a user copies a used URL including data with need of protection. In the case of the POST method, for example, parameters are transmitted in the request body.

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

ID: 3.06-12/7.0

Req 13	The web application must not store data with need of protection on the client side, neither temporarily nor persistently.

This applies both for cookies and for the different kinds of local storage of the browser. Instead data with need of protection must be stored on the server side.

As an exception to this, the web application may transmit session identifiers or (session/access) tokens in non-persistent cookies or store this data in the session storage of the browser temporarily.
As a further exception, the web application may store data for identifying the user, such as the user name or an ID, in persistent cookies or other local storages, if personal data is stored in encrypted form and if the user has granted consent.

The web application must also prevent data with need of protection that the user enters in an HTML form from being stored by the browser and automatically entered the next time such a form is used. For this purpose, the web application must set the “autocomplete = off” attribute. This applies in particular to fields for credit card data and other account or payment information. However, we recommend implementing this in general for data with need of protection.

Motivation: All data stored on the client can in principal be analyzed and manipulated, either locally by direct access or possibly by a successful attack from outside.

In case of cookies, a web application can set an expiry date via the "expires" or "max-age" attribute. But then this is a "persistent cookie". The browser stores persistent cookies on the device of the user until the expiry date. Every person with access to the device can read the persistent cookies. This can happen in an Internet café, in other cases of shared use, or through a successful attack. Cookies without an expiry date are not persistent, meaning they are not permanently stored. The browser deletes these cookies as soon as it is closed. However, until then, attackers can also access this data. Therefore, cookies that are not persistent must not contain data with need of protection either.

The various technologies that are used, among other things, for HTML5 applications provide further mechanisms for saving data locally to the client. But these local storages can be analyzed or tampered with on the client, too. For example, the DOM objects "localStorage" and "sessionStorage" are such an alternative that can be used to store data in a browser. However, in local storage data is stored permanently, while data stored in session storage is deleted with the end of the session. But regarding session storage it must be noted that a new session is created, when a website is opened in a new tab or a new browser window. All scripts of a domain, from which the data was stored, are able to access the stored data (of the same local user profile in case of local storage or of the same tab/browser window in case of session storage). Unlike cookies via the "HttpOnly" flag, there is no inherent protection mechanism against cross-site scripting for these storages.

Deactivating autocomplete prevents data that was entered from being stored locally regardless of the browser configuration. In this context it should be noted that most current browsers ignore the "autocomplete" attribute for "password" fields.

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

ID: 3.06-13/7.0

Req 14	The web application must prevent caching of data with need of protection.

For this purpose, the web application must set the following HTTP headers:

```
Pragma: no-cache
Cache control: no-cache, no-store
Expires: <Current server date>
Date: <Current server date>
```

Content that does not include data with need of protection, for example, images, style sheets or public information, may be cached.

To avoid various security issues regarding caching, the web application must not support "fat GET requests" (that is GET requests with a request body).

Motivation: The web application must prevent data with need of protection from being revealed by means of caching, for example due to other users who use the same computer. The appropriate HTTP headers instruct a browser or a proxy as to how to handle caching. Even if HTTPS prevents the data with need of protection from being stored in proxies, relevant provisions have to be specified. For one thing, this is an additional security measure, and for another, the browser needs to be prevented from caching the data anyway. As browsers and proxies do not act consistently regard-

ing caching, all specified headers need to be set in order to prevent caching as reliably as possible.

- *"Pragma: no-cache" issues the instruction not to allow caching;*
  (This HTTP/1.0 directive must still be used, because certain legacy proxies do not support the HTTP/1.1 directive "Cache-Control". Although this is originally a request directive only, many proxies accept it as a response directive, too.)
- *"Cache-Control: no-cache" instructs the browser to always request a new page;*
- *"Cache-Control: no-store" instructs browsers and proxies not to cache;*
- *"Expires" specifies the date from which content should be treated as expired;*
- *"Date" specifies the date on which the content was generated.*

The usage of meta tags such as <META HTTP-EQUIV="Pragma" CONTENT="no-cache"> is not an ideal solution since, for example, proxies do not evaluate any content of HTML documents.

Fat GET requests can cause caching systems to cache an associated response including data from the original request body. This can lead to various security problems such as cache poisoning.

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

ID: 3.06-14/7.0

Req 15	Error messages of the web application as well as other outputs (for example, HTTP header) must not contain details of implementation or other sensitive information, which might be misused by the user for attacks.

Sensitive information includes, for example, information relating to the used software or middleware, function calls, SQL instructions, version numbers and patch levels.

Motivation: Such information can be used by an attacker to prepare specific attacks. In this way an attacker could, for example, use the precise software version to identify vulnerabilities in the product and, in a second step, exploit them.

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: 3.06-15/7.0

5. Protection of Availability and Integrity

Req 16	The web application must validate on the server side all input data that is transferred by a client, before the data is processed. This does not only apply to all data entered by a user. This explicitly also applies to data that a user does not influence during normal use.

In addition to the GET and POST parameters, this also includes, for example, input data from cookies, hidden fields, HTTP header fields etc.

In this process, all input data must be checked regarding the data type, length, characters, format, value range, encoding and content.

The basic principle here should be: anything not expressly allowed is forbidden. So, it must be ensured that the data contains only contents and formats that were previously defined as valid (positive list).

If there is input data using a markup language (XML, HTML, YAML, JSON etc.), both the correct structure and the data contained in the structure must be checked.

Before the data is validated, it must be decoded and canonicalized so that it is available in a standardized (canonical) form. Usually, the utilized frameworks ensure this.

If the web application discovers during data validation that the data deviates from the expected contents or formats, the web application must reject this data and stop the requested action. Depending on the criticality of the web application, it may also be advisable to terminate the current session. This applies in particular if conscious data manipulation can be assumed. This can, for example, be the case if the length of the transmitted data exceeds the length that was specified in the form field, if additional parameters are transmitted or if the transmission method for the parameters does not match the specifications of the web application (GET vs. POST, cookies vs. hidden field).

In some scenarios, web applications automatically correct incorrect entries (“sanitizing”). This enables a web application to, for example, allow user-friendly entry of data with various spellings. However, this also leads to new risks due to the additional complexity and new attack vectors. Therefore, sanitizing must be avoided wherever possible. Sanitizing should be used only in those scenarios in which misuse of the sanitizing or conscious manipulation of the data cannot occur. With sanitizing, it is also important to pay attention to the nested input of attack vectors (such as “<sc<script>ript>”), which an attacker can use to circumvent filters.

Client-side validations that are, for example, performed using JavaScript are not considered trustworthy from the viewpoint of a server-side application. However, they may still be performed to improve the user-friendliness of the web application.
However, if input data is already processed by web application logic on the client side without any possibility for it to be validated on the server side first, the data must at least be validated on the client side before processing. This concerns input data of any kind, but also data (subsequently) loaded from untrustworthy servers or any data loaded from local media.

Motivation: All data that is transmitted by the client is not considered trustworthy since an attacker would have no difficulty manipulating it. Input validation ensures that no security problems or other unexpected side effects occur due to the processing of manipulated or corrupt data.

Input validation based on filtering out known dangerous patterns is problematic: There is a danger of some patterns being forgotten at the design stage or during implementation. Moreover, due to new attack methods further problematic patterns may arise during live operation.

Input data can be available in different encodings and notations. Depending on the encoding scheme used, the same value can accordingly be interpreted differently. If a web application validates data without taking account of its encoding and notation, it may not recognize harmful data. An attacker will then be able to circumvent the data validation by using a specific encoding or even multiple encodings.

Especially if the user of a web application attacks this application himself, client-side validations do not offer any protection, as the validation logic can also be manipulated or simply bypassed. However, if data is processed exclusively on the client side, this processing must ultimately also be protected by sufficient validation. For example, if JSON from

external sources is processed client-side, return parameters must be analyzed for manipulation; only then may they be passed on to the interpreter. Local media are also easy for an attacker to manipulate, for example, to carry out overflow or injection attacks. However, these validations can then only be carried out on the client side (due to the lack of server-side processing) and thus offer the only possible protection, especially if the attack is carried out by a third party without direct access to the client-side functions (for example, in case of DOM-based XSS).

But not only the application logic can be attacked by manipulated input data. If such data is passed on to an interpreter or other components, injection attacks can occur, among other things. This leads to manipulation of downstream logic, queries or commands. Well-known types of injection attacks are SQL/NoSQL injection, XML/XPath injection, code/command injection or e-mail injection. But other components of web applications can also be affected, for example, template engines (server-side template injection, SSTI), LDAP interfaces (LDAP injection), or even log servers (log injection). Correct and complete data validation is therefore also the basis for protection against injection attacks. Depending on the interpreters or other components used, however, additional specific measures, in particular a consistent separation of input data and commands, may have to be implemented for comprehensive protection.

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

ID: 3.06-16/7.0

Req 17	The web application must initially consider all data that is transmitted from another system to be untrustworthy. Depending on the criticality of the web application and the trustworthiness of the other system, the web application must validate this data, too.

Motivation: Data that is transmitted by a client and can therefore be directly influenced by a user is not the only type of data that can contain malicious character strings. This can also be the case with data that comes from other systems, for example if these systems do not perform sufficient data validation. Therefore, it is important to check whether validation needs to be performed for this data as well. Ultimately, it is important to ensure that the web application uses data in the expected format only, thus preventing any unexpected side effects.

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

ID: 3.06-17/7.0

Req 18	The web application must not use any user input data for direct access to files and directories as well as to other server-side resources.

Instead, indirect object references of server-side resources must be implemented.

Motivation: If a web application uses input data for direct access to files, an attacker may be able to gain access to files that he would not usually be able to access.
If, for example, a web application shows the Deutscht.txt help file, when a user enters "Deutsch", and the web application does also not sufficiently validate the user input, the attacker can possibly cause the web application to show the user file by entering "../../../../etc/passwd%00" (path traversal/local file inclusion). Hazardous characters in this context are the characters for switching the directory and the null byte. In this example, the null byte character ends the string, thus preventing ".txt" from being added. The attacker can also attempt to use these malicious characters in various encodings. Furthermore, if there is a corresponding security gap, an attacker can even incorporate external files into a script and thus cause code to be executed (remote file inclusion).

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

ID: 3.06-18/7.0

Req 19	If the web application deserializes objects, these objects must not contain any input data from users or from other untrusted sources in order to prevent attacks due to insecure deserialization.

If this is not possible and if no formats can be used for serialization that only allow primitive data types (such as JSON), the deserialization must be secured by further measures, for example via integrity controls or strict limitation of the data types. It should be noted, however, that all security measures that are only implemented after deserialization cannot provide complete protection.

Motivation: Serialization refers to the conversion of information, typically data structures of a programming language, into a sequential representation form (typically a stream of bytes, but also other data formats such as JSON). This is used, for example, to transmit the data via communication interfaces or to store it in files or databases. The reverse process, for example making a data structure out of a byte stream, is called deserialization.
However, a byte stream can also be manipulated. If a web application deserializes objects that can be manipulated by an attacker, this can lead to critical security problems, if the implementation is not secure (deserialization attacks or object injection attacks).
To make matters worse, many deserialization attacks affect the deserialization process itself and are completed before that process is finished, so that validations of the deserialized objects cannot provide complete protection. And in that case, a web application can be affected by a deserialization attack even if the manipulated data is not processed in any functionality of the web application.
Depending on the respective application, an attacker can use deserialization attacks to manipulate the logic of the application, carry out DoS attacks or even have his own code executed.
In a very simple and easy to read example of a PHP forum, object serialization could be used to store a cookie containing, among other things, a user ID and the associated user role:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

An attacker could manipulate the serialized object to give himself administration rights:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

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

ID: 3.06-19/7.0

Req 20	If files are uploaded by a client or transferred from other systems, the web application must check these files and process them in a such a way that no risks arise for the application or for users.

Suitable measures for this include:
- Limiting the size of transmitted files.
- Limiting the number of files that a user can transmit.
- Storing files as database BLOB instead of in the file system. If the files are stored in the file system, they are stored outside the website root directory and the system is hardened regarding authorizations. Furthermore, the name of the file in the file system is not defined by the user, but is generated by the application.

- Examining files for malware and (in particular in case of SVG files) for active code or, alternatively, transcoding (in the case of images, video files and audio files).
- Analyzing archive files for malware and ZIP bombs.
- Validating the file names for correct file extensions, integrated path information and active code, in particular JavaScript in file names.
- If meta data of files (for example, Exif data) is processed or displayed, the meta data is also validated for active code.
- Validating the format of uploaded files. The format must correspond to the file extension and be valid, that is the format meets the specification; and only certain files formats are accepted, for example, only image formats in the case of a photo gallery.
- If users call up untrusted files that have been uploaded by other users:
    - in the HTTP response of the download, the HTTP header "Content-Type" is set with the specification of the correct media type of the file,
    - in the HTTP response of the download, the HTTP header "Content-Disposition" is set with the value "attachment" and the parameter "filename",
    - the uploaded files are stored on their own (sub)domain.

In this case, it must be pointed out that it is not necessary to implement all measures at any rate. Instead a selection of measures which is appropriate for the specific use case must be implemented.

Motivation: An upload function is in many use cases very problematic from a security point of view, as the various attack scenarios associated with it are difficult to prevent completely. Thus, active code or malware in uploaded files jeopardize the application and/or other users who open or download these files. In addition, other risks may arise as a result of uploaded files, such as denial-of-service attacks.

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

ID: 3.06-20/7.0

Req 21	Each output of the web application must specify the media type und the character encoding which is used. For this purpose, the HTTP header "Content-Type" must be set. In addition, the HTTP header “X-Content-Type-Options” must be set with the value “nosniff”.

In the case of HTML output, this must be in an encoding which is provided in the HTML standard, as a rule UTF-8: “Content-Type: text/html; charset = utf-8”.

In the case of HTML documents for which the application scenarios lead to the assumption that users will download the documents, media type and character encoding must additionally be defined using a corresponding <meta> directive.

For content that is embedded (usually by means of <embed> or <object>), the media type must be specified using the “type” parameter.

For other output, suitable encoding tags must also be used (for example, via a suitable encoding attribute in the XML declaration for XML documents).

Motivation: These measures prevent clients from interpreting the data that a web application supplies incorrectly or decoding it incorrectly. The "nosniff" header instructs the browser to trust the specified media type and not to attempt to determine it itself (MIME sniffing). Incorrect determination of the media type or faulty decoding can lead to security problems in particular during validation of input data.

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

ID: 3.06-21/7.0

Req 22	The web application must validate all input data that is returned to the browser and it must HTML encode all meta characters to prevent cross-site scripting (XSS).

At least the following HTML meta characters must be converted into normal plain text characters:

- & in: &
- " in: "
- < in: <
- > in: >
- ' in: '

As an additional protective measure, a Content Security Policy (CSP) can be defined by means of the corresponding HTTP header, with a positive list of permitted sources for JavaScript files (as well as for other resources). The policy should include a "default-src" directive as a fallback solution for types of resources for which no separate directive has been defined in the policy.

Motivation: XSS refers to the planting of malicious HTML and JavaScript code on a web page. Such attacks are possible if the web application accepts input data and sends this to a web browser without sufficient validation or encoding, such as this comment function of an insecure web application:
`https://www.insecure-example.com/comment?message=<script src=https://evil-example.com/evilscript.js></script>.`
Thus, an XSS vulnerability enables an attacker to run script code in a victim's browser in order to, for example, take over sessions, change page content or redirect the user to malicious websites.
In principle, there are three different types of XSS attacks: reflected XSS (the server returns manipulated input data directly back to the browser just once), persistent XSS (the malicious code is stored on the web server, meaning it is sent every time it is requested) and DOM-based XSS (where the web application on the server is not involved, because the malicious code is transmitted for execution directly to a script on the client side).

To prevent XSS, the various meta characters need to be escaped or encoded.
Meta characters are characters that have a special meaning for the respective markup language or query language and are interpreted accordingly during processing. By using meta characters, an attacker can attempt to manipulate commands, processing logic or the representation of data and information.
Therefore, escaping of such meta characters must always be performed depending on the specific context in which the meta characters are output (HTML, JavaScript, CSS, URL, etc.). HTML encoding is the basic protective mechanism against these attacks if user input is simply output in the website HTML. In the case of output in other contexts, protection against XSS is much more complicated. Developers must implement this in their web applications if this is not already ensured through a correctly used framework.

By defining a Content Security Policy, the browser (if it supports CSP) is instructed to only execute scripts from files loaded from trusted domains. The browser will ignore all other scripts, including inline JavaScript. This can significantly reduce the possible attack vectors for XSS, but requires that the web pages can be built accordingly and that the policy is carefully defined and permanently maintained.

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

ID: 3.06-22/7.0

Req 23	If the web application outputs input data outside of the HTML context (that is in JavaScript, in the URL or in the CSS, for example), this output must be secured by means of context-specific escaping the input data to prevent XSS.

Hence, escaping must always be performed depending on the specific context in which data is output:
If the data is output as part of a URL, for example, escaping must be performed by URL encoding of the input data: URL encoding replaces insecure characters by a "%"-character followed by the corresponding hexadecimal ASCII value.
If, on the other hand, input data is output within a JavaScript string, non-alphanumeric values must be Unicode-escaped, for example, by means of replacing "<" with "\u003c".

Motivation: Outside of the HTML context, simple HTML encoding is no longer sufficient to prevent XSS. In this case, the input data must be escaped depending on the context in which it is output. In such cases effective protection against XSS attacks may be more complicated and more susceptible to errors.

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

ID: 3.06-23/7.0

Req 24	If changes to the Document Object Model (DOM) of a website are possible based on input data, this input data must be validated on the client regarding meta characters. The output must be secured by means of context-specific escaping of the input data, to prevent DOM-based XSS.

In particular, client-side data validation must always be carried out, if it is not possible to check on the server side what the DOM finally looks like, or if merely data (for example, JSON) is returned that is assembled into HTML only in the client.

However, since it is extremely difficult to effectively protect against DOM-based XSS attacks, it is recommended to refrain from making corresponding changes to the DOM.

If such functionality is nevertheless provided, it is recommended to use secure output methods or properties (especially "element.textContent"), which would not execute malicious code. Unsafe output methods/properties (such as "element.innerHTML", "element.outerHTML", "document.write", "document.writeln") are accordingly to be avoided if input data is used.
In addition, a library with appropriate protection functions (based on context-specific escaping) must be used or, if necessary, implemented very carefully. These can also be DOM-/JavaScript-based XSS filters that perform the analysis of the markup directly in the browser, whereby input data is then first sent to this sanitizer/filter and only afterwards transferred to the actual DOM.

Motivation: In a DOM-based XSS attack (also known as local XSS), the attacker plants malicious code via a vulnerability in the client-side JavaScript code. A web application is vulnerable to such an attack if it uses data from objects that the attacker controls (for example, "document.URL") without first checking this data for planted code. At no point is the malicious code part of the HTML page supplied by the server. Instead, it is parsed into the DOM by the client. In addition, there are many variants of this attack, and the various browsers' implementations regarding DOM manipulation are often inconsistent. For these reasons, it is extremely difficult to protect against these attacks. It is therefore necessary to proceed with particular care here.

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

ID: 3.06-24/7.0

Req 25	If text formatting is permitted in user input, if a WYSIWYG editor is used for user input, or if a markup language which is converted into HTML is used for user input, a filter for permitted tags must be implemented in the web application to prevent XSS.

Attributes that can be used to execute scripts, for example, <style> or <on*>, must not be allowed.

Alternatively or complementarily, DOM-/JavaScript-based XSS filters can also be used that perform the analysis of the markup directly in the browser, whereby input data is then first sent to this sanitizer/filter and only afterwards transferred to the actual DOM.

Motivation: If due to the use cases of the web application HTML encoding is not possible, the most secure alternative for preventing XSS is to allow and to output only (predefined) harmless tags. The permitted tags must be selected carefully and restrictively, otherwise dangerous attack variants can easily be overlooked.

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

ID: 3.06-25/7.0

Req 26	If input data is used when forming database commands, the web application must prevent injection attacks (SQL injection or NoSQL injection) through suitable countermeasures.

This applies both to SQL injection in the case of SQL databases and to, for example, JavaScript or CQL injection in the case of NoSQL databases.

For SQL and CQL requests, all variables must be bound in prepared statements. If stored procedures are used, these must also be used with bind variables. Dynamically combined requests must not be used.

If no secure access method, such as prepared statements, is available, the data must be cleansed. This must be performed depending on the specific interpreter language that is used. All meta characters in the interpreter language must be removed from the input or escaped. Here, meta characters are all characters that are used as string delimiters or for syntactical formatting of the code (" ' ; { } $ % # | etc.).

Motivation: Structured Query Language (SQL) is a query language for relational databases. Predefined sections are combined with user inputs to create a complete query. If user input is not sufficiently checked or preprocessed, an attacker can plant any SQL commands in the query (SQL injection):

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

The attacker modifies the GET parameter "id" and sends ' or '1'='1:

```
http://example.com/accountView?id=' or '1'='1
```

This changes the meaning of the query to return all the records from the table.

```
SELECT * FROM accounts WHERE custID=' ' or '1'='1'
```

The use of prepared statements prevents user inputs from being processed by the SQL interpreter, since these are precompiled and parameterized for processing.

CQL (Cassandra Query Language) is an interpreter language for Cassandra databases. CQL is therefore generally vulnerable to injection attacks, too, although it is a NoSQL database.

Other NoSQL databases, such as MongoDB, use other interpreter languages, JavaScript in that case. Hence, here the meta characters of the respective syntax can be used to perform injection attacks. This means data that is used to form the database queries must be validated with reference to these meta characters.

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

ID: 3.06-26/7.0

Req 27	If the web application creates or processes XML data structures, XSLT style sheets or XPath queries and if it also uses input data for this, the web application must prevent XML injections or XPath injections as well as XML eXternal Entity Injection (XXE) with suitable countermeasures.

For this, the user input must be checked against a positive list and rejected, if it does not meet the definition of the positive list. If this approach is not possible, the characters ( ) [ ] < > / ' = " : , * and all white space characters (spaces and tabs) must be removed or escaped.

To prevent XXE attacks, in particular the evaluation of external entities and the use of external DTDs by the parser must be disabled.

Motivation: XPath is a language for querying XML data. XPath injection is the XML counterpart to SQL injection. If an attacker plants XML meta characters, this can lead to malformed XML. In addition, it is possible to introduce entire tags or attributes.

XPath injection might, for example, be possible in case of the following XPath query that returns the account whose username is "gandalf" and the password is "!c3":

```
string(//user[username/text()='gandalf' and password/text()='!c3']/account/text())
```

If the web application does not properly validate user input, the attacker will be able to inject XPath code and interfere with the query result. So, the attacker could input the following values:

```
Username: ' or '1' = '1
Password: ' or '1' = '1
```

Using these parameters, the query becomes:

```
string(//user[username/text()='' or '1' = '1' and password/text()='' or '1' = '1']/account/text())
```

As in a common SQL Injection attack, the attacker has created a query that always evaluates to "true", which means that the web application will authenticate the user even if no valid combination of username and password has been provided.

XML injection is, for example, possible where CDATA elements are used to insert malicious content that is ignored by the XML parser.

```
<HTML>
<![CDATA[<IMG SRC=http://www.example.com/logo.gif onmouseover=javascript:alert('Attack');>]]>
</HTML>
```

In this example XML injection could be used for an XSS attack against the web application.

XML documents can contain references to external entities or documents that are automatically resolved when the XML document is processed by the web application's parser, which then loads and evaluates the external resource. An attacker could use this for XXE to execute a remote request from the server, scan internal systems, extract data, perform a denial-of-service attack or even other attacks.
In this way, an attacker could try to read data on the server by means of XXE:

```
< <?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

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: 3.06-27/7.0

Req 28	The web application must not use any input data to create shell commands or commands for the active program (for example, via "eval").

Motivation: This prevents OS command injection and code injection.
OS command injection can occur when input data is integrated into a command that is then passed to the operating system for execution. This may allow an attacker to execute arbitrary OS commands on the server running the web application. In that case, he can usually compromise the system completely. Very often, an attacker can even exploit such a vulnerability to compromise other parts of the hosting infrastructure by exploiting trust relationships and extending his attack to other systems.
With code injection, on the other hand, an attacker plants executable code. This can be server-based (for example, via php) or client-based (for example, via JavaScript). If the attacker is successful, this code is executed and can cause major damage, too. However, an attacker can also use code injections to insert XSS or SQL injection attacks.

For example, this code could be part of a web application to administer a web server:

```
string dirName = “C:\\filestore\\” + Directory.text;
ProcessStartInfo psInfo = new ProcessStartInfo (“cmd”, “/c dir “ +
dirName);
…
Process proc = Process.Start(psInfo);
```

This function displays the contents of a directory. The script inserts the value of the parameter "Directory" as provided by the user into a specified command. However, an attacker could use shell metacharacters to inject his own commands and have them executed: `directoryname && rm -rf.`

If the PHP function eval() is used and untrusted data is passed to it that an attacker can influence

```
$username = $_POST[‘username’];
eval(“echo $username”);
```

code injection is possible, for example by entering `mustermann; phpinfo().`

If on the other hand in case of JavaScript a JSON document is interpreted via "eval"

```
eval({"menu":{"address":{"line1":addressLine1,"line2":"","line3":""}}}
);
```

and then the "addressLine1" variable from the request is assigned the value

```
",arbitrary:alert('executed!'),continue:"
```

injected code will be executed here, too.

All user inputs represent a potential attack surface for this. Direct execution via "eval" (or comparable functions) is highly problematic as the data is interpreted directly without further checks regarding security or validity. This applies on both the server side and the client side.

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: 3.06-28/7.0

Req 29	If the web application uses input data to form HTTP headers, this data must be validated. Dangerous characters, in particular CR and LF (%0d and %0a), must be filtered out before the input data is inserted into the header of an HTTP response.

Motivation: If headers (redirection URL, "Set-Cookie" instruction, etc.) are generated dynamically and input data is included that was not verified sufficiently, HTTP header injection can occur. This can be used for HTTP response splitting attacks. To do this, the attacker places a CR/LF character in the input data followed by a complete HTTP header. He thus triggers a response with an HTTP request, which the victim interprets as two HTTP responses. The second of these responses is entirely under the attacker's control.

If, for example, the name of an author of a weblog is read from an HTTP request

```
String author = request.getParameter(AUTHOR_PARAM);
```

and the name is set in a cookie header of an HTTP response, the "Set-Cookie" instruction will only maintain its correct

form, if the value submitted for "AUTHOR_PARAM" does not contain any malicious characters. If an attacker submits a string, such as "`Wiley Hacker\r\nHTTP/1.1 200 OK\r\n...`", then the HTTP response would be split into two responses of the following form:

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Wiley Hacker

HTTP/1.1 200 OK
...
```

Due to the missing validation of input data, the second response is completely controlled by the attacker and can be constructed with any header and body content desired. And the ability of the attacker to construct arbitrary HTTP responses permits a variety of resulting attacks including defacements, cache poisoning, XSS attacks and page hijacking.

In order to prevent this, input data must be validated, and dangerous characters must be filtered out, before the data is inserted into the header of an HTTP response.

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

ID: 3.06-29/7.0

Req 30	If the sending of e-mails is initiated through the web application, steps must be taken to prevent e-mail commands from being injected into the downstream mail server (e-mail injection).

Relevant e-mail commands in this context are SMTP commands, IMAP commands, attributes of Internet messages and MIME boundaries.

A secure e-mail library must be used that does not permit this kind of injections.

If no secure e-mail library is available, all input data must be validated and cleansed before being forwarded to an e-mail server:

- For ARPA Internet messages: Input data must not contain “<CRLF>” (apart from in the body).
- For IMAP data: Input data must not contain “<CRLF>”.
- For SMTP data: Input data must not contain “<CRLF>”. Exception: In the case of the SMTP command DATA, every occurrence of “<CRLF>.” must be replaced by “<CRLF>..”.
- For MIME e-mails: Input data that is sent as part of the e-mail body must not contain “<CRLF>--”. Input data that contains “<CRLF>--” must be cleansed, for example, by inserting a space between “<CRLF>” and “--” (which however cannot be reversed by the recipient).

Motivation: Validating user input prevents attacks via e-mail injection in which the attacker can execute various undesirable actions: sending anonymous e-mails, spamming, relaying, circumvention of CAPTCHAs and other restrictions in the application as well as the exploitation of protocol vulnerabilities, etc.
Secure e-mail libraries exist which execute corresponding validation measures and per se already prevent e-mail injection. The most secure method is therefore the use of such an e-mail library.

An SMTP injection attack could, for example, be performed against a webmail parameter associated with sending an e-mail. Commonly, the webmail application presents to the users a form where they must provide the required information. The corresponding input data is then used to subsequently create SMTP commands.

A typical part of a HTTP request for e-mail sending would look like this:

```
POST http://<webmail>/compose.php HTTP/1.1
...
-----------------------------134475172700422922879687252
Content-Disposition: form-data; name="subject"
SMTP Example
```

```
-----------------------------134475172700422922879687252
...
```

Which would generate the next sequence of SMTP commands:

```
MAIL FROM: <mailfrom>
RCPT TO: <rcptto>
DATA
Subject: SMTP Example
...
```

But if the application doesn't correctly validate the value in the parameter "subject", an attacker could inject additional SMTP commands into it:

```
POST http://<webmail>/compose.php HTTP/1.1
...
-----------------------------134475172700422922879687252
Content-Disposition: form-data; name="subject"
SMTP Injection Example
.
MAIL FROM: notexist@external.com
RCPT TO: user@domain.com
DATA
Email data
.
-----------------------------134475172700422922879687252
...
```

The commands injected above would produce a SMTP command sequence that would be sent to the mail server, including the "MAIL FROM", "RCPT TO" and "DATA" commands as shown here:

```
MAIL FROM: <mailfrom>
RCPT TO: <rcptto>
DATA
Subject: SMTP Injection Example
.
MAIL FROM: notexist@external.com
RCPT TO: user@domain.com
DATA
Email data
.
```

And an additional e-mail will be sent.

For this requirement the following threats are relevant:

- Unauthorized use of services or resources
- Denial of executed activities

ID: 3.06-30/7.0

Req 31	The web application must not use input data to form URLs (for example, for redirects) in order to prevent the web application from being misused for insecure redirects and to prevent other attacks such as Server Side Request Forgery (SSRF) or HTTP host header attacks.

If input data must be used for this purpose, attacks must be prevented via a positive list (of permitted domains, but also permitted parameters, protocols, etc.).
Input data generally includes not only the data that a user enters during normal use, but also explicitly such data that is not influenced during normal use. In this context, this applies in particular to the HTTP header "Host" as well as other HTTP headers, such as "X-Forwarded-Host".

Motivation: Insecure redirects can be used by an attacker, for example, to carry out more effective phishing attacks or to lure users to pages with malicious code. The victim clicks on seemingly legitimate links received from the attacker, but is then redirected to the phishing site. The attacker merely manipulates the input data used for a redirect. He thus uses the original website as a trustworthy springboard.

Web applications that send requests themselves on the server side, for example, to reload and display resources, can be vulnerable to SSRF if they use unvalidated input data for this purpose. In this case, attackers may be able to modify the URL or other data in such a way that otherwise access-protected content (from internal backend services or even from localhost) is displayed.
HTTP host header attacks compromise web applications that process the value of the host header (or other HTTP headers) in an insecure manner. Depending on the individual application, this can lead to various security problems, such as manipulation of the application logic or more advanced attack variants such as SQL injection.

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

ID: 3.06-31/7.0

Req 32	If it is possible to initiate an e-mail or SMS from within the web application without prior authentication (for example, in case of a password reset or for recommendations) or to send data to the web application without prior authentication (for example, contact forms or page ratings), the web application must prevent misuse of this function (in particular for spam or denial of service) through suitable security measures.

For example, the number of e-mails, text messages or forms that can be initiated using this function within a certain time period can be restricted. A disadvantage of this measure is that it might easily be misused to perform a denial-of-service attack for this function, thus blocking legitimate users.
CAPTCHAs can also be used for this. Either the web application requests an answer to a CAPTCHA for each action, or the web application displays a CAPTCHA only in case a defined threshold value is exceeded or in case of other irregularities.
A CAPTCHA is a test to distinguish between machines and people. The most frequently used variety is image CAPTCHAs, consisting of an image with distorted numbers and letters. If the content of a CAPTCHA is successfully entered into a text field, the web application can assume that the input was not performed automatically. However, you should note that these solutions are usually not barrier-free. This can be a problem in particular in the case of web applications that employees use. There are also low-barrier versions of CAPTCHAs. Examples of this are text CAPTCHAs that present easily solvable arithmetic problems or knowledge questions (“the number 12 plus the number 6 equals?”). Alternatively, an audio CAPTCHA can be provided in addition to the image CAPTCHA. So that this audio CAPTCHA cannot be solved via voice recognition, a disguised voice or acoustic backdrop with background noise is usually used.

In case of a function for sending recommendations, the following measures must be implemented:
- The specification of recipient addresses must be limited to a reasonable and minimal amount.
- The web application must specify the subject line of an e-mail recommendation. Either the application automatically inserts the subject line, or the user selects the subject line from a predefined list.
- A note must be included in the e-mail recommendation stating that the sender’s e-mail address has not been verified.
- If the text body of the e-mail recommendation contains a text field that the user can fill out, the e-mail must contain information regarding the purpose of the e-mail. The text the user enters must be identified accordingly as user-defined text. Only letters, digits and certain carefully selected special characters must be permitted in the text field. If an HTML e-mail is sent, special characters must be HTML encoded (for example, “<” instead of “<”).

Motivation: An attacker sending a large number of text messages or e-mails to one or more users leads to dissatisfied users, image problems and possibly to a large number of inquiries forwarded to customer service. If users can send contact forms without prior authentication, an attacker can anonymously send a great many of these forms in an easy way. When these forms are evaluated by customer service, an overload might occur, so other legitimate requests will not be answered (on time). By means of mass transmission or systematic transmission of page ratings, an attacker can

arbitrarily manipulate any rating.

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

ID: 3.06-32/7.0

6. Authentication and Authorization

Req 33	Use of functions of the web application with need of protection and access of data with need of protection must not be possible without successful authentication and authorization.

At this, the access protection of a web application must not rely solely on data, content or functions being kept secret. For example, access protection which is only based on the fact that such resources are not linked is not sufficient.

In particular also after a completed authentication process, for every user action, the web application must verify on the server side that this user is authorized to access the data/content or execute the function. Among others, it is important to note in this context that all parameters and identifiers transmitted by the client are not trustworthy. An attacker can easily manipulate this data to gain access to resources for which he has no authorization. This can, for example, lead to success in case of Insecure Direct Object References (IDOR), when references to internal implementation objects (such as a files, directories, or database keys) are also used for user access without sufficient access control or other protection mechanisms.

In case of forms that must be filled out in a specific order, the authorization check must be carried out for each individual step, because it cannot be assumed that malicious users will call up the forms in the intended order.

Motivation: An authentication is necessary to doubtlessly identify a user because the allocated authorization, and therefore the access on data and functions of the web application depends on that.
A web application must recognize and prevent unauthorized access attempts. Otherwise an attacker can, among others, manipulate parameters and thus gain access to data or execute functions for which he has no authorization. For example if in case of https://www.webanwendung.de/user_account?user_id=123 the parameter "user_id" is used directly and without additional controls as an index for database queries (Insecure Direct Object Reference), an attacker could modify the value of "user_id" in order to view the records of other users. But actually, all parameters that are exchanged between the browser and web application can be manipulated – form fields, URL parameters, hidden fields, HTTP headers, etc.
An attacker can access data and functions that are kept secret by determining the relevant addresses through fuzzing, finding them out via social engineering or because he knows them thanks to previous activities.

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

ID: 3.06-33/7.0

Req 34	The permissions for users and applications must be limited to the extent necessary to fulfill their tasks.

The permissions on a system must be restricted to such an extent that a user can only access data and use functions that he needs in the context of his work. Appropriate permissions must also be assigned for access to files that are part of the operating system or applications or that are generated by the same (e.g. configuration and logging files).

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

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

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

- Unauthorized use of services or resources
- Denial of executed activities

ID: 3.01-23/8.0

Req 35	User accounts must ensure the unique identification of the user.

Users must be identified unambiguously by the system.

This can typically be reached by using a unique user account per user.

So-called group accounts, which are characterized by the fact that they are used jointly by several people, must not be used. This also applies without restriction to privileged user accounts. Most systems initially have only a single user account with administrative privileges after the basic installation. If the system is to be administered by several persons, each of these persons must use a personal, individual user account to which appropriate administrative authorizations or roles are assigned

A special feature are so named technical user accounts. These are used for the authentication and authorization of systems among themselves or of applications on a system and can therefore not be assigned to a specific person. Such user accounts must be assigned on a per system or per application basis. In this connection, it has to be ensured that these user accounts can't be misused.
Ways to prevent misuse of such user accounts by individuals include:

- Configuration of a password that meets the security requirements and is known to as few administrators as possible.
- Configuring the user account that only a local use is possible and a interactive login isn't possible.
- Use of a technique for authentication of the specific user account with public and private key or certificates.
- Limiting the access over the network to legitimate systems.

Additional solution must be checked on their usability per individual case.

Motivation: Unambiguous user identification is mandatory to assign a user permissions that are necessary to perform the required tasks on the system. This is the only way to adequately control access to system data and services and to prevent misuse. Furthermore, it makes it possible to log activities and actions on a system and to assign them to individual users.

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

ID: 3.01-22/8.0

Req 36	Predefined user accounts that are not required must be deleted or at least disabled.

On many systems, there are predefined but unused user accounts (e.g. "guest") after the initial installation.

These predefined user accounts must be deleted or at least disabled immedately after the initial installation; if these measures are not feasible, the corresponding user accounts must be blocked for remote access. In any case, disabled or blocked user accounts must also be provided with an authentication attribute (e.g. a password or an SSH key) so that unauthorized use of such a user account is prevented in the event of a misconfiguration.

Excempt from the requirement to delete or disable predefined user accounts are user accounts that are used exclusively for internal use on the corresponding system and that are required for the functionality of one or more applications of the system. Even for such a user account, it must be ensured that remote access or local login is not possible and that a user of the system cannot misuse such a user account.

Motivation: User accounts that are predefined by default in a product are typically common knowledge and can be targeted by an attacker for brute force and dictionary attacks. If these user accounts are not needed in a specific system, their existence represents an unnecessary attack surface. A particular risk is posed by predefined user accounts that are preconfigured without a password or with a well-known standard password. Such user accounts can be misused directly by an attacker if their security hardening was missed due to the unplanned use in the specific system.

For this requirement the following threats are relevant:
- Unauthorized access to the system
- Unauthorized use of services or resources
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-7/8.0

Req 37	If the web application can be used via the Internet, the registration process for the web application must ensure that the registration cannot be performed automatically.

We recommend using a double opt-in procedure for this – the web application sends a confirmation link (or an initial password/activation token) to the new user's e-mail address. The web application completes the registration only after the user has confirmed the registration via this link.
Additionally, the number of registrations that can be performed within a certain time (for example, from an IP address) can be limited.
CAPTCHAs can also be used for this. A CAPTCHA is a test to distinguish between machines and people. The most frequently used variety is image CAPTCHAs, consisting of an image with distorted numbers and letters. If the content of a CAPTCHA is successfully entered into a text field during a registration, the web application can assume that the registration was not performed automatically. However, you should note that these solutions are usually not barrier-free. This can be a problem in particular in the case of web applications that employees use. There are also low-barrier versions of CAPTCHAs. Examples of this are text CAPTCHAs that present easily solvable arithmetic problems or knowledge questions (“the number 12 plus the number 6 equals?”). Alternatively, an audio CAPTCHA can be provided in addition to the image CAPTCHA. So that this audio CAPTCHA cannot be solved via voice recognition, a disguised voice or acoustic backdrop with background noise is usually used.

Motivation: Attackers create accounts en mass in order to abuse these. Spammers automate the registration of e-mail addresses, for example, and cause the operator major damage when using these accounts.

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

ID: 3.06-37/7.0

Req 38	User accounts must be protected with at least one authentication attribute.

All user accounts in a system must be protected against unauthorized use.

For this purpose, the user account must be secured with an authentication attribute that enables the accessing user to be unambiguously authenticated. Common authentication attributes are e.g.:

- passwords, passphrases, PINs (factor KNOWLEDGE: "something that only the legitimate user knows")
- cryptographic keys, tokens, smart cards, OTP (factor OWNERSHIP: "something that only the legitimate user has")
- biometric features such as fingerprints or hand geometry (factor INHERENCE: "something that only the legitimate user is")

The authentication of users by means of an authentication attribute that can be faked or spoofed by an attacker (e.g. telephone numbers, IP addresses, VPN affiliation) is generally not permitted.

In companies of Deutsche Telekom group where the MyCard or a comparable smartcard is available this should be a preferred authentication attribute.

If the system and the application scenario support it, multiple independent authentication attributes should be combined if possible in order to achieve an additional increase in security (so-called MFA or Multi-Factor-Authentication).

Motivation: User accounts that are not protected by appropriate authentication attributes can be abused by an attacker to gain unauthorized access to a system and the data and applications stored on it.

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

ID: 3.01-20/8.0

Req 39	Privileged user accounts must be protected with at least two authentication attributes from different factors.

A privileged user account is a user account with extended authorizations within a system. Extended authorizations enable access to configuration settings, functions or data that are not available to regular users of the system. In direct dependence on the special tasks that are carried out via a privileged user account within a system, the assigned extended authorizations can be specifically restricted or include completely unrestricted system access.

Examples of privileged user accounts:
- Accounts for administration, maintenance or troubleshooting tasks
- Accounts for user administration tasks (e.g. creating/deleting users; assigning permissions or roles; resetting passwords)
- Accounts that are authorized to legitimize, initiate or prevent business-critical processes
- Accounts that have access to data classified as SCD (Sensitive Customer Data) in the interests of Group Deutsche Telekom, its customers or the public
- Accounts that have extensive access to data defined as "personal" according to the EU-GDPR (e.g. mass retrieval of larger parts or the complete database)

A single authentication attribute for privileged user accounts with their extended authorizations is usually no longer sufficient.

In order to achieve an adequate level of protection, at least two mutually independent authentication attributes must be used. The authentication attributes must come from various factors (knowledge, ownership, inherence). A combination of authentication attributes of the same factor (e.g. two different passwords) is not permitted

This approach is commonly referred to as MFA (Multi-Factor Authentication).
A specific form of MFA is 2FA (2-factor authentication), which combines exactly two authentication attributes.

Motivation: Privileged user accounts represent an increased risk to the security of a system. If an attacker successfully compromises such a user account, he receives extensive authorizations with which he can bring the system or system parts under his control, disrupt system functions, view/manipulate processed data or influence business-critical processes. The combination of multiple authentication attributes of different types significantly minimizes the risk of a user account being compromised.

Implementation example: Very popular is 2FA in a variant consisting of an attribute that the user knows (factor KNOWLEDGE) and an attribute that the user possesses (factor OWNERSHIP).
Examples of such a 2FA are:
- smartcard (e.g. MyCard) plus PIN

- private key plus passphrase
- classic password plus hardware token for the generation of OTPs

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

ID: 3.01-21/8.0

Req 40	The web application must request a new authentication of the user, if the user wants to change critical data or trigger critical actions. In particular for a password change the web application must request that the user enters the current password correctly (and enters the new password twice).

Critical actions are, among other things, triggered, if the user wants to change the e-mail address used for the password reminder, to close his account, to change critical information of a user profile (for example, the address for dispatch) or to change to a role with additional rights.

It is also recommended to inform the user when such critical actions are performed or when other security-critical events occur (such as multiple incorrect login attempts).

Motivation: Asking for the password again prevents an attacker who has taken over a user's session from changing the password or any other critical data as well.
Entering the new password twice prevents a one-off mistake from causing a user to be locked out and having to request a new password.

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

ID: 3.06-40/7.0

7. Protecting Sessions

Req 41	The length of the web application's session identifier must be at least 120 bit. All the relevant characters must be generated at random.

Character strings of more than 36 digits [0...9] or character strings consisting of more than 20 upper case letters [A...Z], lower case letters [a...z] and numbers [0...9], for example, meet the requirements.

Motivation: An attacker can attempt to determine valid session identifiers by means of statistical analyses or brute force attacks. If this is successful, the attacker can take over the victim's session. This can be prevented by using random, complex session identifiers for the web application.

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: 3.06-41/7.0

Req 42	If stateless session/access tokens (such as JSON Web Tokens, JWT) are used, the web application must prevent manipulation, replay attacks or other types of illegitimate use by taking appropriate measures.

In the case of JWT, for example, they must be integrity-protected via secure cryptographic methods and algorithms, preferably using cryptographic signatures. The recipient of the JWT must check the integrity according to its own configuration/application logic (hence, not based on the header information of the JWT) and in particular not accept any unsecured JWT with "{"alg": "none"}". The validity period must also be as short as possible. And the tokens must always be protected by TLS.

Motivation: It must be prevented that an attacker can misuse such tokens. It is therefore necessary that the tokens are transmitted securely and verified restrictively. However, it should be noted in this context that solutions based on JWT may be comparatively easy to implement, especially to realize "stateless" services, but are not suitable for all types of web applications, especially because they are associated with some problems, such as the question of storing the tokens in the browser or the realization of a restrictive server-side timeout and logout mechanism.

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: 3.06-42/7.0

Req 43	To transmit session identifiers or stateful/stateless (session) tokens, the web application must not use URL parameters.

Such identifiers or tokens may only be transmitted outside the URL, that is in a (session) cookie or, alternatively, in another HTTP header, for example.

Motivation: Session cookies offer the advantage that important security-relevant properties can be easily defined by the web application and are guaranteed accordingly by the browsers (no permanent storage, secure transmission, no script access, restricted validity range). If other mechanisms are used, these properties must be ensured by the respective implementation.

The use of URL parameters is not permissible: In this case, it cannot be ruled out that a user copies a used URL including a valid identifier and passes it on. Furthermore, there is a risk of the identifier being sent via the "Referer" header to another web application. And last but not least, using cookies or HTTP headers minimizes the risk, that a web application is attacked by means of session fixation: For an attack of this type, the attacker first establishes a session, and then foists the identifier in question on a victim. This is usually done by means of a link that contains the identifier as a URL parameter. If the victim uses this identifier and then logs on into the web application, the attacker can take on this identity in the established session.

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: 3.06-43/7.0

Req 44	A session identifier or stateful/stateless (session) token must not be stored in the browser persistently.

Motivation: In case of session cookies a web application can set an expiry date for a cookie via the "expires" or "max-age" attribute. But then this is a "persistent cookie". The browser stores persistent cookies on the device of the user until the expiry date. Every person with access to the device can read the persistent cookies. This can happen in an Internet café, in other cases of shared use, or through a successful attack. Cookies without an expiry date are not persistent, meaning they are not permanently stored. The browser deletes these cookies as soon as it is closed.

If session cookies are not used, the browser's session storage can be used, for example, as this is also not persistent, and the content is deleted when the browser is closed. But it must be noted that a new session is created, when a website is opened in a new tab or a new browser window. Using the local storage of the browser, however, the identifiers would be stored persistently.

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: 3.06-44/7.0

Req 45	The web application must set the attribute "secure" in the session cookie.

In addition, it is recommended to use cookie names with the prefix "__Host-" (or alternatively "__Secure-") to protect the integrity of a cookie.

If the web application consistently sets the HTTP response header "Strict-Transport-Security" and thus ensures that the browser sends encrypted requests only, it is no longer mandatory to set the attribute "secure". However, it is still recommended as an additional security measure.

Motivation: The attribute "secure" prevents the browser from sending cookies unencrypted. This happens, for example, with unencrypted contents of a web application. However, this can also happen through an active attack in which an attacker injects or presents unencrypted links or references. So even if a page is only accessible via HTTPS, a page controlled by the attacker could cause the victim's browser to make an unencrypted request. An attacker could intercept this request and would have access to the cookie. If the "secure" attribute is set, this attack is unsuccessful: the browser does not add appropriately secured cookies to unencrypted HTTP requests.

The cookie prefixes protect against cookie injection or cookie clobbering: Even an unencrypted HTTP response can set a "secure" cookie. For example, a man-in-the-middle attacker could therefore set or modify "secure" cookies. Or an

attacker could exploit problems with subdomains, e.g. if he controls evil.example.de and sets a cookie with "domain=example.de" for requests to this subdomain or if he finds an XSS on insecure.example.de and sets a cookie with "domain=example.de". Additional rules for browser behavior can be determined via a prefix in the cookie name (if the browser supports this function): If the cookie name has a prefix "__Secure-", a compatible browser will only set this cookie if it is set by an HTTPS response and the "secure" flag is set (example: Set-Cookie: __Secure-ID=123; Secure; Domain=example.com). If the cookie name has a prefix "__Host-", a compatible browser will only set this cookie if it is set by an https response, the "secure" flag is set, the "path" flag is set to root ("/") and the "domain" flag is not set (example: Set-Cookie: __Host-ID=123; Secure; Path=/).

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: 3.06-45/7.0

Req 46	The web application must set the “HttpOnly” attribute in the session cookie.

Motivation: The “HttpOnly“ attribute does not enforce (as the name suggests) that a cookie is transmitted via HTTP only. Indeed, it prevents the access for JavaScript via the "document.cookie"-API. Therefore, it prevents XSS attacks on the cookie.

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: 3.06-46/7.0

Req 47	The web application must not set the “domain” attribute in the session cookie.

The “domain” attribute specifies the domain name for which the cookie is valid. The browser sends the cookie with all requests that the browser sends to this domain and its subdomains. Not setting this attribute is the most restrictive setting – in this case, the host name of the server that set the cookie is used as the default value.

The “domain” attribute might be set however, if no other web applications run on the specified domain and on all its subdomains, so the session cookie is prevented from being sent to third party web applications, too.

Additionally, it is generally recommended that web applications with different levels of criticality or security are also run under different domains. Otherwise vulnerabilities in one web application may also endanger the security of the other web applications.

Motivation: Restrictive use of the “domain” attribute prevents the session cookie from being sent to other web applications.

If, for example, the web application xyz.telekom.net sets a cookie without a "domain" attribute, the cookie is then sent in all requests to xyz.telekom.net and its subdomains (such as abc.xyz.telekom.net). However, if the web application sets the "domain" attribute to “telekom.net”, every web application in a subdomain of telekom.net will receive the cookie.

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: 3.06-47/7.0

Req 48	The web application must set the “path” attribute in the session cookie so restrictively that the cookie is not sent to other web applications on the same host.

Motivation: Web applications can have the same host name but be in different directories. It is therefore important to ensure that different web applications on the same host do not receive the cookies for the respective other applications. If the “path" attribute is specified when setting a cookie, it is only valid in this directory and all subdirectories. The “path” attribute must therefore be set so that no other web application receives the session cookie.

If, for example, a web application under telekom.net/myapplication/index.jsp sets a cookie with the path specification “;path=/”, the cookie is then sent in all requests to the telekom.net domain and potentially also to other, less trustworthy applications that were placed in the root or any other directories.
However, if the path specification is set to “;path=/myapplication/”, the cookie is sent only for requests to telekom.net/myapplication/ (and to subdirectories, but not to higher level directories). The concluding slash sign must not be missed out, as otherwise the cookie will also be sent to other directories with matching names, like telekom.net/myapplication-exploited.

If no path specification is given, the browser uses the path of the current HTTP request based on which the cookie was set as the default.

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: 3.06-48/7.0

Req 49	Only one session must be active for a user account at any one time.

A second login with the same user account must be prevented. Alternatively, a second login can be permitted; in this case, the first session must be terminated. This variant can be reasonable to prevent a user account from being temporarily blocked, for example, due to the browser being closed or crashing without the user first logging out.

It is recommended that the web application shows the user a warning message when he logs in but a session for this user account is already in progress. This increases the probability that attacks on accounts will be detected.

However, there are web applications that are explicitly designed for access via various channels (web, mobile, TV) or permit multiple logins for other reasons. But in such exceptional cases, it is recommended that the user then has the option of deliberately terminating the other parallel sessions via a corresponding function, for example, in the case of a password change.

Motivation: If several sessions are active simultaneously for a user account, this may mean that different users are using the account at the same time or that a successful attack is taking place.

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

ID: 3.06-49/7.0

Req 50	The web application must have a function that allows a signed in user to logout at any time.

Motivation: A user must have the possibility to protect a session and therefore his data against unauthorized access. A

specific logout can be used to end a session in order to ensure that this session cannot be continued by an unauthorized person.

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: 3.06-50/7.0

Req 51	If a user accessed the web application by means of a single sign-on procedure (SSO) and then logs out of the web application, both sessions must be terminated – the session with the web application and the session with the SSO portal.

If the session with the SSO portal cannot be ended automatically, as an alternative the SSO session must be shown (again) in the user's browser on logging out of the web application. This can be implemented, for example, by means of a redirect to the SSO portal after logging out.

Motivation: SSO means that a user is able to use additional applications following one-off authentication on an SSO portal without having to re-authenticate himself to these applications. If, on logging out of an application, only the session with the application is invalidated and a session with the SSO portal that continues to be valid is not displayed, in most cases the user will not log out of the SSO portal. The session with the SSO portal remains valid and an attacker could possibly take it over without being noticed.

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: 3.06-51/7.0

Req 52	If a user accessed the web application by means of a SSO procedure and then logs out of the SSO portal, the session with the web application must also automatically be terminated.

For this purpose, the SSO portal must initiate a termination of the web application session as well.

Motivation: When the user logs out of the SSO portal, he may not realize that other applications are still in use. If the sessions with these applications are not terminated automatically, an attacker could possibly take them over without being noticed.

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: 3.06-52/7.0

Req 53	After a user is inactive for a specified amount of time, a timeout must occur in the web application and the user must be automatically logged out of the web application.

The precise amount of time after which a timeout occurs must be specified individually for each application. It depends on the application's sensitivity and purpose. For critical web applications accessible from the Internet and with access to data with need of protection (where, for example, a user is supposed to access personal data of different customers), it is recommended that the timeout occurs after 60 minutes at the latest, whereas for internal web applications without access to data with need of protection, this period could be chosen to be significantly longer. It is also recommended that this time period be a configurable system parameter.

Motivation: The timeout protects the user if he forgets to log out and inactive sessions can be taken over by an attacker without being noticed (due to vulnerabilities of the web application or due to shared use of the browser). A 60-minute timeout does not generally inconvenience users, but it does significantly reduce the risk of session hijacking. The reasons for choosing the specific timeout period may change during operation, in some cases at short notice. This situation can be sorted out with a configurable parameter.

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: 3.06-53/7.0

Req 54	When a user logs out or a timeout occurs, the web application must invalidate the corresponding session identifier or stateful/stateless (session) token on the server side.

Motivation: If a session is not completely invalidated on the server side, an attacker can continue an open session if he gains access to this session. This is possible, for example, if the attacker uses the same computer as the victim or if he has determined the session ID using a different attack.

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: 3.06-54/7.0

Req 55	If the session management of the web application is based (exclusively) on session cookies, the web application must use a mechanism against attacks by means of Cross-Site Request Forgery (CSRF). This mechanism must prevent an attacker from placing requests on a website that he controls, which (in case the user is accessing that website) would trigger valid actions of the user for the web application to be protected.

In particular, requests in the web application that can cause a data change or status change must be protected accordingly. Requests that merely request information do not need to be protected against CSRF.

Generally, “CSRF tokens” (also known as synchronizer tokens) are incorporated as a hidden field for this. A CSRF token must be generated on the server side, must be unpredictable and must be individual at least for every session. The web application must check on the server side for all relevant requests whether the token has been transferred and its value matches the expected value before the requested action is performed.
Many frameworks provide corresponding protection mechanisms. In this case, it is usually advisable to use them instead of implementing a corresponding solution yourself.
If implementation as a hidden field is not possible or reasonable, the CSRF token can be transmitted in a specific HTTP header (for example, “X-CSRF token”).
However, information that the browser sends automatically, such as cookies, does not provide any protection.

As an additional protection mechanism, it is recommended to set the attribute "SameSite" (usually with the value "Lax") in the session cookie.

Motivation: Requests that are not protected in this way are susceptible to CSRF (also known as XSRF or session riding). With this, a victim is made to unknowingly send a prepared HTTP request. This occurs, for example, through a visit to a malicious website that contains a relevant link to another web application (for example, as an "img", "script" or "iframe" tag). However, the victim cannot recognize this link. But the browser follows this link in the background and sends the session cookie automatically, too, as long as the user has an active session for this other web application.

If, for example, a user of a web application can transfer money
`https://example.com/app/transferFunds ?amount=1500 &destinationAccount=4673243243`
and the corresponding request does not include a secret token, an attacker can prepare a request that will transfer money from the victim's account to the attacker's account. The attacker embeds this request in an image tag and stores it on a site under his control.
`<img src="https://example.com/app/transferFunds ?amount=1500 &destinationAccount=attackersAcct#" width="0" height="0" />`
If the victim visits this prepared site, while already authenticated to example.com, the forged request will include the user's cookie and will therefore be executed.

However, if every HTTP request includes an unpredictable token, an attacker cannot prepare a valid request. A CSRF attack is then no longer possible.

To ensure comprehensive protection against CSRF, a different CSRF token is to be used for each individual request in a web application that causes data to be modified. This prevents intercepted tokens from being used for a CSRF attack. However, some frameworks do not support CSRF tokens that are individual for each request. This variant may also lead to problems using the web application, for example when working in several tabs or windows in parallel.

By setting the "SameSite" attribute in the session cookie, the browser is instructed not to send the session cookie with a cross-site request. This basically prevents CSRF attacks (if the browser supports this function). If the value "Strict" is set, the cookie is not sent with any such request; however, this can have undesired effects on browser behavior, for example, if the user selects a link to another web application and the session cookie for this web application is not sent, so that a user who was originally logged in has to log in again. If the value "Lax" is set, in the case of cross-site requests the cookie is only sent for HTTP methods that are considered secure (GET) and for top-level navigation (hence, not for inline frames, image tags or similar). Only for a web application that is stringently implemented regarding the HTTP methods will the value "Lax" therefore also provide sufficient protection against CSRF attacks.

With another CSRF variant, login-CSRF, the victim does not have to have a currently valid session. In this case, the attacker forges a login request for the vulnerable web application, using the attacker's credentials in this forged request. If the attack is successful, the victim is logged in with the attacker's account. All subsequent requests are then made with the attacker's account. One scenario where such an attack can be interesting is an unintentional login to a search engine with the consequence that the attacker can track the victim's subsequent search requests.

A logout-CSRF attack, on the other hand, could lead to DoS scenarios, for example.

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

ID: 3.06-55/7.0

Req 56	The web application must use a mechanism against clickjacking attacks which prevents the web application from being embedded by other unauthorized web applications.

This is usually achieved by outputting the HTTP header “X-Frame-Options” in the web application's responses. This specifies whether the contents of the web application may be displayed within a frame (that is a frame or iframe element) of another web application. The header is set to “DENY” or “SAMEORIGIN”.

If it is not possible to use this header, clickjacking can also be prevented via the Content Security Policy (CSP) and its directive "frame-ancestors". When using the CSP header, in addition to the values 'none' and 'self', it is also possible to

specify specific domains (incl. wildcards) to allow dedicated domains to embed the web application.

In particular, forms und functions in the web application that can cause a data change or status change must be protected accordingly. However, it is recommended to also protect pages that only present information from being displayed within a frame, as this is associated with side effects that can lead to further attack scenarios.

Motivation: Clickjacking (a contracted form of click hijacking, also known as UI redressing) is an attack technique in which an attacker tricks a victim into clicking on elements the victim never intended to click. The attacker inserts parts of the legitimate application into his page, usually by means of an inline frame, but masked or hidden by other elements of his own page. The victim is then led to click on content that is supposedly on the attacker's page. However, in reality this executes a click and an action on the legitimate web application.

Further attack scenarios for web applications that can be displayed within a frame result, for example, from the fact that in the case of Internet Explorer, the "document mode" is inherited by the page in the frame. Such a forced "downgrade" of the rendering engine by the attacker can also reactivate vulnerabilities (above all, for XSS) that have only been fixed in newer versions of Internet Explorer.

The “X-Frame-Options” HTTP header prevents a page from being displayed in a frame. If the element is set to “DENY”, the content is generally prevented from being displayed in a frame. The “SAMEORIGIN” value restricts the display in frames to pages within the same domain in which the web application is located.

The HTTP header "Content-Security-Policy" can also be used to prevent the content from being displayed in a frame. If the directive "frame-ancestors 'none' " is specified, the content is generally prevented from being displayed in a frame. The directive "frame-ancestors 'self' " restricts the display in frames to pages within the same domain where the web application is located. However, since one or more domains (including wildcards) can also be specified in the directive, which can then display the web application within a frame, this variant is more flexible than the "X-Frame-Options" HTTP header. However, the directive is not yet supported by all popular browsers.

JavaScript code might also be used to check whether a page containing the code is actually the top page in the frame hierarchy, to then prevent the display within a frame. Unfortunately, various techniques are known which get past the common variants of these JavaScript mechanisms.

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

ID: 3.06-56/7.0

Req 57	Authorizations for cross-domain access must be defined restrictively.

The web application may allow solely such domains cross-domain access, for which this is explicitly provided and absolutely required. Cross-domain access to content of the web application with need of protection must not be permitted. In particular, cross-domain access to the content of authenticated sessions must not be permitted. Intranet applications must not grant authorizations for cross-domain access to domains that do not belong to the intranet.

Cross-Origin Resource Sharing (CORS) is a mechanism to allow exceptions to the rules of the Same Origin Policy (SOP) and to implement communication between web applications via the user's browser. CORS enables the browser to initiate cross-origin requests and to allow the response to be read. These requests are made, for example, by means of XMLHttpRequests (XHR). The mechanism is based on the exchange of header information. Browsers send the "Origin" header, which indicates the domain of the application that sends the request. Servers in turn send back the "Access-Control-Allow-Origin" header to express which domain is allowed to access their resources. This is then ensured by the browsers accordingly.
Authorization for such cross-domain access must therefore be defined restrictively without the use of wildcards in the "Access-Control-Allow-Origin" header. If the value of the "Access-Control-Allow-Origin" header is generated dynamically (possibly depending on the respective "Origin" header of the request), the "Origin" header must be validated, permitted domains must be defined restrictively and additionally the "Vary: Origin" header must be set in order to prevent attacks based on cache poisoning. The headers may also only be sent by the server for selected URLs/resources for which cross-domain access is intended. They may not be sent across the board for the entire application.

The "window.PostMessage"-API expands every window and every frame with a new method that allows text messages to be sent from one window to another. In that case, the “recipient” of the data must be explicitly defined in order to avoid the data being sent to an incorrect destination. In particular, the recipient must not be set to *. The “sender”

(“origin” attribute) must be verified. The data (“data” attribute) must be validated. The exchanged data must be evaluated as data, never as code (for example, using "eval"). To avoid DOM-based XSS attacks, the DOM of a page must not be modified based on the exchanged data (for example, by means of “innerHTML”).

The web application must not use any RIA services that circumvent the SOP. This includes AJAX service bridges, HTML bridges, AJAX proxies and aggregate sites. Workarounds and deprecated functions must also not be used to enable cross-domain access. This applies, in particular, to fragment identifiers (the part of a URL after the hash symbol), to JSONP (JSON with padding) and, as well, to the property "document.domain", by which a web application could change its origin (to a parent domain). Generally, standardized technologies should be used instead of proprietary technologies or workarounds.

All requirements regarding cross-domain interaction apply equally to windows, dialog boxes, frames, etc.

Motivation: The SOP is a security concept implemented in the browsers. It aims to ensure that each resource of a website may only communicate with the server from which it was loaded and may only access objects which were loaded from the same server. Cross-domain access is thus prevented.

However, there are configuration options for individual web technologies that relax the SOP and allow cross-domain access. This is dangerous because it allows other web applications to execute code in the context of the domain of our own web application. A restrictive definition of authorizations which may be necessary can reduce the associated risk. If cross-domain access is granted too freely, it may even be possible to access data with need of protection that is otherwise protected by authentication. This is because when cross-domain requests are sent, browsers send these requests together with the cookies for the domain of the web application to which these requests go. Their responses can be evaluated on the client side and content with need of protection can possibly be accessed. And also the basic protection against access from the Internet is overturned if an application that can, in fact, only be reached from the intranet allows cross-domain access from other domains.

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

ID: 3.06-57/7.0

8. Authentication Parameter Password

Req 58	If a password is used as an authentication attribute, it must have at least 12 characters and contain three of the following categories: lower-case letters, upper-case letters, digits and special characters.

A system may only accept passwords that comply with the following complexity rules:

- Minimum length of 12 characters.
- Comprising at least three of the following four character categories:
    - lower-case letters
    - upper-case letters
    - digits
    - special characters

The usable maximum length of passwords shall not be limited to less then 25 characters. This will provide more freedom to End Users when composing individual memorizable passwords and helps to prevent undesired behavior in password handling.

When a password is assigned, the system must ensure that the password meets these policies. This must be preferably enforced by technical measures; if such cannot be implemented, organizational measures must be established.
If a central system is used for user authentication [see also Root Security Requirements Document[i] "3.69 IAM (Identity Access Management) - Framework"], it is valid to forward or delegate this task to that central system.

Permissible deviation in the password minimum length
Under security-related criteria, conditions can potentially be identified for a system that enable the minimum password length to be reduced:

- It is generally permissible to reduce the minimum password length for systems that use additional independent authentication attributes within the authentication process in addition to the password (implementation of 2-Factor or Multi-Factor Authentication).
- Any reduction in the minimum password length must be assessed individually by a technical security advisor (e. g. a PSM from Telekom Security) and confirmed as permissible. In the assessment, the surrounding technical, organizational and legal framework parameters must be taken into account, as well as the system-specific protection requirements and the potential amount of damage in the event of security incidents.
- The absolute minimum value of 8 characters length for passwords must not be undercut.

Motivation: Passwords with the above complexity offer contemporary robustness against attacks coupled with acceptable user friendliness. Passwords with this level of complexity have proven their efficiency in practice. Trivial and short passwords are susceptible to brute force and dictionary attacks and are therefore easy for attackers to determine. Once a password has been ascertained it can be used by an attacker for unauthorized access to the system and the data on it.

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

ID: 3.01-26/8.0

Req 59	If a password is used as an authentication attribute for technical accounts, it must have at least 30 characters and contain three of the following categories: lower-case letters, upper-case letters, digits and special characters.

Technical user accounts are characterized by the fact that they are not used by people. Instead, they are used to authenticate and authorize systems to each other or applications on a system.

A system must only use passwords for technical user accounts that meet the following complexity:

- Minimum length of 30 characters
- Comprising at least three of the following four character categories:
    - lower-case letters
    - upper-case letters
    - digits
    - special characters

Motivation: Due to their use in machine-to-machine (M2M) communication scenarios, technical user accounts are often equipped with privileges that can be of high interest to an attacker to compromise infrastructures. Without mechanisms of extensive compromise detection, the risk of a password being determined or broken by an attacker can increase significantly over time. A significant increase in password length counteracts these risks and can also be implemented particularly easily in M2M scenarios, since handling a very long password is not a particular challenge for a machine (as opposed to a person).

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

ID: 3.01-27/8.0

Req 60	If a password is used as an authentication attribute, the registration process for the web application must ensure that the user chooses his password on his own: Either the user sets his password during registration, or he receives an individual initial password that he must change immediately after logging in for the first time.

We recommend that the web application informs users about the criteria for passwords. In addition, the web application may display an indicator of the password strength (“password meter”). Alternatively, the web application may visualize, which criteria of the password policy are met by the password.

Motivation: Predefined passwords are frequently not treated with care by the users, possibly even noted down, as they are usually not both secure and easy to remember. In addition, initial passwords are frequently transmitted in unencrypted format or set by third parties within the framework of support processes. To reduce the risk of misuse, new users must change these passwords immediately.
Transparent criteria for selecting strong passwords increase users' acceptance and awareness of security.

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

ID: 3.06-60/7.0

Req 61	If a password is used as an authentication attribute, users must be able to independently change the password anytime.

The system must offer a function that enables a user to change his password at any time.

When an external centralized system for user authentication is used, it is valid to redirect or implement this function on this system.

Motivation: The fact that a user can change his authentication attribute himself at any time enables him to change it promptly if he suspects that it could have been accessed by a third party.

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
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-29/8.0

Req 62	If a password is used as an authentication attribute, it must be changed after 12 months at the latest.

The maximum permitted usage period for passwords is 12 months.
If a password reaches the maximum permitted usage period, it must be changed.

For this purpose, the system must automatically inform the user about the expired usage period the next time he logs on to the system and immediately guide him through a dialog to change the password. Access to the system must no longer be permitted without a successfully completed password change.
For technical user accounts (M2M or Machine-2-Machine), which are used for the authentication and authorization of systems among themselves or by applications on a system, automated solutions must also be implemented to comply with the permitted usage period for passwords.

Alternatively, if such an automatic mapping of the process for changing the password cannot be implemented, an effective organizational measure must be applied instead, wich ensures a binding manual password change at the end of the permissible period of use.

Motivation: Unlike more modern authentication attributes, passwords are easier to attack. Without specific measures for reliable, technically automated detection of compromises, the risk of a password being discovered or broken by an attacker can increase considerably over time.

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
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-30/8.0

Req 63	If a password is used as an authentication attribute, the reuse of previous passwords must be prevented.

A history of the previously used passwords must be recorded for each user account. When a password change is initiated for a user account, the new password must be compared with this password history. If the reuse of a password is detected, the password change must be rejected. This validation process must be implemented in the system on the

basis of technical measures. If a central IAM system is used for user authentication, the implementation can be forwarded to the central IAM system or outsourced there [see also Root Security Requirements Document[i] "3.69 IAM (Identity Access Management) - Framework"].

In general, the password history should ensure that a password that has already been used can never be used again.

However, due to technical limitations, a password history cannot be recorded indefinitely in many IT/NT products. In this case, the following basic rules must be observed:

- a password that has already been used must not be reusable for a period of at least 60 days (measured from the point in time at which the affected password was replaced by another)
- in systems in which the period of at least 60 days cannot be implemented, the longest possible period must be configured. In addition, it must be confirmed by a Project Security Manager (PSM) that the configured period is still sufficient in the overall context of the system with regard to the security requirement.

Annotation:
Some IT/NT products do not offer any technical configuration parameters with which the password history can be linked directly to a time period, but only allow the definition of the number of passwords to be recorded. In such cases, the time period can alternatively be ensured by linking the following, usually generally available configuration parameters. Within the resulting policy, a user can only change his password once a day and, due to the number of passwords recorded, can reuse an old password effectively after 60 days at the earliest.

- Minimum Password Age: 1 day
- Password History: Record of the last 60 passwords used

With this implementation variant, it should be noted that the minimum age for the password should not be more than one day in order not to inappropriately restrict the user with regard to the fundamental need to be able to change the password independently at any time.

Motivation: Users prefer passwords that are easy to remember and often use them repeatedly over long periods of time when the system allows. From the user's point of view, the behavior is understandable, but effectively leads to a considerable reduction in the protective effect of this authentication parameter. With adequate knowledge of the user or information obtained from previous system compromises, an attacker can gain access to supposedly protected user accounts. Particularly in situations in which new initial passwords are assigned centrally as part of an acute risk treatment, but users change them immediately to a previous password for the sake of simplicity, there is a high risk that an attacker will resume illegal access. It is therefore important to prevent users from reusing old passwords.

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
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-28/8.0

Req 64	If a password is used as an authentication attribute, the web application must have a function that is protected against misuse, by means of which a user can reset the password.

Misuse that leads to the existing secure authentication mechanisms being bypassed via this function must be prevented.
For this, we recommend that users must select a “security question" and enter the correct answer (for critical applications possibly several questions and answers). The web application must request this data as part of the registration process or after the user's initial login.
If the user has forgotten his password, there is an appropriate function to select when logging in. Once the user has answered the relevant security question correctly, the password is reset. The web application creates a new initial password or a link with an activation token. It can send the initial password or activation token to the user via push noti-

fication or SMS, alternatively via e-mail, to predefined contact information.
This function must in no way allow the user to successfully log in to the web application or to directly change his password, just by answering the security question.
It must also be prevented that this function offers an attacker the possibility to automatically determine valid usernames. For this purpose, CAPTCHAs can be queried or threshold values for invalid entries/attempts can be provided, which lead to a (temporary) blocking of this functionality (for example, for an IP address).
In the case that a user has also forgotten the answer to his security question, a fallback mechanism can also be provided, if required. For this, the user is offered a further process, but which can usually not be performed automatically, for example calling a hotline. Also, for such a fallback mechanism, it is important to ensure that it cannot be misused.

Motivation: If a user has forgotten his password, it must be possible to reset the password. However, this must not lead to a reduction in the existing security level of the web application. Attackers must be prevented from finding a possibility to determine the (initial) passwords of other users and taking over their accounts. It also should not be possible for a password reset process to be (automatically) misused, for example, by resetting passwords en mass and bothering other users.

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

ID: 3.06-64/7.0

Req 65	If a password is used as an authentication attribute, a protection against online attacks like brute force and dictionary attacks that hinder password guessing must be implemented.

Online brute force and dictionary attacks aim for a regular access interface of the system while making use of automated guessing to ascertain passwords for user accounts.

To prevent this, a countermeasure or a combination of countermeasures from the following list must be implemented:
- technical enforcement of a waiting period after a login failed, right before another login attempt will be granted. The waiting period shall increase significantly with any further successive failed login attempt (for example, by doubling the waiting time after each failed attempt)
- automatic disabling of the user account after a defined quantity of successive failed login attempts (usually 5). However, it has to be taken into account that this solution needs a process for unlocking user accounts and an attacker can abuse this to deactivate accounts and make them temporarily unusable
- Using CAPTCHA ("**C**ompletely**A**utomated**P**ublic**T**uring test to tell**C**omputers and**H**umans**A**part") to prevent automated login attempts by machines ("robots" or "bots") as much as possible. A CAPTCHA is a small task that is usually based on graphical or acoustic elements and is difficult to solve by a machine. It must be taken into account that CAPTCHA are usually not barrier-free.

In order to achieve higher security, it is often meaningful to combine two or more of the measures named here. This must be evaluated in individual cases and implemented accordingly.

Motivation: Without any protection mechanism an attacker can possibly determine a password by executing dictionary lists or automated creation of character combinations. With the guessed password than the misuse of the according user account is 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
- Denial of executed activities

ID: 3.01-25/8.0

Req 66	If passwords are used as an authentication attribute, those must be stored using a suitable and approved "Password Hashing" method to protect against offline-attacks like brute force or dictionary attacks.

This requirement relates to the storage of passwords in all types of user databases, as used in this system, in order to authenticate incoming access (local or remote) by users or other systems.

If an attacker obtains the copy of a user database of the system, he is able to bring it into a fully independent environment and utilize automatized dictionary or brute force attacks to determine contained passwords. Specialized tools in combination with high computing power allow for high cracking rates in a relatively short period of time, if protective measures are insufficient. Due to the independency from the source system, such an offline attack happens unnoticed.

The following countermeasure must be implemented, since this ensures best possible protection against offline attacks:

- passwords must be stored using a cryptographic one-way function (known as "Password Hashing") that is recognized as effective for this purpose and is classified as secure at the current state of the art.

Please Note: valid password hashing algorithms are described in Security Requirement Catalog "3.50 Cryptographic Algorithms and Security Protocols".

Explicitly NOT PERMISSIBLE is:

- to store passwords in cleartext
- to store passwords in any format which can be directly backcalculated
- to store passwords using reversible encryption

Please Note:
In this context, "directly backcalculatable formats" refers to those that simply encode the password, without involving a secret key in the transformation process. Since the password will no longer show up as original cleartext after it has been processed, those formats may easily be mistaken to provide confidentiality. Effectively, they do not offer any protection. The enconding is fixed and therefore an attacker can easily make use of it to compute the original cleartext password from the encoded string.
Examples for directly backcalculatable formats are: "base64", "rot13"
"Reversible" are all encryption methods which, using the appropriate key, enable encrypted content to be transformed back into the original content. Accordingly, with reversible encryption there is always the challenge of keeping the key secure and protecting it from unauthorized access. Reversibility is a required fundamental property in many areas of encryption applications, e.g. for transferring confidential messages, but it is counterproductive for storing passwords: a stored password must remain comparable by means of technical methods, but it must no longer be possible to convert it back into plain text in order to protect it as well as possible from unauthorized viewing.
Examples for reversible encryption are: "AES", "CHACHA20", "3DES", "RSA"

Motivation: Without protective measures, an attacker in possession of a user database copy is able to determine masses of contained passwords in short time by merely trying out character string combinations or making use of dictionaries. Passwords stored in cleartext or any backcalculatable format are fully defenseless to an offline attack. Once a password has been ascertained it can be used by an attacker for unauthorized access to the system and the data on it.

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

- Denial of executed activities
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-24/8.0

Req 67	If initial passwords or activation tokens are used, these must be protected against malicious use and brute force attacks.

In particular, they must lose their validity after a suitable period of time. The precise validity period must be set individually for each application. It depends on the application's sensitivity and purpose, as well as on the password assignment procedure. If, for example, in case of a web application accessible from the Internet, the user starts the registration on his own and initial passwords (or activation tokens) are sent electronically, they must lose their validity within a few hours after they are sent. In case of an internal application and transport via internal networks, this period can be significantly longer (up to several days).

Initial passwords must comply with the same requirements regarding complexity like passwords chosen by the users.

The length of activation tokens must be at least 120 bit (like the length of session identifiers). All the relevant characters must be generated at random.

Motivation: If a new user has not used the initial password after a long time, it can be assumed that there has been an error, or the user does not want to complete the registration process. Furthermore, the initial password is often still accessible (for example, in e-mail inboxes). Unauthorized persons can therefore determine the password and misuse it. Therefore, it must lose its validity after a short time.
Without any protection mechanism an attacker can possibly determine an initial password or an activation token by automated creation of character combinations.

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

ID: 3.06-67/7.0

Req 68	If a user's attempt to log in fails, the web application must not give any information regarding which of the login information entered was incorrect.

Motivation: A general error message makes it more difficult for attackers to find valid usernames.

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

ID: 3.06-68/7.0

Req 69	If passwords are used as an authentication attribute, they must not be displayed in plain text during input.

Passwords must not be displayed in legible plain text on screens or other output devices while they are entered. A display while entering must not allow any conclusions to be drawn about the characters actually used in the password.

This requirement applies to all types of password input masks and fields.
Examples of this are dialogs for password assignment, password-based login to systems or changing existing passwords.

Exceptions:

- Within an input field, an optional plain text representation of a password is permitted, provided that this plain-

text representation serves a valid purpose, exists only temporarily, has to be explicitly activated by the legitimate user on a case-by-case basis and can also be deactivated again immediately by the latter.
A valid purpose would be, for example, to allow the legitimate user an uncomplicated visual check, if necessary, that he has entered the password correctly in a login dialog before finally completing the login.
Such an optional plain text representation of a password must remain fully in the control of the legitimate user so that he can decide on its activation/deactivation according to the situation. In the default setting of the system, the plain text representation must be deactivated.

- The typical behavior on many mobile devices (smartphones) of displaying each individual character very briefly in plain text when entering a password - in order to make it easier for the user to control input - is fundamentally permissible there. However, the full password must never be displayed in plain text on the screen.

Motivation: In the case of a plain text display, there is a risk that third parties can randomly or deliberately spy on a password via the screen output while typing.

Implementation example: When displayed on the screen, each individual character is uniformly replaced by a "*" while entering a password.

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
- Attacks motivated and facilitated by information disclosure or visible security weaknesses

ID: 3.01-31/8.0

9. Content Management Systems (CMS)

Req 70	Access to the editing environment of the content management system (CMS) must not be via the Internet.

The editing environment must either be accessible from internal networks only or access restriction must be implemented by means of a technical solution, such as VPNs.
If protection via access restriction cannot be implemented, the access to the editing environment must be protected by means of another additional security level, for example, by using two-factor authentication.

Motivation: This minimizes the risk of attackers gaining unauthorized access to the CMS (for example, through known CMS vulnerabilities) and modifying content or reading information that has not been published.

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

ID: 3.06-70/7.0

Req 71	The CMS must allow the authorizations for different content management activities to be assigned to different users/user groups so that a multi-stage publication process can be implemented.

Possible different user groups are "reader", "editor", "chief editor", "administrator", "approver", and "publisher". Publication of content must take place according to at least a dual-control principle by an "editor" and an "approver" or "publisher".

Motivation: An appropriate publication process can minimize the risk of undesirable or incorrect content being published (through malicious intent or due to an error).

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: 3.06-71/7.0

Req 72	The CMS must allow certain content to be assigned exclusively to particular editors or a group of editors.

It must then not be possible for other, non-authorized editors or groups to view or modify this content. This can be achieved by allowing the CMS to serve several clients, or by means of a suitable role concept.

Motivation: This enables unpublished (confidential) content to be recorded without it being revealed to all editor groups. It also minimizes the risk of content being modified without authorization.

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: 3.06-72/7.0

Req 73	If the CMS provides functions for creating content that is to be published later, it must not be possible to view or find this content in the web application before the publication date.

Likewise, it must not be possible to find the unpublished content using the web application's search functions, secret URLs or through active manipulation.

Motivation: This enables unpublished (confidential) content to be edited without being viewed in advance by web application users.

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

ID: 3.06-73/7.0

Req 74	The CMS must provide functions that make it possible to restrict the use of active content and scripting within the created content (to specific content or specific user groups) and, if necessary, to prevent it altogether.

Motivation: Content that could potentially pose a risk for users of the web application should be used only when necessary and only be created by editors who are explicitly responsible for this. For example, if script languages are used, content with XSS attacks could be created.

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

ID: 3.06-74/7.0

Req 75	Preview functionalities of the CMS must be protected from unauthorized access – only successfully authenticated CMS users should be able to view the preview content and, in doing so, they may have access only to the content for which they have as a minimum read rights.

Motivation: This enables unpublished or incomplete content to be entered and verified without it being viewed in advance by web application users or other non-authorized editors.

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

ID: 3.06-75/7.0

10. Logging

Req 76	Security-relevant events must be logged dependent on the intended purpose of the web application.

So that these events can be evaluated and classified, they must be logged together with the exact time the incident occurred ("timestamp").

The timestamp of a logged event must contain at least the following information:

- date of the event (year, month, day),
- time of the event (hours, minutes, seconds),
- time zone, that information belongs to.

Logging must be done considering the currently valid legal, co-determinational and operational regulations. Following these regulations logging of events is only allowed for a defined use case. Logging for doing a work control of employees is not allowed.
In addition - as for any data that is processed by a system - an appropriate protection requirement must also be taken into account and implemented for logging data; this applies to storage, transmission and access. In particular, if the logging data contains real data, the same protection requirements must be taken into account that is also used for the regular processing of this real data within the source system.

At least following events are to be logged by a web application (provided that they are relevant for the individual case of application):

Event	Event data to be logged
Correct logins	User account, source (IP address), if available.
Incorrect login attempts	User account, number of failed attempts, source (IP address), if available.
Access with accounts with administrator rights	User account, access timestamp, length of session, source (IP address) , if available.
Account administration	Administrator account, administered user account, activity performed (configure, delete, enable and disable).
Change of group membership for accounts	Administrator account, administered user account, activity performed (group added or removed).

Depending on the individual web application further events can be reasonable, in particular:

- validation failures (for example, wrong data format, wrong encoding, wrong value range), which cannot have arisen due to input errors at normal use of the web application, but indicate attack attempts clearly,
- output failures (for example, database record set mismatch),
- authorization failures (for example, access to resources without corresponding authorization),
- privileged or security-relevant actions (for example, export of data with need of protection),
- session management failures (for example, session identifier manipulation),

- application errors (for example, runtime errors),
- detection of malware or other security issues in the case of a file upload.

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

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

ID: 3.06-76/7.0

Req 77	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 78	Security-relevant logging data must be forwarded to a separate log server immediately after it has been generated.

Logging data must be forwarded to a separate log server immediately after it has been generated. Standardized protocols such as Syslog, SNMPv3 should be preferred.

Motivation: If logging data is only stored locally, it can be manipulated by an attacker who succeeds in compromising the system in order to conceal his attack and any manipulation he has performed on the system. This is the reason why the forwarding must be done immediately after the event occurred.

For this requirement the following threats are relevant:
- Unauthorized modification of data
- Disruption of availability
- Denial of executed activities
- Unnoticeable feasible attacks

ID: 3.01-35/8.0

Req 79	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 80	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