Security requirement

OWASP Mobile Application Security Verification Standard (Android / iOS)

Deutsche Telekom Group

Version	1.1
Date	Jul 1, 2023
Status	Released

Publication Details

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

Reuterstrasse 65, 53315 Bonn
Germany

File name	Document number	Document type
	4.1.1	Security requirement

Version	State	Status
1.1	Jul 1, 2023	Released

Contact	Validity	Released by
Telekom Security psa.telekom.de	Jul 1, 2023 - Jun 30, 2028	Stefan Pütz, Leiter SEC-T-TST

Summary
Android / iOS requirements according to the OWASP Mobile Application Security Verification Standard and OWASP Mobile Application Security Testing Guide. This document provides requirements for software architects and developers who want to develop mobile applications securely. It serves as an industry standard for verifying the security of mobile applications. It also clarifies the role of software protection mechanisms in mobile security and provides requirements for verifying their effectiveness. The document also provides specific recommendations for the level of security recommended for different use cases.
MASVS Release 1.5.0, MASTG Release 1.5.0

Table of Contents

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. General

2.1. Enterprise Appstore

Beyond the requirements described in this document, it must be taken into account that the Enterprise Appstore must be used for the distribution of apps that process data of the "Internal" protection class and higher and that are used Telekom internally. In addition, the SDK for containers must be used for in-house developments of such apps.

2.2. Standards

This requirements document is identical in form and content to the requirements from the OWASP Mobile AppSec Verification Standard (MASVS) and the OWASP Mobile Security Testing Guide (MSTG). The document is thus based on an international standard, on the one hand to ensure comparability of results outside DTAG and, on the other hand, enable automatic, static tests with freely available tools (such as MobSF - the Mobile Security Framework).

3. Android / iOS Development

3.1. Architecture, Design and Threat Modeling Requirements

Req 1	All app components are identified and known to be needed.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Android and iOS offer access to many standard libraries and APIs for the development of applications, which are directly integrated in the SDKs of the vendor. During the development of an application it is important to ensure that only the components that are needed for the execution of the application are linked in order to keep the attack surface as small as possible. For example, a flashlight app typically does not need access to the camera functions of the device.

Implementation example: **Android:**
https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05h-Testing-Platform-Interaction.md#testing-for-insecure-configuration-of-instant-apps-mstg-arch-1-mstg-arch-7

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-1/1.1

Req 2	Security controls are never enforced only on the client side, but on the respective remote endpoints.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Unilateral mapping or control of security functions can be exploited by attackers to gain elevated access on the backend through targeted manipulation of API calls.

Implementation example: **Android:**
https://mobile-security.gitbook.io/mobile-security-testing-guide/general-mobile-app-testing-guide/0x04h-testing-code-quality#injection-flaws-mstg-arch-2-and-mstg-platform-2

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-2/1.1

Req 3	A high-level architecture for the mobile app and all connected remote services has been defined and security has been addressed in that architecture.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The API endpoints associated with the app are documented in the architecture and described in terms of security. This can be achieved with tools such as Swagger to ensure a quick and detailed overview and thus prevent errors such as unauthenticated access and counteract them accordingly as early as the design phase.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-3/1.1

Req 4	Data considered sensitive in the context of the mobile app is clearly identified.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The data to be processed and transmitted by the app is clearly classified as "public", "internal", "confidential" or "TBS" (Top Business Secret) according to the protection classes. This is necessary to ensure that, based on the requirements, all data is protected accordingly at all times.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-4/1.1

Req 5	All app components are defined in terms of the business functions and/or security functions they provide.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The components used by the app are technically required for the execution of the app and the security functions are clearly described and defined. At any time it is ensured that the security dependencies are already taken into account in the development process.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-5/1.1

Req 6	A threat model for the mobile app and the associated remote services has been produced that identifies potential threats and countermeasures.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: For the API endpoints in use, checks ensure already during the development phase that the security functions have been implemented as planned. In the event of any unintended exposure, countermeasures are available to respond to threats accordingly.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-6/1.1

Req 7	All security controls have a centralized implementation.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: There are app types, such as Google Play Instant Apps, which can be executed without installation. These apps do without a large part of the available APIs and permissions (due to size restrictions, among other things), which is purchased with correspondingly fewer options for security-related protection at runtime.

Implementation example: **Android:**
https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05h-Testing-Platform-Interaction.md#testing-for-insecure-configuration-of-instant-apps-mstg-arch-1-mstg-arch-7

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-7/1.1

Req 8	There is an explicit policy for how cryptographic keys (if any) are managed, and the lifecycle of cryptographic keys is enforced. Ideally, follow a key management standard such as NIST SP 800-57.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: In all cases concerning the key material, it is necessary that the key management is carried out exclusively on the basis of established principles. There may be different requirements that need to be taken into account depending on the country and company.
Ideally, key material is generated directly on the end device and private key material is not stored outside the end device. However, there may be cases where it is not possible to generate the key material on the end device or there are legal obligations to store key material centrally. In both cases, it must be ensured that the key material is transmitted and stored with adequate protection to prevent misuse by third parties.

Implementation example: **Android:**
https://mobile-security.gitbook.io/mobile-security-testing-guide/general-mobile-app-testing-guide/0x04g-testing-cryptography#cryptographic-policy-mstg-arch-8

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-8/1.1

Req 9	A mechanism for enforcing updates of the mobile app exists.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Apple Appstore and Google Playstore offer automatic update mechanisms, but these can be delayed or deactivated by the user. To ensure that critical security updates in particular are rolled out as quickly as possible, apps must have their own mechanisms.

Implementation example: **Android:**
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05d-Testing-Data-Storage.md#testing-logs-for-sensitive-data-mstg-storage-3
https://developer.android.com/guide/playcore/in-app-updates

iOS:
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06d-Testing-Data-Storage.md#checking-logs-for-sensitive-data-mstg-storage-3

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

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-9/1.1

Req 10	Security is addressed within all parts of the software development lifecycle.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Security requirements are considered at all stages of the development to avoid costly delays or critical security issues caused by late introduction of the security requirements.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-10/1.1

Req 11	A responsible disclosure policy is in place and effectively applied.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Applications published in the app stores should ensure that there is a direct information channel for responsible disclosures. This means that security vulnerabilities discovered by users, companies or security researchers can be reported directly and solved in exchange so that (critical) security vulnerabilities can be solved quickly and easily.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-11/1.1

Req 12	The app should comply with privacy laws and regulations.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data

Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The development of the application takes into account the requirements of data protection at every stage of development and applies security features such as encryption to technically meet the requirements.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-12/1.1

3.2. Data Storage Requirements

Req 13	System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Data in the "internal" and "confidential" protection classes uses the data encryption and key management functions provided by Android and iOS. This ensures that no proprietary developments are used that may be less secure and offer corresponding attack surfaces.

Implementation example: **Android:**
https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md#testing-local-storage-for-sensitive-data-mstg-storage-1-and-mstg-storage-2

iOS:
https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06d-Testing-Data-Storage.md#testing-local-data-storage-mstg-storage-1-and-mstg-storage-2

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-13/1.1

Req 14	No sensitive data should be stored outside of the app container or system credential storage facilities.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Storage locations outside the app container can rarely be influenced by the app and are often also ac-

cessed by other apps to exchange data such as pictures, e.g. the folders for photos can be read and written by all apps. On Android devices, it can happen that the storage location is a removable medium and is not affected by the operating system's encryption options.

Implementation example: **Android:**
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05d-Testing-Data-Storage.md#testing-local-storage-for-sensitive-data-mstg-storage-1-and-mstg-storage-2
https://developer.android.com/guide/playcore/in-app-updates

iOS:
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06d-Testing-Data-Storage.md#testing-local-data-storage-mstg-storage-1-and-mstg-storage-2

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-14/1.1

Req 15	No sensitive data is written to application logs.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Data of the "confidential" protection class or personal data, such as passwords, private keys, etc., must not be included in the logs. Logs can typically be shared with third parties, e.g. for necessary analysis purposes, and it must be ruled out that employees or third parties receive information about the user's account.

Implementation example: **Android:**
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05d-Testing-Data-Storage.md#testing-logs-for-sensitive-data-mstg-storage-3
https://developer.android.com/guide/playcore/in-app-updates

iOS:
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06d-Testing-Data-Storage.md#checking-logs-for-sensitive-data-mstg-storage-3

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-15/1.1

Req 16	No sensitive data is shared with third parties unless it is a necessary part of the architecture.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The app does not share any data with third parties unless the app architecture provides for this and all services involved have been described, released and / or corresponding protective measures for the storage, transmission and, if necessary, further processing of the data have been ensured as part of the PSA process.
Functions built into apps to share data with third parties typically requiring the consent of the user and active usage of another service, are intended to ensure that the interfaces are operated securely and data collected by services such as Crashlytics, Embrace or Sentry is neither shared with third parties without consent nor is the data transmitted without protective measures.

Implementation example: **Android:**
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05d-Testing-Data-Storage.md#determining-whether-sensitive-data-is-shared-with-third-parties-mstg-storage-4
https://developer.android.com/guide/playcore/in-app-updates

iOS:
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06d-Testing-Data-Storage.md#determining-whether-sensitive-data-is-shared-with-third-parties-mstg-storage-4

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-16/1.1

Req 17	The keyboard cache is disabled on text inputs that process sensitive data.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Fields processing sensitive information must disable keyboard caching. This measure prevents sensitive data from being stored and processed in a potentially insecure manner. Sensitive input fields are, for example, personal information (e.g. birthday), bank data (account no., credit card data) or passwords.

Implementation example: **Android:**

```
<EditText
android:id="@+id/KeyBoardCache"
android:inputType="textNoSuggestions" />
```

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05d-Testing-Data-Storage.md#determining-whether-the-keyboard-cache-is-disabled-for-text-input-fields-mstg-storage-5

iOS:

```
textObject.autocorrectionType = UITextAutocorrectionTypeNo;
textObject.secureTextEntry = YES;
```

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06d-Testing-Data-Storage.md#finding-sensitive-data-in-the-keyboard-cache-mstg-storage-5

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-17/1.1

Req 18	No sensitive data is exposed via IPC mechanisms.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Android offers a very open concept for inter-process communication (IPC) with providers, intents and permissions, which can be restricted to the exchange of data by the same developer via "Signature", while iOS offers only limited options via NSMachPort. If no interaction by the user is intended, i.e. no share sheet or similar is to be used, then the IPC can be used.
Since the exchange of data typically takes place without interaction with the user, IPC should only be used with apps from the same developer and should not expose any sensitive data of the app or the user to ensure that third-party apps definitely cannot access the data.

Implementation example: **Android:**
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05d-Testing-Data-Storage.md#determining-whether-sensitive-stored-data-has-been-exposed-via-ipc-mechanisms-mstg-storage-6

iOS:
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06d-Testing-Data-Storage.md#determining-whether-sensitive-data-is-exposed-via-ipc-mechanisms-mstg-storage-6

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-18/1.1

Req 19	No sensitive data, such as passwords or pins, is exposed through the user interface.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Sensitive data such as passwords is entered in a obscured manner to prevent third parties from gaining access to the data through shoulder surfing. In the best case, the use of passwords is avoided and alternative login methods such as FIDO2, 3rd party identity providers (Apple, Google, Verimi, etc.) or similarly secure passwordless methods are given priority.

Implementation example: Android:

```
android:inputType="textPassword"
```

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05d-Testing-Data-Storage.md#checking-for-sensitive-data-disclosure-through-the-user-interface-mstg-storage-7

iOS:

```
sensitiveTextField.isSecureTextEntry = true
```

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06d-Testing-Data-Storage.md#checking-for-sensitive-data-disclosed-through-the-user-interface-mstg-storage-7

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-19/1.1

Req 20	No sensitive data is included in backups generated by the mobile operating system.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Android and iOS offer different options for storing the data of a device in the cloud, on removable media or on another computer. In all cases, however, it can be ensured that the data cannot be stored outside the device in the first place.
The requirement is not intended to prevent an app from reading and writing data outside the device, but to ensure that private keys, confidential configurations, etc. are not unintentionally included in the backup without additional protection. Sensitive and confidential items should only be stored using the operating systems' secure storage options (Android key attestation, Apple keychain) while sensitive and confidential files should be additionally *encrypted using the operating system's options.*

Implementation example: **Android:**

```
android:allowBackup="false"
```

https://developer.android.com/training/articles/security-key-attestation
https://source.android.com/security/encryption/file-based
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05d-Testing-Data-Storage.md#testing-backups-for-sensitive-data-mstg-storage-8

iOS:

```
let kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly: CFString
```

https://developer.apple.com/documentation/security/ksecattraccessiblewhenpasscodesetthisdeviceonly
https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/encrypting_your_app_s_files
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06d-Testing-Data-Storage.md#testing-backups-for-sensitive-data-mstg-storage-8

For this requirement the following threats are relevant:

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-20/1.1

Req 21	The app removes sensitive data from views when moved to the background.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: If the app is sent to the background, a preview image (snapshot) is also generated, among other things, to make the user's selection for foreground mode more intuitive and simpler. The operating systems save the preview image in a folder on the end device, which can also contain sensitive data that can be included in backups without additional protection.

Implementation example: **Android:**

https://developer.android.com/reference/android/view/Display#FLAG_SECURE
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05d-Testing-Data-Storage.md#finding-sensitive-information-in-auto-generated-screenshots-mstg-storage-9

iOS:

```
private var backgroundImage: UIImageView?

func applicationDidEnterBackground(_ application: UIApplication) {
     let myBanner = UIImageView(image: #imageLiteral(resourceName:
"overlayImage"))
    myBanner.frame = UIScreen.main.bounds
    backgroundImage = myBanner
    window?.addSubview(myBanner)
}

func applicationWillEnterForeground(_ application: UIApplication) {
    backgroundImage?.removeFromSuperview()
}
```

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06d-Testing-Data-Storage.md#testing-auto-generated-screenshots-for-sensitive-information-mstg-storage-9

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-21/1.1

Req 22	The app does not hold sensitive data in memory longer than necessary, and memory is cleared explicitly after use.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Despite all efforts by Apple, Google and app developers, programming errors cannot be avoided even by regular updates. Newer and newer methods are used to attack data from the operating systems or other installed apps. The smaller the attack surface is kept, i.e. the less data is kept on the end device, the less likely it is that attackers will gain access to larger amounts of data.
Ideally, data is only cached in RAM and the corresponding memory areas are deleted again after processing. Depending on the application, it may also be necessary to work with gradations, i.e. also caching or permanent storage in the file system (e.g. after inputs and simultaneous loss of the network connection, network change between WiFi <=> mobile radio, etc.), whereby methods of secure and encrypted storage should be used.

Implementation example: Android:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05d-Testing-Data-Storage.md#testing-memory-for-sensitive-data-mstg-storage-10

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06d-Testing-Data-Storage.md#testing-memory-for-sensitive-data-mstg-storage-10

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-22/1.1

Req 23	The app enforces a minimum device-access-security policy, such as requiring the user to set a device passcode.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Outside the company, it cannot be guaranteed that end devices have PIN or password protection. Without appropriate protection, the data stored in an app cannot be adequately protected by OS security features, since either the corresponding APIs and/or functions for encrypted storage, for example, are not available in a secure manner. Although the app could be protected by an individual password, implementing one's own protection and encryption measures is much too complex and definitely not recommended from a security point of view.
For this reason, Apple and Google offer options to check whether a corresponding device protection is set. If this is not the case and the app depends on it for various reasons, a corresponding response can be made and the user can be asked to activate the corresponding protection mechanisms.

Implementation example: Android:

https://developer.android.com/reference/android/provider/Settings.Secure.html
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05f-Testing-Local-Authentication.md#testing-confirm-credentials-mstg-auth-1-and-mstg-storage-11

iOS:

https://developer.apple.com/documentation/localauthentication/lapolicy/deviceownerauthentication
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06f-Testing-Local-Authentication.md#testing-local-authentication-mstg-auth-8-and-mstg-storage-11

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-23/1.1

Req 24	The app educates the user about the types of personally identifiable information processed, as well as security best practices the user should follow in using the app.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The user can decide whether and how to use the app within the scope of the declaration.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04i-Testing-User-Privacy-Protection.md#testing-user-education-mstg-storage-12

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-24/1.1

Req 25	No sensitive data should be stored locally on the mobile device. Instead, data should be retrieved from a remote endpoint when needed and only be kept in memory.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Despite all efforts by Apple, Google and app developers, programming errors cannot be avoided even by regular updates. Newer and newer methods are used to attack data from the operating systems or other installed apps. The smaller the attack surface is kept, i.e. the less data is kept on the end device, the less likely it is that attack-

ers will gain access to larger amounts of data.
Ideally, data is only cached in RAM and the corresponding memory areas are deleted again after processing. Depending on the application, it may also be necessary to work with gradations, i.e. also caching or permanent storage in the file system (e.g. after inputs and simultaneous loss of the network connection, network change between WiFi <=> mobile radio, etc.), whereby methods of secure and encrypted storage should be used.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-25/1.1

Req 26	If sensitive data is still required to be stored locally, it should be encrypted using a key derived from hardware backed storage which requires authentication.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Sensitive and confidential elements (typically small amounts of data such as authentication data, keys, tokens, etc.) should only be stored using the operating systems' secure storage options (Android Key Attestation, Apple Keychain), while sensitive and confidential files (typically larger amounts of data such as databases) should also be encrypted using the operating system's options. To ensure that access is only granted by an authorized user, authentication is required upon access.

Implementation example: **Android:**

https://developer.android.com/training/articles/security-key-attestation
https://source.android.com/security/encryption/file-based

iOS:

```
let kSecAttrAccessibleWhenUnlockedThisDeviceOnly: CFString
```

https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlockedthisdeviceonly
https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/encrypting_your_app_s_files

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-26/1.1

Req 27	The app’s local storage should be wiped after an excessive number of failed authentication attempts.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: To ensure that an attacker using bruteforce methods does not gain access to the app's sensitive or confidential data, the time period between two login attempts should be increased after a maximum of three attempts (Tarpit). After the seventh failed login attempt, the user should be informed that the app will be reset to its factory default state after another three attempts, and all local data will be deleted in the process.
The option should be addressed directly in the setup dialog (e.g., when setting the app PIN), and the user can be offered an opt-out if simultaneously informed about the risks.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-27/1.1

3.3. Cryptography Requirements

Req 28	The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The app always establishes a hardware and user reference for symmetric encryption to ensure that the encrypted data generally cannot be decrypted without the hardware and user-related key. Key material built into the app (hardcoded keys, passwords) is not permitted under any circumstances for apps being developed, as attackers could draw conclusions about its use once published in the app stores.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04g-Testing-Cryptography.md#common-configuration-issues-mstg-crypto-1-mstg-crypto-2-and-mstg-crypto-3

Android:

https://developer.android.com/training/articles/security-key-attestation
https://source.android.com/security/encryption/file-based
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05e-Testing-Cryptography.md#testing-symmetric-cryptography-mstg-crypto-1

iOS:

https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlockedthisdeviceonly
https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/encrypting_your_app_s_files
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06e-Testing-Cryptography.md#testing-key-management-mstg-crypto-1-and-mstg-crypto-5

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

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-28/1.1

Req 29	The app uses proven implementations of cryptographic primitives.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The development, implementation and maintenance of secure crypto functions and libraries is not only complex and time-consuming, but should above all be available to a broad specialist audience for evaluation in order to identify vulnerabilities in good time. For this reason, only the functions and libraries provided by Apple and Google should be used for app development. These have been further developed for years and are repeatedly checked and tested for vulnerabilities by security researchers.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04g-Testing-Cryptography.md#common-configuration-issues-mstg-crypto-1-mstg-crypto-2-and-mstg-crypto-3

Android:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05e-Testing-Cryptography.md#testing-the-configuration-of-cryptographic-standard-algorithms-mstg-crypto-2-mstg-crypto-3-and-mstg-crypto-4

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06e-Testing-Cryptography.md#verifying-the-configuration-of-cryptographic-standard-algorithms-mstg-crypto-2-and-mstg-crypto-3

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-29/1.1

Req 30	The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Depending on the use case, the appropriate cryptographic methods, functions, libraries, ciphers and algorithms are used, applying up-to-date security best practices. For example, after salting a password, SHA-2 384 is

used to create the hash while MD5 and SHA-1 are considered insecure and must not be used.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04g-Testing-Cryptography.md#common-configuration-issues-mstg-crypto-1-mstg-crypto-2-and-mstg-crypto-3

Android:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05e-Testing-Cryptography.md#testing-the-configuration-of-cryptographic-standard-algorithms-mstg-crypto-2-mstg-crypto-3-and-mstg-crypto-4

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06e-Testing-Cryptography.md#verifying-the-configuration-of-cryptographic-standard-algorithms-mstg-crypto-2-and-mstg-crypto-3

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-30/1.1

Req 31	The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Deprecated methods, functions, libraries, ciphers, and algorithms or those considered insecure must not be used because attackers exploit the already published vulnerabilities to gain access to the supposedly securely encrypted data or accounts. For example, after salting a password, SHA-2 384 is used to create the hash while MD5 and SHA-1 are considered insecure and must not be used.

Implementation example: **Allgemein:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04g-Testing-Cryptography.md#identifying-insecure-andor-deprecated-cryptographic-algorithms-mstg-crypto-4

Android:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05e-Testing-Cryptography.md#testing-the-configuration-of-cryptographic-standard-algorithms-mstg-crypto-2-mstg-crypto-3-and-mstg-crypto-4

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-31/1.1

Req 32	The app doesn't re-use the same cryptographic key for multiple purposes.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The reuse of cryptographic keys is to be evaluated in the same way as the reuse of passwords. After successful compromise, all data secured with the key is exposed. If the key material is evenly spread, the risk of compromising all data is reduced accordingly.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05e-Testing-Cryptography.md#testing-the-purposes-of-keys-mstg-crypto-5

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06e-Testing-Cryptography.md#testing-key-management-mstg-crypto-1-and-mstg-crypto-5

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-32/1.1

Req 33	All random values are generated using a sufficiently secure random number generator.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The functions provided by Android and iOS must be used to generate cryptographically secure pseudo-random numbers. Own implementations or the generation of simple random numbers is cryptographically insecure and therefore not allowed.

Implementation example: **Android:**

```
SecureRandom random = new SecureRandom();
byte bytes = new byte[20];
random.nextBytes(bytes);
```

https://developer.android.com/reference/java/security/SecureRandom.html
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05e-Testing-Cryptography.md#testing-random-number-generation-mstg-crypto-6

iOS:

```
func SecRandomCopyBytes(
```

```
_ rnd: SecRandomRef?,
_ count: Int,
_ bytes: UnsafeMutableRawPointer
) -> Int32
```

https://developer.apple.com/documentation/security/1399291-secrandomcopybytes
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06e-Testing-Cryptography.md#testing-random-number-generation-mstg-crypto-6

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-33/1.1

3.4. Authentication and Session Management Requirements

Req 34	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: To prevent data from being read without authorization, each remote endpoint is authenticated and authorized. Anonymous use is permitted only if the service is designed for this purpose and performs appropriate authentication and authorization when accessing data of protection class "internal" or higher.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04e-Testing-Authentication-and-Session-Management.md#verifying-that-appropriate-authentication-is-in-place-mstg-arch-2-and-mstg-auth-1

Android:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05f-Testing-Local-Authentication.md#testing-confirm-credentials-mstg-auth-1-and-mstg-storage-11

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-34/1.1

Req 35	If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user's credentials.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: To prevent user data from being compromised by third parties, methods are used to reduce the risk of compromise accordingly. Among other things, randomly generated session IDs are used.

Implementation example: **Common:**

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#session-attacks-detection
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04e-Testing-Authentication-and-Session-Management.md#testing-stateful-session-management-mstg-auth-2

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-35/1.1

Req 36	If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: To ensure that created tokens are not manipulated by third parties, the tokens are cryptographically securely signed when issued and always verified by the server.

Implementation example: **Common:**

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#session-attacks-detection
https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04e-Testing-Authentication-and-Session-Management.md#testing-stateless-token-based-authentication-mstg-auth-3

Android:

https://developers.google.com/identity/protocols/oauth2/native-app?hl=en

iOS:

https://developer.apple.com/documentation/sign_in_with_apple/sign_in_with_apple_rest_api/authenticating_users_with_sign_in_with_apple
https://developer.apple.com/documentation/sign_in_with_apple/sign_in_with_apple_js/incorporating_sign_in_with_apple_into_other_platforms

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

- Unauthorized use of services or resources

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-36/1.1

Req 37	The remote endpoint terminates the existing session when the user logs out.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Many mobile apps don't automatically log users out. There can be various reasons, such as: because it is inconvenient for customers, or because of decisions made when implementing stateless authentication. The application should still have a logout function, and it should be implemented according to best practices, destroying all locally stored tokens or session identifiers. If session information is stored on the server, it should also be destroyed by sending a logout request to that server. In case of a high-risk application, tokens should be invalidated. Not removing tokens or session identifiers can result in unauthorized access to the application in case the tokens are leaked. It should be noted that other sensitive types of information should be removed as well, as any information that is not properly cleared may be leaked later, for example during a device backup.

Implementation example: **Common:**

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html
https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04e-Testing-Authentication-and-Session-Management.md#testing-user-logout-mstg-auth-4

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-37/1.1

Req 38	A password policy exists and is enforced at the remote endpoint.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Password strength is a key concern when passwords are used for authentication. The password policy defines requirements to which end users should adhere. A password policy typically specifies password length, password complexity, and password topologies. A "strong" password policy makes manual or automated password cracking difficult or impossible.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04e-Testing-Authentication-and-Session-Management.md#testing-best-practices-for-passwords-mstg-auth-5-and-mstg-auth-6

For this requirement the following threats are relevant:

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-38/1.1

Req 39	The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The remote backend implements protective measures against password brute-forcing. This includes ensuring that authorized users are not locked out during a denial of service attack. For this purpose, tarpitting (delayed login attempts), temporary IP address blocks or similar techniques can be used.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04e-Testing-Authentication-and-Session-Management.md#testing-best-practices-for-passwords-mstg-auth-5-and-mstg-auth-6

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-39/1.1

Req 40	Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Minimizing the lifetime of session identifiers and tokens decreases the likelihood of successful account hijacking.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04e-Testing-Authentication-and-Session-Management.md#testing-session-timeout-mstg-auth-7

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

- Unauthorized use of services or resources

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-40/1.1

Req 41	Biometric authentication, if any, is not event-bound (i.e. using an API that simply returns "true" or "false"). Instead, it is based on unlocking the keychain/keystore.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The APIs for evaluating whether a biometric authentication by the user was successful typically only return "true" or "false". If the app only checks the Boolean value, attackers could exploit this situation to gain unauthorized access to the application.
The correct approach is to store keys in the keystore (Android) or the keychain (iOS) during setup and to read and use them after successful authentication to log in to the app or to unlock the content or the encryption keys. This is the only way to ensure that a successful (biometric) unlocking process has actually taken place.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05f-Testing-Local-Authentication.md#testing-biometric-authentication-mstg-auth-8

iOS:

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06f-Testing-Local-Authentication.md#testing-local-authentication-mstg-auth-8-and-mstg-storage-11

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-41/1.1

Req 42	A second factor of authentication exists at the remote endpoint and the 2FA requirement is consistently enforced.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The remote system requires multi-factor authentication to prevent that even if passwords have been compromised, access to the system with stolen user credentials is not readily possible.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04e-Testing-Authentication-and-Session-Management.md#testing-two-factor-authentication-and-step-up-authentication-mstg-auth-9-and-mstg-auth-10

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-42/1.1

Req 43	Sensitive transactions require step-up authentication.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Access to sensitive, personal, data of the protection class from "confidential" and higher or, depending on the app, critical elements (e.g. deletion of the account, changing billing data, etc.) is secured by interactive re-authentication of the user to prevent third parties from gaining access to corresponding content in the event of compromise.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04e-Testing-Authentication-and-Session-Management.md#testing-two-factor-authentication-and-step-up-authentication-mstg-auth-9-and-mstg-auth-10

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-43/1.1

Req 44	The app informs the user of all sensitive activities with their account. Users are able to view a list of devices, view contextual information (IP address, location, etc.), and to block specific devices.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Multiple logins, especially from different end devices, are expected by users today. However, it is often the case that users are rarely aware of how many (long-lived) sessions accumulate over time from the most diverse end devices and browsers. The app should actively support the user in keeping a secure overview of all active logins and especially in informing about new logins, e.g. via push notifications and emails. However, in order not to cause security fatigue among users due to too many notifications, the application can also actively learn and know over time to distinguish between normal logins of the user and those of a possible attacker. E.g. the user typically does not log in between 23:00 and 07:00 hours in the morning and otherwise only within the EU; the app warns the user about login attempts from outside the EU or his typical time of use. Within the app, the user can view all active logins and their origin (IP, end device, etc.) and actively terminate individual or all sessions (except the currently active one). These measures ensure that the user is always aware of open sessions and can immediately eliminate unknown ones accordingly

and initiate countermeasures, such as changing the password, etc.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04e-Testing-Authentication-and-Session-Management.md#testing-login-activity-and-device-blocking-mstg-auth-11

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-44/1.1

Req 45	Authorization models should be defined and enforced at the remote endpoint.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Authentication and authorization take place on the remote system, since this is ultimately where the user's accounts, information, and data that match the identity converge and are kept or established.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-45/1.1

3.5. Network Communication Requirements

Req 46	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: One of the core mobile app functions is sending/receiving data over untrusted networks like the Internet. If the data is not properly protected in transit, an attacker with access to any part of the network infrastructure (e.g., a Wi-Fi access point) may intercept, read, or modify it.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04f-Testing-Network-Communication.md#verifying-data-encryption-on-the-network-mstg-network-1

Android:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05g-Testing-Network-Communication.md#testing-data-encryption-on-the-network-mstg-network-1

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06g-Testing-Network-Communication.md#testing-data-encryption-on-the-network-mstg-network-1

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-46/1.1

Req 47	The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Connections are not automatically secure or trustworthy when using TLS or HTTPS. This is only achieved in combination with correspondingly secure cipher suites. The cipher suites must support PFS (Perfect Forward Secrecy) with ECDHE (Elliptic Curve Diffie-Hellmann Ephemeral) key exchange together with AES256.

For developments, TLS 1.3 (cipher suites are specified) and TLS 1.2 with the following cipher suites are permitted:
- *TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384*
- *TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384*

TLS 1.3 is already enabled by default since Android 10 and iOS 12.2, so incompatibilities are no longer expected in the field. TLS 1.2 should therefore only be used if incompatibilities occur during development or field tests and cannot be ruled out for production.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04f-Testing-Network-Communication.md#verifying-the-tls-settings-mstg-network-2

Android:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05g-Testing-Network-Communication.md#testing-the-tls-settings-mstg-network-2

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06g-Testing-Network-Communication.md#testing-the-tls-settings-mstg-network-2

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

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-47/1.1

Req 48	The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Using TLS to transport sensitive information over the network is essential for security. However, encrypting communication between a mobile application and its backend API is not trivial. Less secure solutions (e.g., those that accept any certificate) facilitate the development process, and sometimes these weak solutions make it into the production version, potentially exposing users to man-in-the-middle attacks.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05g-Testing-Network-Communication.md#testing-endpoint-identify-verification-mstg-network-3

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06g-Testing-Network-Communication.md#testing-endpoint-identity-verification-mstg-network-3

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-48/1.1

Req 49	The app either uses its own certificate store, or pins the endpoint certificate or public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, even if signed by a trusted CA.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Certificate pinning is the process of associating the backend server with a particular X.509 certificate or public key instead of accepting any certificate signed by a trusted certificate authority. After storing ("pinning") the server certificate or public key, the mobile app will subsequently connect to the known server only. Withdrawing trust from external certificate authorities reduces the attack surface (after all, there are many cases of certificate authorities that have been compromised or tricked into issuing certificates to impostors).

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05g-Testing-Network-Communication.md#testing-custom-certificate-stores-and-certificate-pinning-mstg-network-4
https://developer.android.com/training/articles/security-config#CertificatePinning

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06g-Testing-Network-Communication.md#testing-custom-certificate-stores-and-certificate-pinning-mstg-network-4
https://developer.apple.com/news/?id=g9ejcf8y

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-49/1.1

Req 50	The app doesn't rely on a single insecure communication channel (email or SMS) for critical operations, such as enrollments and account recovery.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Email and SMS are generally not secure communication channels and should therefore not be used by default for apps that process data in the "confidential" protection class, sensitive data or strictly personal data. Such apps include banking apps, collaboration tools (with access to a lot of confidential data), health apps, and so on. Appropriate alternatives should be prioritized, including hardware-based tokens, push notifications (approval / rejection of the transaction), Google Sign-In, Sign In With Apple, QR code, TOTP (time-based tokens such as Google Authenticator or Microsoft Authenticator), activation letter, etc.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04f-Testing-Network-Communication.md#making-sure-that-critical-operations-use-secure-communication-channels-mstg-network-5

Android:

https://developers.google.com/identity/sign-in/android/start-integrating

iOS:

https://developer.apple.com/documentation/authenticationservices/implementing_user_authentication_with_sign_in_with_apple

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-50/1.1

Req 51	The app only depends on up-to-date connectivity and security libraries.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: If outdated libraries are used for app development, there is a risk that errors or known security vulnerabilities will be shipped with the application, making the application vulnerable to attacks. Developers must therefore ensure that the libraries used are up to date and switch to updated versions in good time. In particular, the integration of third-party libraries should be avoided when it comes to core components such as secure network connections.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05g-Testing-Network-Communication.md#testing-the-security-provider-mstg-network-6
https://developer.android.com/training/basics/network-ops/connecting
https://developer.android.com/training/articles/security-ssl

iOS:

https://developer.apple.com/documentation/network

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-51/1.1

3.6. Platform Interaction Requirements

Req 52	The app only requests the minimum set of permissions necessary.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Apps can be granted more permissions than they need for execution. A possible vulnerability could then be exploited to manipulate the app or the end device. A QR-Code reader, for example, needs access to the camera while the app is being used, but never to the microphone. In addition, both Android and iOS have guidelines on how to request permissions from the user.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05h-Testing-Platform-Interaction.md#testing-app-permissions-mstg-platform-1
https://developer.android.com/guide/topics/permissions/overview.html
https://developer.android.com/training/permissions/requesting

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06h-Testing-Platform-Interaction.md#testing-app-permissions-mstg-platform-1
https://developer.apple.com/documentation/uikit#//apple_ref/doc/uid/TP40007072-CH3-SW7
https://developer.apple.com/design/human-interface-guidelines/patterns/accessing-private-data/

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-52/1.1

Req 53	All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: In particular, apps that use WebView (Android) or UIWebView (iOS) are often more susceptible to XSS (cross-site scripting) vulnerabilities or attacks. The situation is similar with SQL / XML injection-based attacks. In all cases, vulnerabilities are exploited due to lack of input validation and sanitization. Therefore, it is necessary to validate and, if necessary, sanitize every input made on the client as well as the backend to prevent vulnerabilities from being exploited to compromise the application or the stored data.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04h-Testing-Code-Quality.md#injection-flaws-mstg-arch-2-and-mstg-platform-2

Android:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05h-Testing-Platform-Interaction.md#testing-for-injection-flaws-mstg-platform-2

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-53/1.1

Req 54	The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected.

Applies to apps that process the following data:

Protection class "public" and higher

Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: URL schemes represent a potential attack vector on apps. All URL parameters should therefore be validated and non-compliant URLs / URIs should be discarded. The available actions should also be restricted to those that do not pose a threat to the user's data. Other applications should not be given access to sensitive content or data of protection class "internal" or higher. When performing tests, it is essential to ensure that URL processing recognizes incorrectly formatted URLs / URIs and reacts appropriately (e.g., discard).

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05h-Testing-Platform-Interaction.md#testing-deep-links-mstg-platform-3
https://developer.android.com/training/app-links

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06h-Testing-Platform-Interaction.md#testing-custom-url-schemes-mstg-platform-3
https://developer.apple.com/documentation/xcode/defining-a-custom-url-scheme-for-your-app
https://developer.apple.com/documentation/uikit#//apple_ref/doc/uid/TP40007072-CH6-SW1

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-54/1.1

Req 55	The app does not export sensitive functionality through Inter Process Communication (IPC) facilities, unless these mechanisms are properly protected.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: The IPC system functionality offered by mobile application platforms should be used because it is much more mature than traditional techniques. Using IPC mechanisms with no security in mind may cause the application to leak or expose sensitive data, therefore all IPC must be protected accordingly.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05h-Testing-Platform-Interaction.md#testing-for-sensitive-functionality-exposure-through-ipc-mstg-platform-4
https://developer.android.com/guide/topics/manifest/activity-element#exported
https://developer.android.com/guide/topics/manifest/activity-element#prmsn
https://developer.android.com/guide/topics/manifest/permission-element#plevel

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06h-Testing-Platform-Interaction.md#testing-for-sensitive-functionality-exposure-through-ipc-mstg-platform-4
https://developer.apple.com/documentation/xcode/defining-a-custom-url-scheme-for-your-app

For this requirement the following threats are relevant:

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-55/1.1

Req 56	JavaScript is disabled in WebViews unless explicitly required.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: JavaScript can be injected into web applications via reflected, stored, or DOM-based Cross-Site Scripting (XSS). Mobile apps are executed in a sandboxed environment and don't have this vulnerability when implemented natively. Nevertheless, WebViews may be part of a native app to allow web page viewing. If the WebView implementation is not sufficiently secured and allows usage of JavaScript, JavaScript can be used to attack the app and gain access to its data.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05h-Testing-Platform-Interaction.md#testing-javascript-execution-in-webviews-mstg-platform-5
https://developer.android.com/reference/android/webkit/WebSettings#setJavaScriptEnabled%28boolean%29

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06h-Testing-Platform-Interaction.md#testing-ios-webviews-mstg-platform-5
https://developer.apple.com/documentation/webkit/wkwebview
https://developer.apple.com/documentation/safariservices/sfsafariviewcontroller

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-56/1.1

Req 57	WebViews are configured to allow only the minimum set of protocol handlers required (ideally, only https is supported). Potentially dangerous handlers, such as file, tel and app-id, are disabled.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: If a WebView is loading content from the app data directory, users should not be able to change the filename or path from which the file is loaded, and they shouldn't be able to edit the loaded file.

Implementation example: **Common:**

- Create a list that defines local and remote web pages and URL schemes that are allowed to be loaded.
- Create checksums of the local HTML/JavaScript files and check them while the app is starting up.

Android:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05h-Testing-Platform-Interaction.md#testing-webview-protocol-handlers-mstg-platform-6
https://developer.android.com/reference/android/webkit/WebSettings#setAllowContentAccess(boolean)
https://developer.android.com/reference/android/webkit/WebSettings#setAllowFileAccess(boolean)
https://developer.android.com/reference/android/webkit/WebSettings#setAllowFileAccessFromFileURLs(boolean)
https://developer.android.com/reference/android/webkit/WebSettings#setAllowUniversalAccessFromFileURLs(boolean)

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06h-Testing-Platform-Interaction.md#testing-webview-protocol-handlers-mstg-platform-6
https://developer.apple.com/documentation/webkit/wkwebview/1415004-loadhtmlstring
https://developer.apple.com/documentation/webkit/wkwebview/1415011-load
https://developer.apple.com/documentation/foundation/bundle/1410989-path
https://developer.apple.com/documentation/foundation/bundle/1411540-url

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-57/1.1

Req 58	If native methods of the app are exposed to a WebView, verify that the WebView only renders JavaScript contained within the app package.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: **Android:**

Android offers a way for JavaScript executed in a WebView to call and use native functions of an Android app (annotated with @JavascriptInterface) by using the `addJavascriptInterface` *method. This is known as a WebView JavaScript bridge or native bridge.*
Using `addJavascriptInterface`*, explicitly grants access to the registered JavaScript Interface object to all pages loaded within that WebView. This implies that, if the user navigates outside the app or domain, all other external pages will also have access to those JavaScript Interface objects which might present a potential security risk if any sensitive data is being exposed though those interfaces.*

iOS:

Apple introduced APIs that allow communication between the JavaScript runtime in the WebView and the native Swift or Objective-C objects. If these APIs are used carelessly, important functionality might be exposed to attackers who manage to inject malicious scripts into the WebView (e.g., through a successful Cross-Site Scripting attack).

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05h-Testing-Platform-Interaction.md#determining-whether-java-objects-are-exposed-through-webviews-mstg-platform-7

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06h-Testing-Platform-Interaction.md#determining-whether-native-methods-are-exposed-through-webviews-mstg-platform-7

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-58/1.1

Req 59	Object serialization, if any, is implemented using safe serialization APIs.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Serialization is not inherently secure. It is just a binary format (or representation) for locally storing data in a file.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05h-Testing-Platform-Interaction.md#testing-object-persistence-mstg-platform-8
https://developer.android.com/reference/java/io/Serializable.html

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06h-Testing-Platform-Interaction.md#testing-object-persistence-mstg-platform-8
https://developer.apple.com/documentation/foundation/archives_and_serialization
https://developer.apple.com/documentation/foundation/NSSecureCoding

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-59/1.1

Req 60	The app protects itself against screen overlay attacks. (Android only)

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Screen overlay attacks occur when a malicious application manages to put itself on top of another application which remains working normally as if it were on the foreground. The malicious app might create UI elements mimicking the look and feel and the original app or even the Android system UI. The intention is typically to make users believe that they keep interacting with the legitimate app and then try to elevate privileges (e.g by getting some permissions granted), stealthy phishing, capture user taps and keystrokes etc.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05h-Testing-Platform-Interaction.md#testing-for-overlay-attacks-mstg-platform-9
https://developer.android.com/reference/android/view/View#onFilterTouchEventForSecurity%28android.view.MotionEvent%29
https://developer.android.com/reference/android/view/View#attr_android:filterTouchesWhenObscured
https://developer.android.com/reference/android/view/View.html#setFilterTouchesWhenObscured%28boolean%29
https://developer.android.com/reference/android/view/MotionEvent.html#FLAG_WINDOW_IS_OBSCURED
https://developer.android.com/reference/android/view/MotionEvent.html#FLAG_WINDOW_IS_PARTIALLY_OBSCURED

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-60/1.1

Req 61	A WebView's cache, storage, and loaded resources (JavaScript, etc.) should be cleared before the WebView is destroyed.

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: In particular, data in the protection class "confidential" or higher requires special protection and should therefore be deleted before the WebView is closed. This ensures that neither affected areas in RAM or fixed memory contain artifacts that can be exposed by malicious activities.

Implementation example: **Android:**

https://developer.android.com/reference/android/webkit/WebView.html#clearCache%28boolean%29

iOS:

https://developer.apple.com/documentation/foundation/urlcache/1417802-removeallcachedresponses

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-61/1.1

Req 62	Verify that the app prevents usage of custom third-party keyboards whenever sensitive data is entered (iOS only).

Applies to apps that process the following data:

Protection class "confidential" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: There are plenty of useful keyboards available, but unfortunately their settings cannot be controlled and sensitive data or data with a protection class of "confidential" or higher could be unintentionally cached or shared with third parties.

Implementation example: **iOS:**

```
[size]func application(_ application: UIApplication, shouldAllowExten-
sionPointIdentifier extensionPointIdentifier: UIApplica-
tion.ExtensionPointIdentifier) -> Bool {
  switch extensionPointIdentifier {
    case UIApplication.ExtensionPointIdentifier.keyboard: return false
    default: return true
  }
}
```

https://developer.apple.com/documentation/uikit/uiapplication/extensionpointidentifier

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-62/1.1

3.7. Code Quality and Build Setting Requirements

Req 63	The app is signed and provisioned with a valid certificate, of which the private key is properly protected.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: **Android:**

Android requires all APKs to be digitally signed with a certificate before they are installed or run. The digital signature is used to verify the owner's identity for application updates. This process can prevent an app from being tampered with or modified to include malicious code.

iOS:

Code signing your app assures users that the app has a known source and hasn't been modified since it was last signed. Before the app can integrate app services, be installed on a non-jailbroken device, or be submitted to the App Store, it must be signed with a certificate issued by Apple.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05i-Testing-Code-Quality-and-Build-Settings.md#making-sure-that-the-app-is-properly-signed-mstg-code-1
https://developer.android.com/studio/publish/app-signing.html

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06i-Testing-Code-Quality-and-Build-Settings.md#making-sure-that-the-app-is-properly-signed-mstg-code-1
https://developer.apple.com/documentation/xcode/using-the-latest-code-signature-format

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-63/1.1

Req 64	The app has been built in release mode, with settings appropriate for a release build (e.g. non-debuggable).

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: While debugging is a useful feature when developing an app, it has to be turned off before releasing apps to the Apple App Store, Google Playstore or within an enterprise program. Building apps in release mode prevents access to an app and its data from debugging consoles.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05i-Testing-Code-Quality-and-Build-Settings.md#testing-whether-the-app-is-debuggable-mstg-code-2
https://developer.android.com/guide/topics/manifest/application-element.html#debug

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06i-Testing-Code-Quality-and-Build-Settings.md#determining-whether-the-app-is-debuggable-mstg-code-2
https://developer.apple.com/documentation/xcode/testing-a-release-build

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-64/1.1

Req 65	Debugging symbols have been removed from native binaries.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Generally, you should provide compiled code with as little explanation as possible. Some metadata, such as debugging information, line numbers, and descriptive function or method names, make the binary or bytecode easier for an reverse engineer to understand, but these aren't needed in a release build and should therefore be omitted without impacting the app's functionality.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05i-Testing-Code-Quality-and-Build-Settings.md#testing-for-debugging-symbols-mstg-code-3
https://developer.android.com/studio/build/shrink-code

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06i-Testing-Code-Quality-and-Build-Settings.md#finding-debugging-symbols-mstg-code-3

To prevent the inclusion of debug symbols, set `Strip Debug Symbols During Copy` to `YES` via the XCode project's build settings.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-65/1.1

Req 66	Debugging code and developer assistance code (e.g. test code, backdoors, hidden settings) have been removed. The app does not log verbose errors or debugging messages.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Debugging code, development artifacts, or code supporting development, such as generating verbose error or debug messages must be removed to ensure that the security and stability of the app is maintained in production. This helps avoid potential vulnerabilities and attack vectors that could be exploited by attackers.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05i-Testing-Code-Quality-and-Build-Settings.md#testing-for-debugging-code-and-verbose-error-logging-mstg-code-4
https://developer.android.com/reference/android/os/StrictMode.html

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06i-Testing-Code-Quality-and-Build-Settings.md#finding-debugging-code-and-verbose-error-logging-mstg-code-4

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-66/1.1

Req 67	All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Applications often make use of third party libraries which accelerate development as the developer has to write less code in order to solve a problem. However, third party libraries may contain vulnerabilities, incompatible licensing, or malicious content. Additionally, it is difficult to manage application dependencies, including monitoring library releases and applying available security patches.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05i-Testing-Code-Quality-and-Build-Settings.md#checking-for-weaknesses-in-third-party-libraries-mstg-code-5

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06i-Testing-Code-Quality-and-Build-Settings.md#checking-for-weaknesses-in-third-party-libraries-mstg-code-5

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-67/1.1

Req 68	The app catches and handles possible exceptions.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Exceptions occur when an application gets into an abnormal or error state. Testing exception handling is

about ensuring that the app will handle an exception and transition to a safe state without exposing sensitive information via the UI or the app's logging mechanisms.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05i-Testing-Code-Quality-and-Build-Settings.md#testing-exception-handling-mstg-code-6-and-mstg-code-7
https://developer.android.com/reference/java/lang/RuntimeException.html

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06i-Testing-Code-Quality-and-Build-Settings.md#testing-exception-handling-mstg-code-6
https://docs.swift.org/swift-book/LanguageGuide/ErrorHandling.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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-68/1.1

Req 69	Error handling logic in security controls denies access by default.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: Exceptions occur when an application gets into an abnormal or error state. Testing exception handling is about ensuring that the app will handle an exception and transition to a safe state without exposing sensitive information via the UI or the app's logging mechanisms.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05i-Testing-Code-Quality-and-Build-Settings.md#testing-exception-handling-mstg-code-6-and-mstg-code-7
https://developer.android.com/reference/java/lang/SecurityException

iOS:

https://developer.apple.com/documentation/swift/error
https://docs.swift.org/swift-book/LanguageGuide/ErrorHandling.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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-69/1.1

Req 70	In unmanaged code, memory is allocated, freed and used securely.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: **Android:**

Android applications often run on a VM where most of the memory corruption issues have been taken care off. This does not mean that there are no memory corruption bugs.

iOS:

iOS applications have various ways to run into memory corruption bugs: first there are the native code issues, various unsafe operations with both Objective-C and Swift to actually wrap around native code which can create issues and memory leaks due to retaining objects which are no longer in use.

Implementation example: **Common:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x04h-Testing-Code-Quality.md#memory-corruption-bugs-mstg-code-8

Android:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05i-Testing-Code-Quality-and-Build-Settings.md#memory-corruption-bugs-mstg-code-8

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06i-Testing-Code-Quality-and-Build-Settings.md#memory-corruption-bugs-mstg-code-8

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-70/1.1

Req 71	Free security features offered by the toolchain, such as byte-code minification, stack protection, PIE support and automatic reference counting, are activated.

Applies to apps that process the following data:

Protection class "public" and higher
Sensitive and / or personal data
Intellectual Property Rights, highly sensitive and business-critical data

Motivation: While the latest installations of the development environments for Android and iOS already have the corresponding settings preset by default, it may be that the security mechanisms are still disabled on updated installations or have been deactivated on a test basis.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05i-Testing-Code-Quality-and-Build-Settings.md#make-sure-that-free-security-features-are-activated-mstg-code-9

iOS:

https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06i-Testing-Code-Quality-and-Build-Settings.md#make-sure-that-free-security-features-are-activated-mstg-code-9

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-71/1.1

3.8. Resilience Requirements

Req 72	The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous development while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, the apps must ensure that the content is adequately protected. On devices with an active root/jailbreak, this is not ensured and the app must inform the user of this and refuse to run.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md#testing-root-detection-mstg-resilience-1

iOS:

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md#jailbreak-detection-mstg-resilience-1

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-72/1.1

Req 73	The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous development while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, the apps should prevent debugging or react appropriately to debugging attempts in order to make reverse engineering attempts and thus tests for possible vulnerabilities and attack possibilities correspondingly more difficult.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md#testing-anti-debugging-detection-mstg-resilience-2

iOS:

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md#testing-anti-debugging-detection-mstg-resilience-2

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-73/1.1

Req 74	The app detects, and responds to, tampering with executable files and critical data within its own sandbox.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous development while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, the apps should detect manipulations and react appropriately to ensure that injected code is not executed. This prevents the app or data from being compromised and the user from being harmed as a result.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md#testing-file-integrity-checks-mstg-resilience-3

iOS:

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md#file-integrity-checks-mstg-resilience-3-and-mstg-resilience-11

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-74/1.1

Req 75	The app detects, and responds to, the presence of widely used reverse engineering tools and frameworks on the device.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous development while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, the apps should detect installed reverse engineering tools, apps or frameworks and react appropriately to them to ensure that these measures significantly increase the analysis effort and thus deter potential attackers.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md#testing-reverse-engineering-tools-detection-mstg-resilience-4

iOS:

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md#testing-reverse-engineering-tools-detection-mstg-resilience-4

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

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-75/1.1

Req 76	The app detects, and responds to, being run in an emulator.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous development while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, the apps should implement emulator detection to increase the difficulty of running the app on an emulated device, which impedes some tools and techniques reverse engineers like to use. This increased difficulty forces the reverse engineer to defeat the emulator checks or utilize the physical device, thereby barring the access required for large-scale device analysis.

While only Android devices could be emulated comfortably in the past, there are now providers like Corellium that offer corresponding services not only for Android, but also for iOS devices, including jailbreaking and large scale. **For this reason, corresponding protection is also necessary in iOS apps**.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md#testing-emulator-detection-mstg-resilience-5

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-76/1.1

Req 77	The app detects, and responds to, tampering the code and data in its own memory space.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous devel-

opment while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, apps should verify the integrity of the their memory space to defend the app against memory patches applied during runtime. Such patches include unwanted changes to binary code, bytecode, function pointer tables, and important data structures, as well as malicious code loaded into process memory.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md#testing-runtime-integrity-checks-mstg-resilience-6

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-77/1.1

Req 78	The app implements multiple mechanisms in each defense category (Requirements from 72 to 77). Note that resiliency scales with the amount, diversity of the originality of the mechanisms used.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous development while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, apps should implement several mechanisms from requirements 72 to 77 to appropriately and adequately increase resilience against attackers.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-78/1.1

Req 79	The detection mechanisms trigger responses of different types, including delayed and stealthy responses.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous development while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, the resilience mechanisms implemented for this purpose should trigger various reactions in order to repeatedly set new hurdles for attackers and thus deter them accordingly.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-79/1.1

Req 80	Obfuscation is applied to programmatic defenses, which in turn impede de-obfuscation via dynamic analysis.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous development while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, the apps should implement obfuscation measures to adequately respond to attempts to de-obfuscate and thus prevent reverse engineering.

Implementation example: **Android:**

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md#testing-obfuscation-mstg-resilience-9

iOS:

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md#testing-obfuscation-mstg-resilience-9

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

- Unauthorized use of services or resources

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-80/1.1

Req 81	The app implements a 'device binding' functionality using a device fingerprint derived from multiple properties unique to the device.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous development while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, the apps should implement a fingerprint of the device that is determined from various properties of the device and cannot be duplicated on another device. Execution of the app or security-critical areas can thus be prevented even after duplication.

Implementation example: **iOS:**

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md#device-binding-mstg-resilience-10

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-81/1.1

Req 82	All executable files and libraries belonging to the app are either encrypted on the file level and/or important code and data segments inside the executables are encrypted or packed. Trivial static analysis does not reveal important code or data.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous development while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, the apps should utilize encryption of important files and code segments to prevent third parties from drawing conclusions about the security mechanisms even in the course of a static analysis.

Implementation example: **iOS:**

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md#file-integrity-checks-mstg-resilience-3-and-mstg-resilience-11

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-82/1.1

Req 83	If the goal of obfuscation is to protect sensitive computations, an obfuscation scheme is used that is both appropriate for the particular task and robust against manual and automated de-obfuscation methods, considering currently published research. The effectiveness of the obfuscation scheme must be verified through manual testing. Note that hardware-based isolation features are preferred over obfuscation whenever possible.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous development while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, the apps should use obfuscation methods to adequately prevent deobfuscation attempts. This makes reverse engineering activities and thus tests for possible vulnerabilities and attack possibilities correspondingly more difficult.

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-83/1.1

Req 84	As a defense in depth, next to having solid hardening of the communicating parties, application level payload encryption can be applied to further impede eavesdropping.

Applies to apps that process the following data:

Intellectual Property Rights, highly sensitive and business-critical data

Apps that process sensitive data or copyright-protected content must provide special protection for the data being processed. These include, for example, banking and insurance apps, health apps, video and music players and eReaders, and messengers. These apps require a higher level of security to adequately protect them against analysis and attacks.

Ready-made SDKs such as Trusteer or no-code solutions such as Appdome offer the advantage of continuous development while developers can concentrate on the core functions of the app.

If there are no special protection requirements for the app, the implementation of the requirement is not mandatory!

Motivation: To achieve the described level of protection, the apps should also encrypt the transmitted data (payload) before transmission (end-to-end encryption) in the context of a TLS-encrypted connection in order to defend the transmitted data itself against possible man-in-the-middle attacks. If the TLS connection were to be compromised by a third party, the transmitted data would still be encrypted and would not be readily visible to the attacker.

Implementation example: **Android:**

https://developer.android.com/guide/topics/security/cryptography

iOS:

This is a simple example using CloudKit. Depending on the intended use, iCloud features may not be available, especially in enterprises:
https://github.com/apple/sample-cloudkit-encryption
https://support.apple.com/de-de/guide/security/sec3cac31735/web

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

For this requirement the following warranty objectives are relevant:

ID: 4.1.1-84/1.1