Security requirement

Client Applications

Deutsche Telekom Group

Version	6.2
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
	3.54	Security requirement
Version	**State**	**Status**
6.2	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
The description "Client Applications" applies to every application (shortcut - "App") which is executed on mobile or desktop hardware. The App can be executed independently or in communication with a backend according to the client-server model.

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

Table of Contents

1.	Introduction	4
2.	General security requirements for apps	5
3.	Requirements during the planning and design phase	10
4.	Requirements during the development phase	16
5.	Installation and update processes	19
6.	Uninstalling requirements	20
7.	General requirements	21
7.1.	Protecting availability and integrity	21
7.2.	Protecting sessions	21
7.3.	Authentication parameter password	22
7.4.	Logging	26

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 standards in 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. General security requirements for apps

Req 1	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 2	Known vulnerabilities in the software or hardware of the system must be fixed or protected against misuse.

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

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

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

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

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

Additional information:
The primary sources of known vulnerabilities in software/hardware are lists in the release notes as well as the security advisories from the official reporting channels of the supplier or independent CERTs. In particular, the reporting channels are sensibly integrated into continuous processes of security patch management for a system, so that newly discovered vulnerabilities can be registered promptly and led into operational remedial measures.
As a complementary measure to the detection of potentially still contained types of vulnerabilities that have in principle already become known, targeted vulnerability investigations of the system can be carried out. Particularly specialized tools such as automated vulnerability scanners are suitable for this purpose. Examples include: Tenable Nessus, Qualys Scanner Appliance.

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

ID: 3.01-10/8.0

Req 3	Software components that are no longer supported by vendor, producer or developer must not be used.

Only those middleware and application software may be used on a system which are supported by the vendor, the producer, the developer (this includes open source communities) or other contractual partner of Deutsche Telekom AG. Components that have reached end-of-life or end-of-support must not be used. Excluded are components that have a special support contract. This contract must guarantee the correction of vulnerabilities over components lifetime.

Motivation: Software components that have reached end of life or end of support represent a risk for a system. This means that a vendor does not supply remedial updates or patches for a component should errors or vulnerabilities occur. This means that vulnerabilities cannot be fixed when they occur and could be exploited to compromise the system or to impair its availability.

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

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

ID: 3.54-3/6.2

Req 4	Security best practices that exist for an execution environment must be complied with.

Best-practice recommendations are suggestions for solving recurring problems. They explain how – in relation to an individual execution environment – security problems can be avoided and security functions (e.g., the sandbox principle) used. They also provide information on how to avoid typical programming errors which can lead to security vulnerabilities.

Recommendations for prevention of security vulnerabilities:

Usage of secure library functions.

The length and coding for app input data, in particular if it does not stem from a trusted source, should always be checked and not used as format strings in order to prevent buffer overflows, null-byte poisoning, format-string- or similar attacks. Untrusted data sources for apps include not only user inputs but also parts of URL, text message or email content, environment variables, locally read data (e.g., from an SD card) or data received from a server. It shall be ensured that input data is only interpreted as data and not as program code.

Prevent buffer overflows.

To avoid buffer overflows, in particular in machine-oriented programming languages such as C, C++ and Objective C, library functions shall not be used without ength validation. Self-written code which processes strings or memory contents shall explicitly check the length of the processed data.

Prevent race conditions.

Today, nearly all operating systems and execution environments, even for smaller devices, support multitasking and multithreading, i.e., they enable several processes to run in parallel. The disadvantage of this is that there is no guarantee that two consecutive operations will be executed sequentially in a program ithout other operations being executed concurrently. This may lead to errors and, in a worst case scenario, to security vulnerabilities in an app.

Motivation: Best-practice recommendations usually contain not only a lot of security knowledge but also incorporate experience from past errors and security problems. Instructions and sample source code show how the execution environment's security features can be used.

If length is not validated and non-secure library functions are used, basic programming errors may result in security vulnerabilities, which cause confidential information to be compromised or circumvent security mechanisms of the app or the execution environment (sandbox). Such security vulnerabilities may also occur in Java apps (e.g., Android or OSGi) when unchecked input data is passed to low-level (C) functions via a Java library function. Special care is necessary when JNI is used in Java apps.

Buffer overflows are one of the main causes of foreign code execution. The jailbreaking and rooting methods used on devices usually exploit buffer overflows in routines of a device's operating system. A buffer overflow vulnerability in an app may circumvent an existing sandbox model of the execution environment and execute external code in the app context.

Race conditions are not only errors that are difficult to find but can, nowadays, be the cause of major security vulnerabilities. Semaphores or flags can be used to combat this, i.e., variables are assigned to signal states which block or unblock them for changes.

Implementation example: GSMA Smarter Apps for Smarter Phones Guideline: http://www.gsma.com/technicalprojects/wp-content/uploads/2012/04/gsmasmarterappsforsmarterphones0112v.0.14.pdf

Apple iOS: http://developer.apple.com/library/ios/#documentation/Security/Conceptual/SecureCodingGuide/Introduction.html#//apple_ref/doc/uid/TP40002415

Google Android: http://developer.android.com/guide/topics/security/security.html , http://source.android.com/tech/security/index.html

ID: 3.54-4/6.2

Req 5	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 6	The app must run with unprivileged user rights only on the end user device.

Jailbreaking and rooting of mobile devices enable developers to fully exploit the device's hardware and system resources. This jeopardizes the security of apps and information on the device, since it will usually override the execution environment's sandbox model and enable full access to the device's operating system and hardware. Hence an app shall run with a non-privileged account. Requiring a privileged account enables developers to fully exploit the operating system’s features, but the user may want to run the app in a nonprivileged account as this is more secure. However, for installing the app administrative privileges might be necessary and may be requested.

Motivation: Jailbreaking and rooting make it possible to circumvent existing security mechanisms. The entire execution environment becomes insecure and apps running and information stored on it are exposed to threats. Security loopholes in an app on a PC, which runs with administrative privileges, can compromise the security of the operating system. Hence the user should have the choice to run the app in a non-privileged account.

ID: 3.54-6/6.2

3. Requirements during the planning and design phase

Req 7	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 8	If the application uses passwords for user authentication, then it must protect them during storage by storing only passwort hashes. The hashing must be done using a password hashing algorithm such as scrypt, bcrypt, or Argon2. If this is not possible, a cryptographic hash function with salt must be used.

Motivation: A secure storage prevents user passwords from being compromised and being misused. Even if an attacker gains access, he can only read password hashes. When an adequate hashing function is chosen and passwords are long enough, then the effort to infer the original passwords from the password hashes through brute forcing is so high, that this cannot be done in acceptable time. Password hashing algorithms are developed in a way that hashing is time-consuming and thus brute force attacks are slowed down.

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

ID: 3.54-8/6.2

Req 9	Applicable retention and deletion periods must be observed for security-relevant logging data that is recorded locally within applications, if they are managed by Telekom.

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. [1*]
- after 90 days, stored logging data must be deleted immediately.

[1*] 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.

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.

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 email 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: 3.54-9/6.2

Req 10	If a permanent login is required OAuth 2.0 refresh tokens or a comparable mechanism must be used.

OAuth 2.0 is the draft of an open protocol which permits standardized, secure authentication and authorization, e.g., also for mobile apps. With this protocol, a user can enable an app to access back-end systems via his specific access without disclosing a password or granting full authorization to access his data.

Motivation: OAuth 2.0 refresh tokens or comparable mechanisms (referred to in general as 'security tokens') offer greater protection for permanent login: security tokens have limited validity and extremely narrow applicability. They

are only valid for authentication to the back-end system for which they were issued, and authorizations for access can be limited to the functions of the app. Furthermore, they can be revoked, e.g., if the user loses the device. One important feature of the OAuth refresh tokens is that each time they are used, i.e., during each login process onto the back-end system, the current token loses its validity and is replaced by a new one. If, for example, a token has been compromised, it automatically loses its validity at the next login process without the password having to be changed.

Implementation example: OAuth 2.0 RFC: https://tools.ietf.org/html/rfc6749

ID: 3.54-10/6.2

Req 11	If an app offers the option of a permanent login (e.g., using OAuth refresh tokens), it must be possible for the user to enable and disable the option.

The option to permantly save the login state should be inactive as a default.

Motivation: Security-aware users and people who share a device with others shall be given the possibility to prevent permanent logins.

ID: 3.54-11/6.2

Req 12	The use of system functions that require protection as well as access to internal or confidential data must not be possible without prior authentication and authorization.

The use of functions of the system that require protection as well as access to data classified as internal or confidential must only be possible after the user has been uniquely identified and successfully authenticated by means of the user name and at least one authentication attribute. In addition, it must be verified that the user is authorized to access the affected functions and data within the user role assigned to him or her in the system.

An exception to this are functions and data that may be used publicly without restriction; for example, the area of a website on the Internet where only public information is provided.

Examples of features that require prior authentication include:

- Remote access to network services (such as SSH, SFTP, web services)
- Local access to the management console
- Local use of operating system and applications

Examples of authentication features that can be used:

- Passwords
- cryptographic keys or certificates (e.g., in the form of smart cards)

This requirement also applies without restriction to any machine access to the system (here the implementation is usually carried out by using so-called M2M - "Machine-to-Machine" - user accounts).

Motivation: The unambiguous authentication and authorization of access to a system are elementary to protect functions and data from misuse.

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: 3.01-19/8.0

Req 13	In-app-payment or -booking must be confirmed with the additional approval of the user password.

In the case of an in-app payment or -booking of additional products or product options, the payment or booking transaction may only be initiated once the user has proved his identity by entering his password or another secret known solely to the user. The payment or booking shall not be assigned to a specific user on the basis of a permanent login.

Motivation: This guarantees that no non-legitimized user can perform payment or booking transactions with the aid of a device he has found or stolen. If implemented as required here the legitimate user could not easily deny that he performed the payment or booking.

ID: 3.54-13/6.2

Req 14	The system must allow users to log out of their current session.

The system must have a feature that enables the logged-in user to log out at any time. It must not be possible to resume a logged-out session without re-authenticating the user.

Motivation: A user must retain complete control over the sessions he has established in order to be able to terminate his access to a system at any time according to the situation and thus protect data and functions exposed via this access. In addition, the user must be able to assume that sessions specifically terminated by him cannot subsequently be resumed and continued by unauthorized third parties.

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: 3.01-17/8.0

Req 15	If an app stores user-specific access credentials permanently (e.g., security tokens for back-end systems) then it must have a logout function which deletes a permanent login (logout).

Users shall have the possibility to disable permanent login and to execute full logout. Deletion of device memory (factory reset) or removal of an app is too work-intensive if a user merely wants to revoke a permanent login.

Motivation: A user could, for example, share a device with another user but not want to grant that person access to his information on the back end. Or a security-aware user who has mistakenly stored a login would like to revoke it.

ID: 3.54-15/6.2

Req 16	Outputs and messages must not disclose information on internal structures of the system.

Information about the internal structures of a system, including the components used there, and corresponding implementation details are generally considered to be in need of protection.

In general, this concerns information on
- Product names and product identifiers of implemented system components
- Operating systems, middleware, backend software, software libraries and internal applications as well as their software versions
- installed service packs, patches, hotfixes
- Serial numbers of components as well as stored product licenses
- Database Structures

Typical examples of outputs and messages in which disclosure of such system information can potentially occur:

- Login windows and dialogs
- Error messages
- Status messages
- Banners of active network services
- System logs and log files
- Debug logs, stack traces

As far as it is technically feasible without impairing the function and operation of the system, the output of affected system information must always be deactivated.

Access to affected system information must only be possible for authorized users of the system. As a rule, this circle of authorized users is to be limited to administrators and operators of the system. Access for authorized monitoring and inventory systems within the operating environment is also permitted.

A permissible exception to these restrictions exists for specific individual system information, the disclosure of which is technically mandatory for the intended function of the system in conjunction with third-party systems; For example, the presentation of supported protocols and their versions during the initial parameter negotiation in session setups between a client and a server.

Motivation: Information about the internal structures of a system can be used by an attacker to prepare attacks on the system extremely effective. For example, an attacker can derive any known vulnerabilities of a product from the software version in order to exploit them specifically during the attack on the system.

Implementation example: [Example 1]
Deactivation of the display of the product name and the installed version of a Web server in its delivered error web pages.

[Example 2]
Removal of the product name and the corresponding version string from the login banner of a deployed SSH server.

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

ID: 3.01-9/8.0

Req 17	If data with need for protection are transmitted using HTTP(S), then the POST method must be used for requests.

If the web service of a REST API ensures that repeated requests have always the same effect on a given URI, then also PUT can be used instead of POST. The GET method must only be used for retrieving content from the server. Data with need of protection include identifiers such as session IDs or the MSISDN of a mobile subscriber; these data must not be transmitted using GET requests.

Motivation: The POST method transmits parameters in the HTTP request body. This ensures that no data with need of protection are logged by web servers, security information and event management (SIEM) systems and clients, or are easily visible in the URL. The GET method transmits parameters in the HTTP header: The parameters are visible in the URL and can be logged by servers and clients. Logging often takes place in zones which are not approved for the required protection class. It is also possible that a URI (URL or URN) is made available via the referrer field in the HTTP header to other servers and applications. This information can be used to enumerate link lists as well as for logging.

Implementation example: OWASP REST Security Cheat Sheet (last invocation: 04/10/19): https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html

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

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

ID: 3.54-17/6.2

4. Requirements during the development phase

Req 18	Standard libraries must be used to implement cryptographic operations.

In order to prevent side channel attacks and errors in the software implementation of cryptographic operations, the standard libraries of the operating systems or open source libraries must be used. Open source libraries should be preferred to own implementations. If open source libraries are used, then these should be downloaded only from trusted sources and be checked with regards to their software licenses. In addition, support must be guaranteed. Libraries of any kind with known vulnerabilities must not be used. Security patches must be installed promptly after their availability.

Motivation: Update cycles of operating systems standard libraries and of open source libraries (such as OpenSSL) are often very short and security weaknesses can be remedied quickly. Open source libraries are usually under a continuous community review process. Security patches become available after publication of the weakness. Example: Heartbleed in OpenSSL - https://de.wikipedia.org/wiki/Heartbleed

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: 3.54-18/6.2

Req 19	Individual (i.e., created for an individual app instance, device or user) crypto keys must be used for encrypted storage of confidential information.

Suitable crypto keys shall be used for the type of confidential information involved: user- or device-specific, or both user- and device-specific. One and the same key shall not be used for several app instances on different devices, especially not if it is used to encrypt user-specific confidential information of different users. Likewise, the same key shall not be used when confidential information from different users is stored on one device (in this case the key must be user-specific).

Motivation: If confidential information is encrypted with the same key on different devices, an attacker only needs to gain knowledge of one key in order to decrypt the information on other devices. If information from different users is encrypted with the same key and stored on one device, a user with knowledge of 'his' key could also decrypt confidential information of other users.

ID: 3.54-19/6.2

Req 20	Crypto keys and security tokens must only be saved on the device if the execution environment provides a secure data storage. Otherwise the crypto keys must be calculated in a deterministic way during application execution and the calculation must make use of a factor stored during first start or installation of the app.

Some execution environments offer a secure storage for crypto keys and access data (Keychain of iOS 5.x and higher, security account manager, secure local storage ...). If available, these mechanisms should be used to securely store keys and security tokens. If the execution environment has no access to a secure storage, the crypto keys must not be saved. Instead the keys must be derived in a deterministic way during execution from appropriate data, such as hardware-IDs or similar. Using unique IDs helps to ensure that different app instances (on different devices) derive different keys. As this data is also available for other applications or an attacker, the derivation of crypto keys must be obfuscated within the application to increase the complexity of reverse engineering the application. The derivation of crypto keys must also use another factor ("master key") which is generated during installation (e.g. by use of a random num-

ber generator) and then saved on the device. Furthermore a user set password should be used in addition for the derivation of crypto keys (depends on the use case and the value of the data processed by the app).

Motivation: Secure storages are usually protected by operating system mechanisms from unauthorized access, and therefore offer effective protection against malware, for instance. If the operating system gives users the possibility of protecting the device with an access password, the password will normally also protect the secure storage. If no secure storage will be used: Encrypted information is only as secure (confidential) as the key which is used for the encryption. If the key is known then an encrypted storage does not prevent unauthorized access of the confidential information.

ID: 3.54-20/6.2

Req 21	If the HTTP protocoll is being used for the transmission of encrypted information, then TLS from version 1.2 on must be used.

TLS is a protocol which is used for encrypted data transmission and is generally acknowledged to be secure; it has been standardized by IETF under the name TLS (Transport Layer Security). TLS 1.0 and 1.1 are considered to be secure enough, as they allow usage of outdated algorithms and hash functions, such as SHA-1 and MD5. In addition there are attacks such as BEAST and POODLE, which can effectively be mounted against TLS v1.0 and v1.1. TLS must be used in version 1.2 or higher. Especially when using the HTTP protocol, which is often used by apps; it is common to implement this via TLS (using HTTPS). Also for transfering data from and to mail servers using the protocols SMTP, POP3 or IMAP there are standards that make use of TLS.

Motivation: The use of standardized encryption protocols which are generally acknowledged to be secure offers optimal protection against faulty implementations or security vulnerabilities inherent in protocols during encrypted transmission.

ID: 3.54-21/6.2

Req 22	Before confidential information is transmitted to servers or other instances, these must be authenticated.

When confidential information is transmitted, it is not sufficient to identify the transmission destination via a network identifier (e.g., domain name, IP address, Bluetooth MAC ...); the destination must also be authenticated. X.509 certificates (for SSL/TLS or IPSec), pre-shared secrets or keys built during a previous pairing (with Bluetooth) can all be used to authenticate the destination.

Motivation: If the transmission destination is not reliably identified and authenticated, confidential information may be mistakenly passed to an attacker, e.g., through man-in-the-middle, ARP request poisoning or DNS spoofing attacks. In these cases, encrypted transmission would be useless since a session key would have been negotiated with the attacker.

ID: 3.54-22/6.2

Req 23	If TLS server certificates are being used, then they must be checked by the app for their validity and revocation status. In case of certificate invalidity or timeout of the connection to the revocation or OCSP server, the connection attempt must be aborted and the user informed.

If the destination for encrypted transmission of confidential information with TLS is authenticated with a X.509 certificate, then the validity of the certificate shall be checked. During this process, the whole certificate chain must be validated up to a trusted root Certification Authority (root CA) according to RFC5280. The updates to RFC5280 defined in RFC6818, RFC8389 and RFC8399 must be implemented as well.

A certificate is deemed valid if

- the date of the validity check lies within the certificate's validity period,
- the certificate chain can be validated up to a trusted root Certification Authority (root CA) (i.e., the certificate

must be issued by a Certification Authority, whose certificate was directly or indirectly issued by a trusted root Certification Authority),

- the signature of the Certification Authority (CA) is correct, and
- the certificate has not been revoked by the Certification Authority. In order to validate whether a certificate has been revoked, the signature and the serial number of the certificate must be compared with a current CRL. Alternatively the client application must use OCSP (Online Certificate Status Protocol) according to RFC6960 and OCSP stapling according to RFC6961 for that purpose. All certificates along the entire certificate chain must be valid (the certificate's issuing date is significant for checking the validity of a CA certificate: the CA certificate must have been valid at the point in time when the certificate has been issued). When using OCSP, the behaviour "fail gracefully" must only be implemented after reconciliation with the Project Security Manager and the need for protection of the data processed by the application must be taken into account.

Additional information regarding the certificate validity check: Most execution environments simplify the validity check for certificates by providing relevant libraries or APIs. But even when such libraries or APIs are provided, it shall be ensured that all steps in the validity check are executed. It may be necessary to invoke several API functions in an appropriate sequence. Further information can be found in the security how-tos or best practices for the appropriate execution environment.

Motivation: The check allows to detect compromised or expired server certificates. At worst, compromised certificates of a Certification Authority (CA certificates) are used to issue illegitimate server certificates (see the case of the DigiNotar CA in 2011). A complete validity check of the certificate and of the certificate chain can prevent such attack scenarios. Another attack scenario is possible, if "fail gracefully" is allowed when using OCSP, i.e. if the OCSP responder does not send an answer, it is assumed that the certificate is still valid: In this case an attacker can make sure that no answer reaches the client, and thus a compromised certificate will still be accepted.

ID: 3.54-23/6.2

Req 24	If an SSL/TLS server certificate is being used and it is found to be invalid, a connection must be interrupted.

Motivation: In the case of a man-in-the-middle or impersonation attack (e.g., DNS spoofing), a compromised or expired server certificate could be used to give the impression that the attacker's server is a legitimate server to which confidential information is to be transmitted.

ID: 3.54-24/6.2

5. Installation and update processes

Req 25	Apps must have (semi-)automatic update procedures.

Some execution environments offer automatic update mechanisms which are also used by the corresponding distribution platforms (e.g. app store, market, marketplace). If no automatic update mechanisms are available, an app should check at regular intervals whether a new version is available and, after confirmation by the user, automatically download and install it.

Motivation: Updates are used to remedy and eliminate errors and security vulnerabilities. An automatic update mechanism or appropriate information and prompts for the user guarantee that updates are quickly distributed and old, error-prone non-secure versions swiftly replaced.

ID: 3.54-25/6.2

6. Uninstalling requirements

Req 26	Uninstalling must take place in such a way that confidential user data and corresponding app-specific information is deleted from the execution environment, from the device, from security modules, and from other storage media.

If a user decides to uninstall an app, confidential user data and the corresponding app-specific data such as crypto keys shall be irreversibly deleted. In some cases, the operating system or the execution environment assumes responsibility for deleting all app data during deinstallation and the app itself has no influence on the uninstalling process; in such a case, this requirement is not applicable. In other cases, users want their data to remain stored even after uninstalling, e.g., when saving media files to the SD card. If the user is aware of this circumstance, this data can remain stored. Crypto keys and security tokens shall, however, be deleted and overwritten, if the app is involved in the uninstallation process.

Motivation: The irreversible deletion of confidential user data, crypto keys and security tokens during app uninstallation prevents leakage, at a later time, of confidential information that is left on the device.

ID: 3.54-26/6.2

7. General requirements

7.1. Protecting availability and integrity

Req 27	The system must be implemented robustly against unexpected inputs.

Data transferred to the system must first be validated before further processing to ensure that the data corresponds to the expected data type and format. This is intended to eliminate the risk of manipulation of system processes and states by appropriately constructed data content. Validation must be carried out for any data that is transferred to the system. Examples include user input, values in data fields, and log contents.

The following typical implementation mistakes must be avoided:

- lack of validation of the length of passed data
- Incorrect assumptions about the format of data
- lack of validation of received data for conformity with the specification
- Inadequate handling of protocol deviations in received data
- Insufficient limitation of recursion when parsing complex data formats
- Insufficient implementation of whitelisting or escaping to protect against inputs outside the valid value range

Motivation: An attacker can use specifically engineered data content to try to put a system that does not sufficiently validate received data before internal processing into an unstable state or to trigger unauthorized actions within the system. The damage potential of such attacks depends on the individual system, but has a theoretical range from uncontrolled system crashes to a controlled execution of specially injected code and the resulting complete compromise of a system.

For this requirement the following threats are relevant:

- Unauthorized access to the system
- Unauthorized access or tapping of data
- Unauthorized modification of data
- Unauthorized use of services or resources
- Disruption of availability
- Unnoticeable feasible attacks

ID: 3.01-11/8.0

7.2. Protecting sessions

Req 28	Sessions must be protected against unauthorized takeover ("session hijacking").

Interfaces that provide session functionality to the system must implement technical measures to prevent a legitimate user's session from being taken over and continued by an unauthorized third party.

Such protection can be achieved, for example, by implementing a combination of the following options that makes sense for the specific system:

- At the transport layer: Use of the TCP protocol (with its sequence numbers) and corresponding filter lists
- At the session layer: Use of the TLS Protocol
- At the application layer: Negotiation of a random secret session key between sender and receiver to authorize all session traffic (e.g. session ID, session cookie, session token)
- Use of cryptographic methods to protect session keys from eavesdropping or modification attacks

Motivation: Unprotected sessions can potentially be hijacked and continued by an attacker in order to exercise unauthorized access to the system in the context of the affected user.

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: 3.01-16/8.0

Req 29	Sessions must be automatically terminated after a period of inactivity adapted to the intended use.

It is necessary that sessions on a system are automatically terminated after a specified period of inactivity.

For this reason, a time-out for sessions must be set. The time period to be selected here depends on the use of the system and, if applicable, the physical environment. For example, the time-out for an application in an unsecured environment must be shorter (a few minutes) than the time-out for an application used by operations personnel for system monitoring tasks in an access-protected area (60 minutes or more).

Motivation: For an open but unused session, there is a risk that an illegitimate user may take over and continue it unnoticed in order to exercise unauthorized access to the system and the data contained therein on behalf of the affected user.

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: 3.01-18/8.0

7.3. Authentication parameter password

Req 30	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 31	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 32	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 33	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 34	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 35	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 dis-

play 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

7.4. Logging

Req 36	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