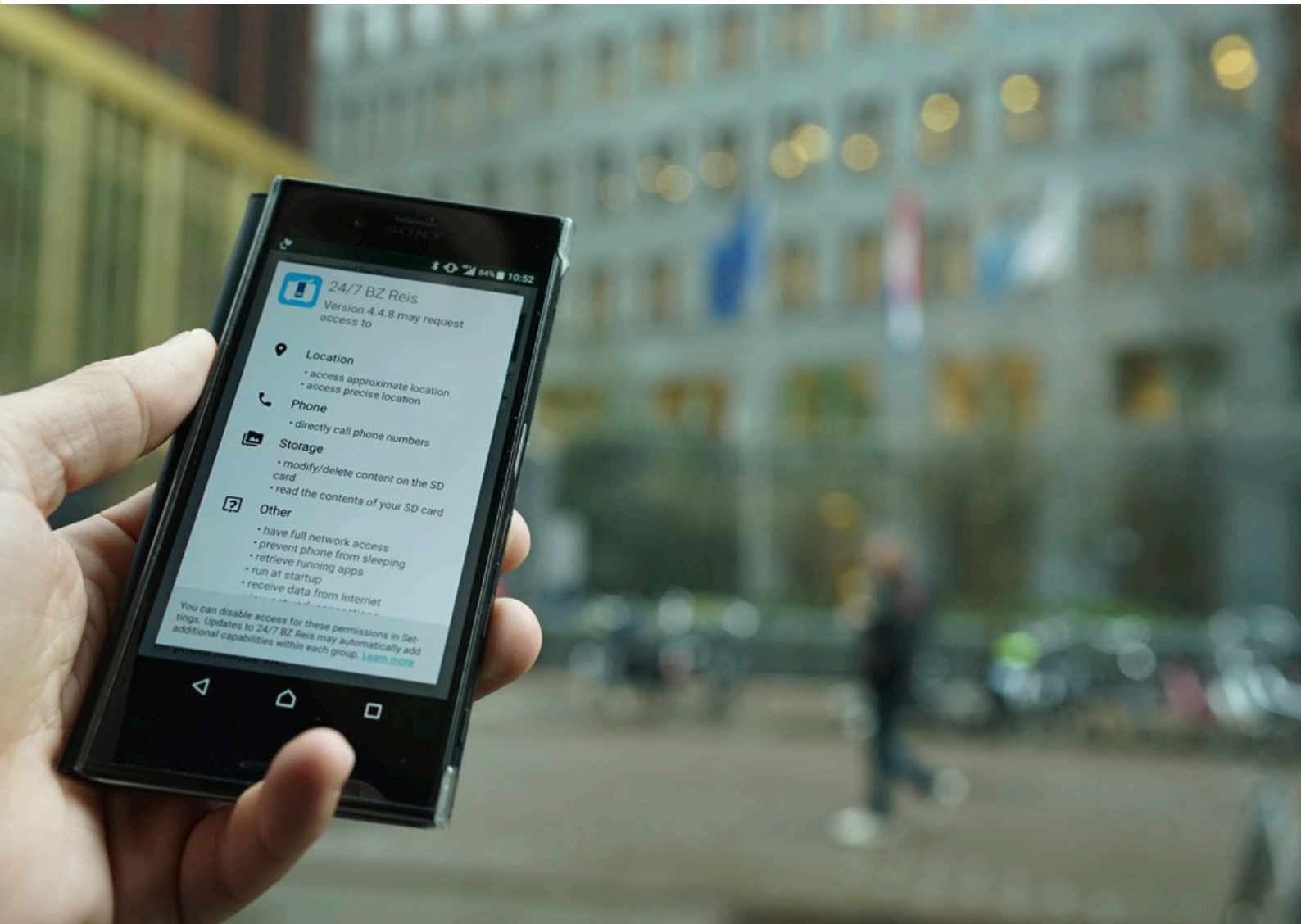
National Cyber Security Centre
Ministry of Justice and Security

IT Security Guidelines for Mobile Apps



National Cyber Security Centre

The National Cyber Security Centre (NCSC), in collaboration with the business community, government bodies and academics, is working to increase the ability of Dutch society to defend itself in the digital domain.

The NCSC supports the central government and organisations in the critical infrastructure sectors by providing them with expertise and advice, incident response and with actions to strengthen crisis management. In addition, the NCSC provides information and advice to citizens, the government and the business community relating to awareness and prevention. The NCSC thus constitutes the central reporting and information point for IT threats and security incidents.

The NCSC is part of the Cyber Security Department of the National Coordinator for Security and Counterterrorism (NCTV).

Partnership

This publication was produced in collaboration with the following partners:

- Centre for Information Security and Privacy Protection (Centrum Informatiebeveiliging en Privacybescherming, CIP)
- Dutch Tax and Customs Administration (Belastingdienst)
- Rabobank
- ING
- Centric
- CERT.be
- Dutch Ministry of Defence
- ICTU
- Dutch Directorate-General for Public Works and Water Management (Rijkswaterstaat)

IT Security Guidelines for Mobile Apps

Table of contents


Home - Living Room
21.5°
52%
23%
on
locked
38%
54%
2/6
3/5

Introduction

The number of mobile apps has grown explosively in recent years. Many organisations release apps to simplify their services for customers, to facilitate contact, to entertain their target audience and much more. As mobile devices play an increasingly important role in people's lives, this situation means that malicious parties are finding these platforms increasingly attractive to attack digitally. For this reason, it is important that both the mobile devices and the apps running on them are secure. These IT Security Guidelines for Mobile Apps provide measures to protect apps and their users from various attacks.

Target audience

This document has three primary target audiences.

- The first target audience consists of parties responsible for establishing security frameworks and monitoring compliance with them. This group could include security managers and system owners (the clients) of the IT services to be provided.
- The second target audience consists of those involved in the design and development process, implementation and management of mobile apps. This target audience must apply the security guidelines.
- The third target audience consists of the enforcement agencies (IT auditors) who perform an objective IT security assessment based on these guidelines.

Scope and context

These guidelines focus on the protection of mobile apps, i.e. the portable software that is run on a mobile device. The guidelines do not focus on the server-side back end of the app, nor on the configuration of the mobile device itself. Therefore, these guidelines do not include direct security measures for servers or for the way a device must be set up to use apps securely. The NCSC publishes separate guidelines for this purpose.[1]

These guidelines are not comprehensive and can be used in addition to security regulations and baselines with a broader scope, such as the 'BIR' (Civil Service Baseline Information Security) and the 'BIG' (Dutch Municipalities Baseline Information Security). Such baselines are a subset of the measures from ISO standard 27002. Although these guidelines partially overlap with ISO 27002 and the baselines, certain aspects have been elaborated in more detail.

The guidelines offer a more in-depth treatment of mobile app security. App security must fit in with the security designs already put in place by organisations for their other processes and environment; for example, based on ISO 27002.

These guidelines are primarily technical in nature. This fact means that a number of information security aspects are not covered by the framework used in these guidelines. For example, the framework pays little or no attention to matters such as the security organisation, physical security and personnel. Non-technical measures are only included when deemed necessary for the technical context or where other standards frameworks or standards do not sufficiently address the particular issue. If a risk analysis results in the implementation of these additional security measures, reference is made to other security standards such as ISO 27001 and ISO 27002.

These guidelines are the basis for securing mobile apps. An organisation can test the security of its apps (or have it tested) based on these guidelines. The assessment organisations can use these guidelines to perform an objective security assessment. These guidelines can be referred to when assessing a specific situation and when implementing the guidelines (i.e. solving shortcomings).

Application

Organisations can turn parts of these guidelines into a standards framework for certain areas of application. Unlike security guidelines, which are of an advisory nature, a standards framework is mandatory for the area of application. The guidelines can also be used in tenders, the outsourcing of services and in mutual agreements on chain processes. Depending on the nature and specific characteristics of the service concerned, security guidelines can be selected and the weighting factors of the individual security guidelines adjusted to reflect the desired situation.

1 If the app communicates with the back end via HTTPS, the server side can be seen as a web application. This service can be based on the IT Security Guidelines for Web Applications [1]. Mobile devices are usually outside the control of the issuing organisation and therefore cannot be centrally secured. However, if mobile devices are managed by an organisation, the Security Guidelines for Mobile Devices [2] can be used.

Priority

In general, the priority of each security guideline is weighted according to the classification High, Medium or Low. These three classifications form three points on a scale, where 'High' is the strongest degree of desirability (must have), 'Medium' is a reasonably strong degree of desirability (should have), and 'Low' is a desired, but not necessary condition (nice to have). The three values are difficult to define precisely, but they are a function of the probability of a threat occurring and the possible damage resulting from it.

The final decision to use a specific app for a specific organisation will depend on the weighting of risks that emerge from a risk analysis. This procedure involves looking at the chance of a threat occurring, the interest to be defended and the possible impact of this threat on business. The security guidelines provide the measures that can be taken to reduce the occurrence of threats or at least limit the impact in the event of a threat occurring.

As an example of an adaptation of the general classifications in specific situations, availability measures can be considered. For example, the need for availability measures may be low in situations where the unavailability of a service has little impact on business operations. By contrast, the need can be high in situations where the impact and likelihood of a threat occurring are severe.

Outline of the document

The guidelines are classified according to the SIVA framework. [4] The structure of the SIVA framework consists of three domains.

Policy domain

This domain includes elements that indicate what can be achieved across the organisation. It therefore contains elements that are essential to the other layers, such as objectives, information security policy, strategy and innovation, organisational structure and architecture.

Implementation domain

The implementation of the mobile app is set out in this domain. Of the three domains, the implementation domain is the one discussed in this document. The policy domain and control domain are dealt with in a separate document. [3]

Control domain

This domain deals with evaluation aspects and measurement aspects. In addition, it also describes the management processes that are necessary for the maintenance of IT services. The information from the evaluations and management processes is not only focused on adjusting the implemented mobile apps, but also on adjusting or adapting the previously formulated conditional elements.

Relationship to other documents

These guidelines have been compiled on the basis of the document 'Grip op Secure Software Development (SSD)' ('Grasping SSD'), and the Mobile App Standards of the Centre for Information Security and Privacy Protection (CIP). [5] The Implementation domain of this document corresponds to the CIP standards framework. The NCSC and CIP provide for the joint and coordinated maintenance of the guidelines so as to ensure that mobile apps, if they comply with one of the guideline sets, are also inherently technically compliant with the others.

The NCSC IT Security Guidelines for Web Applications [1] are linked to these guidelines. They draw on a common policy and control domain; the implementation domains focus on the various architectural components of an application environment.

Implementation domain Mobile apps

U/MA.01 Operational policy for mobile apps

The operational policy for mobile apps describes the way that the organisation deals with designing apps and making them available. The operational policy is a more concrete elaboration of the higher-level policy. A solid operational policy is therefore a prerequisite for the safe design of an app and its environment.

U/MA.01 Operational policy for mobile apps

Criterion *(Who and what?)*	The operational policy for mobile apps contains **guidelines** and **instructions and procedures** for the development, maintenance and phasing out of apps.
Objective *(Why?)*	Supporting the development of the app optimally and providing the customer with reliable services.
Risk	No clear direction for apps, so the app's security is unrelated to its surroundings. This situation increases the risk of security incidents.
Classification	High

Measures

Guidelines

01 Prepare guidelines for:
- development, maintenance and phasing out of apps;
- app security;
- data processing;
- links with underlying systems.

When developing apps, use methods for secure software development.
Underlying systems are systems that are made accessible or reached directly or indirectly by the app.

Instructions and procedures

02 Prepare instructions and procedures:
- working with separate Development, Testing, Acceptance and Production environments (DTAP);
- content management.

Pay attention to regularly reviewing and updating the guidelines.
If these instructions and procedures allow room for derogation from the guideline, the requirement of 'comply or explain' should be linked to this process.

U/MA.02 Secure server-side application

The app on the mobile device usually forms a chain together with the application on the server side. Safe use of the app is only possible in combination with a secure server-side application.

The absence of a secure server-side application or secure server will result in the inability to build a secure chain and thus in an environment that is not suitable for storing and exchanging confidential information.

Attention to the information security of the server application will prevent requirements being imposed on the server application that conflict with the information security of the entire chain from mobile device to server.

U/MA.02 Secure server-side application

Criterion *(Who and what?)*	When building the app, the **security requirements** of the application on the server are observed by the developer.
Objective *(Why?)*	During the development of the app, only design choices are made that will not compromise the security of the server or the server application. This way, the security of the chain from app to server-side application is guaranteed.
Risk	The building of an app makes it impossible or more difficult to design the server-side application in a secure manner.
Classification	High

Measures

Security requirements

01 Prevent requirements imposed on the server-side application from causing a breach of compliance with the IT Security Guidelines for Web Applications. [1]

02 As client, ensure that the server-side applications used by the app comply with the IT Security Guidelines for Web Applications.

The implementation of the guidelines for web applications on the server is a prerequisite for a secure chain.

U/MA.03 Third-party apps

Apps often work together with other apps such as viewers and keyboards. These apps usually come from other suppliers and are referred to as 'third-party apps'. Such apps process information outside the app, which means that the protection of the information lies beyond the developer's control. These apps may have hidden functionality, allowing access to confidential information even if it is protected by the app.

When choosing such apps, the benefits and the risks must therefore be weighed up. For example, an advantage can be cost savings. Moreover, building your own functionality does not necessarily mean a safer app. Building a secure app is no easy task in many situations, such as with cryptographic tools.

Third-party apps may require permissions or have hidden functionality, allowing access to confidential information even if it is protected by the app. They may also make use of unsafe APIs and vulnerabilities in apps or the operating system. The basic principle is that third-party apps should in principle be considered unsafe unless the building process and source code have been inspected. This fact is particularly true if users make use of third-party apps themselves in combination with the app, such as third-party keyboards.

U/MA.03 Third-party apps

Criterion *(Who and what?)*	The use of third-party apps is based on a **risk assessment**.
Objective *(Why?)*	Preventing third-party apps from gaining unauthorised access to information that must be protected by the app.
Risk	Confidential information is made accessible to attackers via a third-party app.
Classification	High

Measures

Risk assessment

01 Check the third-party apps for vulnerabilities, such as hidden functionality and unsafe APIs. Determine the risks.

The choice of third-party apps is based on balancing the safe operation of the app and the benefits of using existing third-party apps.

The risks are assessed by looking at the following aspects:

- *Can the third-party apps and their code be tested on delivery of the app?*
- *Is it also possible to test any patches for or new versions of the third-party apps' code after the first use of the third-party app?*
- *Have agreements on keeping the app up to date been laid down in a contract and has an adequate governance process been set up?*
- *Does the third-party app comply with the security guidelines?*

02 Base the choice of third-party apps on a risk analysis. Record the results of the risk analysis.

The results of the vulnerability checks were used in the risk analysis (measure 01). The benefits of third-party apps must also be included in the risk assessment, such as the security features built into the third-party app, when proven to be effective in practice.

03 When selecting third-party apps, pay special attention to third-party keyboards.

On mobile devices, in addition to the standard keyboard belonging to the operating system or provided by the supplier, it is often possible to use third-party alternatives (third-party keyboards). These third-party keyboards can record keystrokes and can thus leak or misuse data.
Users can often choose alternative third-party keyboards themselves. If the use of third-party keyboards cannot be prevented, choose to have any highly confidential information entered within the user interface of the app.

U/MA.04 Secure code on delivery

App development can be accelerated by using external code libraries. However, these libraries may contain vulnerabilities or malware. Information about the libraries used and the way that the app works can provide the attacker with information on weaknesses in the app.

The security of the code must be guaranteed when using existing code or code developed elsewhere.

U/MA.04 Secure code on delivery

Criterion *(Who and what?)*	The developer uses only **trusted source code** libraries and the source code does not contain any **technical information** on the operation of the app.
Objective *(Why?)*	The source code used is secure and does not disclose any information about the internal operation of the app.
Risk	Third-party source code contains vulnerabilities, malware or information that allows an attacker access to the operation of the app or to confidential data.
Classification	Medium

Measures

Trusted source code

01 Use only code whose origin is known and whose security has been established.

The source is known and of the version used there are no weaknesses known within the industry that constitute a threat.
Do not trust closed source components from third parties, unless it is known exactly what the components do. An analysis of open source components must have been performed which shows that the software can be regarded as secure.

02 Keep the code and code libraries that are used up to date, so they do not contain known vulnerabilities.

03 Determine the origin and security of mobile code downloaded by the app.

Measure 01 also applies to any software downloaded during the building of an application in terms of the reputation of its origin.
The usual method to ensure this aspect is code signing. This process is a security feature on the device that prevents attempts to run unauthorised applications on the device by validating the signature on the app. Validation is performed every time that the app is called. This fact means that apps can only execute code with a valid, trusted signature.

04 Ensure that the settings in the configuration file of the app in question guarantee optimum safety.

The settings for debugging, permissions and the security options of the configuration settings are considered here.

05 Publish the hash of the downloaded app binary and sign the hash as well as the app binary, so it is clear from whom the app originates.

A hash is calculated on the basis of the binary. Changing the binary will usually always result in a different hash value (depending on the quality of the hash algorithm). This procedure allows you to check whether the binary has changed.
In addition, the hash and the binary themselves can be secured by signing them. The binary and hash are provided with a digital signature unique to that hash by means of a cryptographic key.
Not every user checks the digital signature and thus the authenticity of the app. It is therefore useful to perform periodic searches for imitated or copied and modified apps in the app stores.

Technical information

06 Do not store sensitive technical information in the app. If storage is considered necessary, this process is done on the basis of a risk assessment.

Pay attention to cryptographic keys, passwords, internal URLs, and so on. Because a binary can never be completely protected, security measures and security settings must never be stored in the code itself.

07 Do not store security settings and measures in files that fall outside the scope of the app's signature and lie beyond the protection of the operating system.

08 Shield information on the operation of the app and relevant confidential information that must be protected in the binary to prevent reverse engineering and manipulation.

As a rule, obfuscation cannot be used as a replacement in situations where encryption is required.
When delivering the app to the app store, the app is digitally signed and offered via a secured connection and account. The app store must know the public key (and the algorithm) so the app store can inspect the app and, after approval, publish it. In the Apple certificate, Apple also specifies the supplier's identity. This policy has allowed Apple to limit apps being offered to members of the iOS Developer Program.
Keep in mind that the binary on a jailbroken mobile device is stored within the memory in unencrypted form and is therefore accessible to malicious parties.

U/MA.05 Secure operation of the app

Unlike server-side applications, apps are not run in a familiar environment but on the mobile device itself. This situation allows an attacker to obtain the full source code: the binary and the running app. As a result, the attacker is able to control each piece of code during each phase of the app's program, plus the information stored in the binary files of the app such as configuration files. Manipulation of the app's operation by malicious parties can therefore only be limited by electronically protecting the binary and the running app.

U/MA.05 Secure operation of the app

Criterion *(Who and what?)*	The app on the mobile device is shielded from unwanted or unintended **manipulation** and the results of the **business logic** are always checked on the server.
Objective *(Why?)*	Preventing the operation of the app from being unintentionally affected.
Risk	The confidentiality of the information, the correct functioning of the app and the integrity of the output fall prone to the influence of an attacker.
Classification	High

Measures

Manipulation

01 Use Position-Independent Code (PIC) and Address Space Layout Randomisation (ASLR).

The decision whether or not to include ASLR in parts or all of the code is based on a risk assessment.

Position-Independent Code (PIC) provides the option to run the code from any location in the memory. PIC makes it possible to give the code a constantly changing position, so an attacker cannot exploit the memory location of the code. Apps that have PIC enabled are configured as a Position-Independent Executable (PIE).

Address Space Layout Randomisation (ASLR) is a security feature which ensures that the PIE code has an ever-changing location in the process memory. Enabling ASLR makes it harder to carry out attacks using fixed, predictable code locations in the memory, such as buffer overflows.

Business logic

02 Check decisions based on critical business logic in the app within a secure environment on the server.

While protecting the app minimises the risk of the business logic being changed, there is no guarantee that the logic in the app will remain intact in the event of a hacker attack.

U/MA.06 Storage location

The choice where to store data will to a large extent determine the options when it comes to protecting the data from unwanted access. The choice where to store each data item or set must therefore be made consciously. Because better protected locations offer greater security, the basic principle in this security guideline is that the safest location should be chosen for storage, unless it is certain that a less secure location can be demonstrably suitably secured.

Storage on the mobile device through its own functionalities is more difficult to shield from anyone who has access (physically or via malware) to the device than a place that is electronically and physically protected.

Appropriate shielding of data may be provided when its confidentiality has been established.

U/MA.06 Storage location

Criterion *(Who and what?)*	The selection of the **location** where the data and information on the logic of the app are stored is based on the principle of **confidentiality**.
Objective *(Why?)*	Confidential information and critical logic are only stored in locations where effective security measures can be taken.
Risk	Confidential information or critical logic comes into the hands of unauthorised parties.
Classification	Medium

Measures

Location

01 Only store sensitive logic or sensitive information about the mobile device if necessary for the operation of the app.

02 As a rule, store confidential data and sensitive information about the logic of the app in a server-side secure environment.
All data and logic are stored on the back end if a risk analysis determines that the consequences of this information being made public are unacceptable.

03 A basic principle to be used in the design is that local storage on the mobile device must be prevented as much as possible. For each data item or set, the assessment is made whether offline local availability is required or can be avoided.

04 In a risk analysis, determine which protection is required if confidential information is stored on the mobile device.
The risk analysis also takes into account the operation of components used by the app and thus also the data stored behind the scenes, unnoticed by the user (caching, logging, and so on).
The risk analysis may then, for example, lead to the encrypted storage of information.

05 Store confidential data on the device in the app's internal storage, unless the required level of protection (measure 04) indicates otherwise. It has been taken into account that the standard folders can be part of backups or synchronisation with the cloud or with a linked computer.
By using a sandbox, the app and information in the app are protected. A sandbox is a container on the mobile device that is protected by encryption. Even if an attacker has access to the device, they cannot access the data as long as the key used to encrypt the sandbox is protected.
On the mobile device, the 'internal storage' is the storage within the folder in which the app is installed. This folder, together with the app, is protected by the sandbox. Other apps do not have access to these files unless the device has been jailbroken or rooted, or vulnerabilities in the operating system are abused.
As a rule, certain folders in the internal storage are included in backups or synchronisation with the cloud or with a trusted computer. Refer to the platform documentation to select a folder within the internal storage which is appropriate based on functional and risk considerations.
If the key of a sandbox is safely stored on a back end (measure 01), it is also shielded if the device has been jailbroken or rooted. As a result, this situation offers the possibility of better security.

06 Limit the storage of data on the device outside the internal storage of the app to non-confidential information, or to information that users know that they must manage and therefore protect.

07 Only allow storage of confidential information in a public cloud if storage on the server (measure 02) is not possible, and only after the advantages as well as the disadvantages for the client and user – including privacy aspects – have been determined in advance and a positive assessment has been made. This measure also applies to backups.
Storage on the mobile device and in the public cloud, such as on social media, but also other third-party storage is considered unsafe for confidential information. In all circumstances, the data should be encrypted prior to storage.

Confidentiality

08 Specify the classification of the data confidentiality in or around the app.

The specification clarifies which data must be protected.
If classifications are used, the security requirement is defined per classification.
The analysis also includes data backups.
In practice, it will not be known what the classification is for each data item, especially if it is a data item used for the technical operation of the app.

09 Treat data as confidential and secure this information as such if confidentiality has not been established.

The analysis also includes the business logic in the software.

U/MA.07 Storage on the mobile device

Sensitive data can be protected by using cryptographic techniques. Cryptographic techniques are the only way to protect information efficiently when it is physically accessible.

Which data are sensitive or confidential must be determined by the organisation. Confidentiality is determined as part of determining the location of the storage (see U/MA.06 Storage location).

U/MA.07 Storage on the mobile device

Criterion *(Who and what?)*	When storing confidential information on the mobile device, confidential data is protected by using **cryptographic techniques**.
Objective *(Why?)*	Prevent unauthorised use, viewing, alteration and loss of confidential data that have been unsafely stored.
Risk	The confidential data in the mobile app come into the hands of unauthorised parties.
Classification	High

Measures

Cryptographic techniques

01 Protect confidential information with at least a PIN.

Protection based on the usual PIN screen lock is relatively easy to break through brute force. Additional measures are required for highly confidential information such as medical and identity data. For this reason, secret keys and identity data are not stored in the code.
The app and the information can be protected with a key comprising a combination of the PIN and unique features of the device. This derived key is used to encrypt the actual key.

02 Do not store encryption keys on the mobile device.

To protect stored data by encryption, a private encryption key must be stored in a secure location.
If the encryption key is stored on the mobile device, its protection can once again easily be circumvented. The key must therefore be stored at a different location; for example, on another device of the user or on a server. This process can be done, for example, by storing it in the user's profile.

03 Consider the use of a passphrase or of delaying key stretching algorithms.
The time needed to break in through brute force is increased by virtue of a passphrase or of delaying key stretching algorithms such as PBKDF2 or bcrypt. A passphrase uses a string of words instead of a password.

04 Generate a temporary key for temporary files and keep the key in the internal memory until the app is closed.
Use a cryptographically strong random number generator to generate the key.

05 Only store confidential information in a database if the database is encrypted.
SQLite databases are often used without encryption. It is possible to encrypt the entire database. However, it is preferable to encrypt the data fields individually and not to keep the decrypted data in the memory any longer than necessary for processing.

06 Use well-known, proven algorithms and the implementation of these algorithms for encryption. No self-developed cryptographic functions must be used.
The robust implementation of cryptographic functions is very complex. It is usually more responsible to use either the cryptographic functions of the operating system, or known and proven cryptography libraries by third parties.

U/MA.08 Unnecessary information in RAM

The temporary storage of confidential information in the mobile device's memory is unavoidable for most apps. However, attacks are known where a memory dump is made while the device is locked. Confidential information stored in the memory at that time (such as PINs) can then be obtained by an attacker. Confidential information is also accessible after a crash, for example.

U/MA.08 Unnecessary information in RAM

Criterion *(Who and what?)*	Confidential information must be **kept to a minimum** in the cache memory on the basis of a **risk assessment** per data item; this rule applies both inside and outside the internal app.
Objective *(Why?)*	Preventing sensitive information from becoming and remaining accessible via the cache memory.
Risk	Temporarily stored sensitive information comes into the hands of unauthorised parties.
Classification	Medium

Measures

Risk assessment

01 Restrict the temporary availability of confidential information in RAM to a minimum and base it on a risk assessment per data item.
In the risk assessment, take in account whether the duration of the temporary storage will be extended; for example, because a service or access to a file or network is unavailable for a longer period.

02 Determine for how long information is stored in the cache memory and whether this information will be made unreadable, based on the confidentiality of the information; for example, by overwriting it through data wiping.
Java has a garbage collection mechanism for clearing up leftover data. Because it may take some time for the garbage collector to be activated, this measure is not considered sufficient for the deletion of confidential data. As a consequence, a text field in Java should not be converted into a string if the text field contains confidential data; instead, modifiable sequences or arrays should be used.

03 Where possible, mask data through truncating.
Truncating means that part of the characters of the confidential data item are not stored, so the remaining characters do not compromise confidentiality; for example, by saving only the last four digits of a credit card number.

Kept to a minimum

04 Only store sensitive information necessary for the operation of the app.
Do not save confidential information in, for example, the cookies, web history, web cache, property files and SQLite files present.

05 Do not store any confidential data on the mobile device that is also stored on a server.
WebViews and JSON parsing can be used here. The data remain on the server as much as possible and are viewed with WebViews or JSON parsing.
It is also possible not to bring back confidential data into the app, but to process the confidential data on the server side and only show the results in the app.
This measure combines well with measure U/MA.04/02.

06 Avoid the storage of confidential information for the purposes of auto-complete functionality.
When entering text into a text field, the data can be saved and used by the auto-complete feature. This procedure also allows confidential information to be presorted and thus made visible. To prevent this risk, an input field can be classified as a 'password'.
This situation cannot always be avoided when using third-party keyboards.

07 Avoid caching HTTPS traffic.

08 Delete highly confidential information or make it illegible in the screenshot when the application is moved to the background.
Many operating systems keep a screenshot of an app as soon as it is sent to the background. This process allows confidential data displayed on the screen at that time to be viewed. To this end, we recommend that you switch off the 'screen captures on backgrounding' feature. This setting can often be toggled by the user. Highly confidential information can be protected by means of an overlay or by replacing the screenshot with the splash screen of the app.

U/MA.09 User session timeouts

While the user is accessing the app, a user session is open. During this session, information is accessible via the app. Other people may abuse this accessibility. Session termination ensures that the session ends after a prescribed time interval of inactivity.

Inactivity in the user session is a period without user interaction with the app.

U/MA.09 User session timeouts

Criterion *(Who and what?)*	The app terminates a user session after a **predetermined period** of user inactivity via **automatic session termination**.
Objective *(Why?)*	Preventing a session from being accessible to other people even for a limited time only when the user has left the session (on the device) unattended.
Risk	The open session is abused by malicious parties.
Classification	Medium

Measures

Predetermined period

01 Determine the session timeout period using a risk analysis.
A timeout of 2 to 5 minutes is normally used for sensitive apps. In the event of a low risk, this period is 15 to 30 minutes.

02 Substantiate the need for a longer predetermined period.
The substantiation is drawn up by the client and coordinated with the developer.

Automatic session termination

03 The app automatically terminates the session after a period of inactivity prescribed by the client.

04 Session termination corresponds to logging out the user.

05 After session termination, the login screen of the app is shown again when screen activity is resumed, and the user must re-authenticate.

U/MA.10 Logging

While logging, user actions and messages about the app's operation can be recorded in log files. Logging can be used to track security incidents and errors in the operation of the app, but sensitive information can also end up in the log files. If this information falls into the wrong hands, not only would this situation result in the privacy of the user being at risk, but the information could also be used to find security weaknesses in the app. Access to this sensitive log information should therefore be prevented.

Debug logging is a function to record activities and events in the app. Debugging offers the ability to detect events and errors during the development, testing or use of the app. Activities and events are recorded in a debugging log.

U/MA.10 Logging

Criterion *(Who and what?)*	**Debug logging** is deactivated **before taking into production** and the log files are deleted. When **statistical logging** on the use of the app takes place, these logs do not contain any personal data.
Objective *(Why?)*	Preventing unnecessarily detailed information on the operation of the app or on the user from becoming available to an attacker.
Risk	Sensitive information or information on the weaknesses in the app's security comes into the hands of an attacker.
Classification	Low

Measures

Debug logging

01 Switch off the logging mechanisms for debugging upon delivery.
Debug logging is used to find errors. To prevent various privacy and security risks, this logging must be disabled when the app is commissioned.
Remember that debugging must also be disabled for all libraries used.

02 Do not record sensitive information on the operation or the user.
It is preferred not to log any information. If the developer does decide to log information, they must inform the client that these logs do not include any sensitive information.
If information is logged, it must be verified whether this procedure reveals detailed technical information which could endanger the security of the app.
The possible release of privacy-sensitive information on the user must also be tested and excluded.

03 Prevent the app from containing log files on delivery.

04 Disable the crash log and dump storage features.
Prevent a crash log or memory dump from being made if the app crashes. Preferably, error messages should be handled by the app.

05 Prevent the app from containing test data on delivery.
Test data can be found, for example, in files with the extensions .ipa, .apk and .jar.

Before taking into production

06 Prevent the app offered by the developer or tester to the user from containing any log files or logging functions.
By linking all logging and debugging operations to the release configuration, all logging and debugging code is automatically disabled in the app when built for production. This measure prevents debug code and login instructions from ending up in the production version by accident.

07 Ensure that apps offered to a wider group of users for testing and acceptance include only those logging features that are determined to be necessary for testing and acceptance (reduce production logging to a minimum; see Measure 02).
In specific cases, it may be useful to have logging functions available for testing and acceptance purposes. This process should be laid down in a test plan.

08 Check the presence of information in log files upon delivery, prior to the public offering of the app in the app store.

09 Make sure that the supplied keyboard is known not to record any login data, financial information or other confidential information.

Statistical logging

10 Do not allow the app and any third-party program libraries to include sensitive information in the log files.
Logging of activities and events for functional purposes on behalf of the user is subject to the requirements for storage.

11 Make sure that logging is kept to a minimum and does not contain any personal data.
Statistical information on the use of the app never contains personally identifiable information.
Please note that logging seemingly innocent requests can also result in leakage of user data; for example, through the query string.

U/MA.11 Transport encryption

Encryption of data transport between the server component of the application and the client component of the application protects confidential data. Mobile devices often use open and therefore unsafe Wi-Fi networks. As a result, the communication is sensitive to man-in-the-middle attacks. Protecting the session by encryption through TLS, even when the information exchanged is not confidential, is nowadays a standard requirement for securing mobile devices.

Networks that are deemed insecure must be encrypted. Unsafe networks are networks that are not protected against unauthorised access. This category includes corporate networks in offices that are not demonstrably protected physically and electronically. As mobile devices are also used over non-secure and public networks, the apps use encryption for secure communication.

U/MA.11 Transport encryption

Criterion *(Who and what?)*	The application uses **encryption** for all **communication** over networks.
Objective *(Why?)*	Ensuring the confidentiality and integrity of data supplies and transactions.
Risk	The information exchanged falls prone to the influence (i.e. reading and changing) of an attacker.
Classification	High

Measures

Encryption

01 Encrypt the communication between the app and the server.

02 Only use protocols and cryptographic techniques designated as safe.

There are no known weaknesses of the versions used within the field that are demonstrably a threat.
Currently, encryption based on X.509 certificates (or similar) offers sufficient protection in most cases.
Guidelines for the safe deployment of TLS are discussed in the IT Security Guidelines for Transport Layer Security (TLS). [6]

03 When calling services, encrypt all information in the call.
This procedure applies to privacy-sensitive information in URLs and cookies, for example.

Communication

04 As a rule, encrypt all communication over the network.
It is assumed that communication can take place over unsecured networks. Please note that the user can always choose an unsafe network unintentionally.

U/MA.12 Certificate pinning

One of the most important security measures for apps is to secure communication with the server through encryption (U/MA.11).

Certificates for encryption are issued via a trusted Public Key Infrastructure (PKI), where a certificate is issued by a certificate authority (CA). Usually, there is a root CA and several intermediate CAs that issue the certificates. The final certificate issued contains the specific URL for which the certificate was issued.

As long as a certificate authority has not been compromised, the certificates issued by it are safe. However, practice shows that CAs can also be compromised. It is therefore necessary to apply an additional security measure: certificate pinning.

With certificate verification, the app checks whether the certificate used has been issued by a trusted certificate authority and whether the certificate is unchanged (uncompromised). With certificate pinning, the certificate is validated against one specific CA or final certificate. This measure prevents a hack of an unused CA from posing a risk. If the certificate has been issued by a non-trusted certificate authority, the app can refuse the connection.

U/MA.12 Certificate pinning

Criterion *(Who and what?)*	When setting up an encrypted connection, the app checks whether the server certificate is **trusted** and takes the necessary **measures**.
Objective *(Why?)*	Ensuring the confidentiality, integrity and, if desired, the non-repudiation of data and transactions by means of a secure key or secure certificate.
Risk	The information exchanged falls prone to the influence (i.e. reading and changing) of an attacker, even though an encrypted connection is used.
Classification	Medium

Measures

Trusted

01 Provide the app with certificate pinning for all communication over the network.
By default, certificate pinning takes place by hard coding the certificate into the app.

Measures

02 Check the certificate. Communication is stopped in the event of an incorrect certificate.

03 Provide procedures in the event that a certificate has become untrusted.

This process will limit the consequences of the certificate or key being unsafe and, consequently, of the communication being unsafe or the app being unavailable. For example, the developer will then have a second certificate from another certificate authority available and pinned. For the certificate no longer trusted, a new certificate is created and pinned which becomes available as soon as possible through an update to the app (see U/MA.19).

U/MA.13 App hardening

When the app is hardened, the app's communication capabilities are kept to a minimum (only what is strictly necessary). One of the ways to achieve this situation is by removing or deactivating unnecessary interfaces. By taking stock of the necessary interfaces and then determining the dependencies, a minimum list of interfaces that the app needs to have at its disposal can be compiled. All other interfaces can be removed. Keep in mind that inactive interfaces still present on a system can ultimately lead to a vulnerable app. It is therefore safer to keep the attack surface as small as possible. To this end, unnecessary interfaces and access rights are preferably removed.

It is possible to reduce the number of potential attacks by stripping the app of functionality that is not related to and not required (i.e. strictly necessary) for the functioning of the app. If removal is impossible, all functionality not strictly necessary must be disabled.

Most libraries used in the development of apps contain more functionality than the app needs for the purpose for which it was developed. To prevent unsafe (outdated) protocols or functions from being active, the use of interfaces is tracked. As part of U/MA.19 Up-to-date app, it is ensured that the versions of the app used are up to date.

U/MA.13 App hardening

Criterion *(Who and what?)*	The developer ensures that access to the app is only possible through **interactions** that are **required** for the application to function properly.
Objective *(Why?)*	The app and the data are protected against any additional risk of misuse via weaknesses, by limiting the functionality to what is necessary for the correct functioning of the app.
Risk	An attacker breaks in via unused inputs on the app, as a result of which the confidential data and the app fall prone to the influence of an attacker (i.e. reading and changing).
Classification	High

Measures

Interactions

01 Deactivate program code, libraries and components that are not in use and remove them where possible.

An interface of the app for communication outside the app is only available and accessible if the purpose and hence their importance to the user has been demonstrated.

02 Deactivate and, where possible, remove any unnecessary protocols or functions that can be used to access the app (e.g. by other apps).

03 Only grant authorisations if this process is necessary for the correct functioning of the app.

Required

04 Only use interactions when their purpose and hence their importance to the user has been demonstrated.

Once the app has been built, it is checked every time that the app is changed whether the modified app offers more interaction possibilities than are necessary based on the design.

05 Make the app available with an up-to-date overview of the necessary interaction possibilities and provide supporting information.

06 Make the app available with an up-to-date overview of the libraries used.

07 Provide the app with the latest relevant versions of the communication options.

08 Do not use deprecated functionality in the app.

Deprecated functionality is software that is rejected by the original developer (or on developer platforms); for example, because it contains vulnerabilities or leads to vulnerabilities in the functioning of the app.
Parts of APIs (classes or functions) may be rejected. This procedure applies to APIs of the platform, but it can also apply to third-party libraries. The developer of that API then advises not to use certain functions, usually because these functions are deprecated and involve risk. It should be taken into account that rejected functionality is removed over time.
If certain functions are rejected, new facilities are usually available to achieve the intended functionality. Where rejected code was present in an old version, it is replaced by the most current version or this function is no longer used.

09 Perform a risk analysis if deprecated functionality in the app is to be used, because older versions of the operating system can otherwise no longer be supported. Submit the results of the analysis to the client for consideration and acceptance.

Depending on the oldest version of the operating system that needs to be supported, new functionality may only be available from a specific API. Being able to run on older versions is then the only way to maintain the rejected logic. A risk assessment must be performed to determine whether to support both the old and the new way, or whether to require the API and hence a different version of the operating system in which the new functionality concerned has become available.

Elaboration

Interactions can work synchronously or asynchronously, be available in the background, or allow interaction with the user.

- A service is part of an app that runs in the background without the need for interaction with the user and that provides functionality for interaction with other applications. Services can launch other background processes and exchange data with other apps via so-called 'content providers'. In addition to services, there are also so-called 'Intents' within Android, which allow asynchronous communication.
- Compared to regular services, these services are very suitable for question-and-answer interactions in a system (whether or not distributed over multiple devices) where apps/services are not offered by a single party. The questions and answers are coordinated via nodes that coordinate the questions and answers. An example of this service in Android is a 'broadcast receiver'. On iOS, this process is referred to as a 'Notification center service'.

U/MA.14 Principle of least privilege for other apps

System abuse risks can be significantly reduced by limiting rights on the app. Common policy principles include those based on 'standard no access', 'least privilege' and 'need-to-know'. This measure applies to users but also to apps among themselves. According to the principle of 'least privilege', the rights on an app are limited to the minimum set of rights needed for it to function properly.

In the case of a remote hack of the app, attackers can do everything that the app is allowed to do. For example, if the app has unnecessary rights to use the camera, attackers can abuse this superfluous right to take photos and videos remotely. These rights should therefore be kept to a minimum in order to keep the risk profile of the app as low as possible. As a consequence, the principle of least privilege must be taken into account in the design of the application.

U/MA.14 Principle of least privilege for other apps

Criterion *(Who and what?)*	The **access rights** to functions and data on the app are only issued where they serve the demonstrated purpose and interest of the user.
Objective *(Why?)*	Another app and users will only be granted those rights on an app that are necessary for the operation of the app.
Risk	An attacker, user or compromised app may abuse rights, resulting in the operation of the app and sensitive information falling prone to the influence of unauthorised parties (i.e. reading and changing).
Classification	Medium

Measures

Access rights

01 When designing an app, one must avoid granting unnecessary permissions to other apps.
During the design stage, unnecessarily issuing rights to other apps can already be prevented by determining the necessary and thus the least necessary rights.

02 Determine on the basis of a risk analysis which access rights must be granted that are necessary for the correct functioning of the app.

03 Make the technical interface of the app accessible only by trusted apps through the use of access rights.

04 Assign explicit authorisations for the use of a function and/or access to data on the app.
The rights required by the app are tracked in an app manifest file in the permissions element. This procedure prevents an unsafe app from gaining access to information entered by the user in the user interface of the app.

U/MA.15 Input standardisation

As with all software, apps are highly dependent on a variety of inputs such as user input, data received from external servers, other apps and local files. Apps depend heavily on internet standards such as JSON, XML, SQL, HTML and JavaScript. Input may contain characters or commands that affect the operation of the app. Such input does not comply with the rules for secure input.

Just as server-side apps and/or web apps must be protected, so must mobile apps. If the app does not handle malicious input correctly, certain input can be used in order to gain access to the data that should be protected by the app.

Standardisation of content means that the content will comply with a number of restrictive rules. Through standardisation, the app can prevent rogue requests from erroneously passing through validation filters. In addition, the input is represented in such a way that it can be safely processed at all places in the app (elimination of SQL and HTML injections). Standardisation is also known as 'anti-evasion' or 'canonicalisation'.

The app receives input from the user, other apps, the back end and services on the mobile device as well as via the network. This input can take various forms, which the app must normalise.

U/MA.15 Input standardisation

Criterion *(Who and what?)*	The app prevents manipulation of all **received input** through **standardisation**, before processing it.
Objective *(Why?)*	Preventing access, alteration, loss and misuse of data by means of appropriate characters or commands in the input of the app.
Risk	Attackers can influence the operation of the app and obtain, change or add confidential data.
Classification	High

Measures

Received input

01 Make an inventory of all the app's inputs.
The developer keeps an inventory of all places where the app can expect input.

02 Apply standardisation to all inputs, from user interfaces, other apps and files on the mobile device to external network resources.
In addition to responses from the server side that have been changed by a man-in-the-middle or server hack, the user input can also originate from an attacker. After all, an attacker may have gained physical access to the mobile device. Other apps, Intents or files on the mobile device, including the SD card, or communication in public locations may also generate manipulated input. As a result, standardisation must be applied to all inputs.

03 Whitelist all trusted sources in the app.
At the inputs, it is checked whether the source is a trusted source.

Standardisation

04 Provide standardisation according to what is necessary for the type of input, among other things (but at least the following):
- converting null characters to spaces;
- normalising path references such as '/./' and '/../';
- removing superfluous spaces and line breaks;
- removing unnecessary white spaces;
- converting upper case to lower case or vice versa.

Use at least the standard services that exist for this purpose.
Convert all high-risk characters to safe versions by escaping them. Take into account differences between character sets; e.g. ASCII and UTF-8.
For example: The escape (\) preceding a character indicates that the character is to be interpreted literally, thus, \"becomes". Escaping can also be done through the hexadecimal value of a character, thus, \x22 becomes". However, this procedure will not provide the desired text if interpreted with an ASCII-incompatible character set.
Characters from the input that are processable and not undesirable can still be dangerous when used within the program logic. Risky characters may be part of legitimate input. For example, the city name 's-Hertogenbosch will result in a syntactically incorrect query. By placing an escape in front of the apostrophe, the database will consider the apostrophe part of the input string rather than part of the query. Many programming languages support standard services for escaping dangerous characters.
Similar conversions apply, for example, to HTML, XML, and so on. Perform escaping on the input after applying whitelists and, if necessary, blacklists. Escaping should focus on the program components where input is processed.

U/MA.16 Input validation

The most important rule of thumb for input in an app is that the application may not trust any input and must therefore validate all input for accuracy, completeness and validity. At a minimum, the input should be validated for values outside the valid range (limit values), invalid characters, missing or incomplete data, data that do not conform to the correct format and inconsistency of data with respect to other data within the input or in other data files. Non-trusted input can reach the app from all kinds of different attack vectors, such as Intents, services, network traffic, binding interfaces and access to files. Input validation is the most important condition for reliable data processing and invalid input must be rejected by the app.

Validating the content ensures that only valid data are processed. Validation takes place both at the protocol level (usually HTTP) and at the application level. The aim is to prevent the software from being misused at the application level or from failing due to user input.

U/MA.16 Input validation

Criterion *(Who and what?)*	The app eliminates the possibility of manipulating by subjecting all **received input** to **validation** before processing it.
Objective *(Why?)*	Preventing manipulation of the app through its input.
Risk	Attackers can influence the operation of the app and obtain, change or add confidential data.
Classification	High

Measures

Received input

01 Make an inventory of all the app's inputs.
The app receives input from the user, other apps, the back end and services on the mobile device as well as via the network. This input can take various forms. In addition to responses from the server that have been changed by a man-in-the-middle or server hack, the user input can also originate from an attacker. After all, an attacker may have gained physical access to the mobile device. Other apps, including Intents or files on the mobile device, including the SD card, and communication in public locations may also generate manipulated input.

02 At all inputs, validate the input from user interfaces, other apps and files on the mobile device, as well as from external network resources.

03 Whitelist all trusted sources in the app.
If an app does perform input validation and filtering, this filtering is often not effective enough to block all possible attacks on the app. This fact is especially true when the app uses blacklisting. In the case of blacklisting, the hazardous sources must be known in order to blacklist them. With whitelisting, not all dangerous sources need to be known, but only trusted sources are allowed.

Validation

04 Validate all input.
Validate the content of an HTTP request based on processable input (whitelist). Validate the input for rogue keywords, characters and patterns (blacklist).

05 Reject incorrect, invalid or prohibited input.

06 Do not make WebView calls until the input validation has been completed.
WebViews allow apps to access content from online sources within the app. WebViews are in fact comparable to web browsers in terms of risks, because they are vulnerable to all kinds of browser-related actions such as same-origin policy bypasses, JavaScript parser buffer overflows and cross-site scripting. In terms of information security, WebViews are susceptible to risks that cannot be solved easily. These risks mean that an attacker is able to control information loaded into the WebView. This situation can occur in many ways, such as man-in-the-middle, client-side injection or cross-site scripting.

07 Whitelist the permitted methods in, and the sources for, the WebViews.

08 If JavaScript is required, use addJavaScriptInterface() only for the web pages from which all input is reliable.
In general, only the JavaScript of your own app is reliable.

Elaboration

Uncontrolled (unvalidated) user input is a major threat to an app. If user input is used directly in HTML output, cookie values or SQL queries, for example, there is a good chance that a malicious party will compromise the app. Lack of input validation can lead to cross-site scripting, SQL and command injection vulnerabilities, as explained below.

- Cross-site scripting (XSS): In addition to the traditional XSS exploits, XSS allows attackers to start application code used by WebViews.
- SQL injection: When using a local database, SQL injection makes it possible to read, modify and delete the locally stored data.
- Command injection: The logic of the app is changed by manipulating the input for a command, making it possible to read, modify and delete the locally stored data.

There are a number of basic principles with regard to input when developing apps:

a. Other apps are untrusted, so their input is also untrusted.
b. Any input that does not meet one or more checks will be deleted or refused. Only the characters appearing in a predefined list are allowed. This approach is also referred to as the whitelist approach.

In the development process of the app, the software will have to be explicitly examined for a correct interpretation of these principles. This measure will require extensive testing or targeted code reviews.

SQL injection
When developing the apps, information can be stored locally in different ways, such as in an SQLite database. In general, such databases are only accessible by the app for which they were used, or only from a specific sandbox.

The model with a content provider is the recommended way to share data between apps. Interfaces that use SQL are sensitive to SQL injections if they are not used safely. These interfaces can consist of native code in the apps or web components (HTML5).

With SQL injection, an attacker can expand the input of allowable SQL commands with their own SQL commands. This process allows the attacker to create, modify, edit, read or delete data in the database. Such non-trusted input can reach the app from all kinds of different attack vectors, such as Intents, services, network traffic, binding interfaces and access to files. Android content providers and SQLite Query Builder offer full support for parametrised queries. Through parametrisation, the input only consists of the specific variables or parameters of the query. The query code itself is fixed in the app. By validating the variables and parameters entered, SQL injection is prevented.

U/MA.17 HTTP methods

The back-end web server supports the HTTP protocol. HTTP has methods, headers and error information that may be misused. As a consequence, its use is limited to the minimum necessary for the functioning of the accessible apps.

HTTP 1.1 and 2.0 support various functions. In practice, apps use only the GET and POST methods. In fact, for many scripts and objects, only GET is needed. The remaining functionality is hardly ever needed within traditional apps and poses an additional security risk.

U/MA.17 HTTP methods

Criterion *(Who and what?)*	The app uses only the **HTTP methods** needed to communicate with other apps and services, whether or not over the network.
Objective *(Why?)*	Preventing the use of unnecessary methods that can be used to manipulate the logic of the app.
Risk	The operation of the app is manipulated, which places it under the control of an attacker.
Classification	Medium

Measures

HTTP methods

01 Only use GET and POST HTTP methods. Substantiate and describe any necessary methods other than GET and POST and record this fact in the design documentation.

The following is laid down in the design documentation:
- *which HTTP request methods (GET, POST) are required for the supported apps;*
- *what information in the HTTP headers is important for the functioning of the system;*
- *which standard error messages are displayed or sent;*
- *how the above has been realised; for example, consider the configuration of the web server and – if applicable – the application firewall.*

Methods other than GET and POST are hardly ever needed within traditional apps and only pose an additional security risk (in the event of abuse). Any deviations from the above which are necessary for the app to function must be substantiated. This situation may be the case, for example, when using PUT and DELETE together with REST.
In a CORS preflight scenario, the use of the OPTIONS header is acceptable. It is important to prevent CSRF attacks by using explicit Origins where possible. If no explicit Origins are used, a risk analysis must determine that this process does not adversely affect security.

02 Define in the configuration documentation which HTTP methods are used.

U/MA.18 XML External Entity injection

In addition to JSON, XML is a commonly used format to read data into the app. The XML-based input contains data surrounded by codes in a specific structure: in XML, the entities (data) are shown between an opening tag and a closing tag.

The codes are used to determine the structure and meaning of the data. Due to the description of the structure and the meaning of the data, the data can be reused in a number of ways. This possibility of portability has made XML one of the most widely used technologies for data exchange. As a result, XML is widely used in apps.

Parsers are often used to read out the data from the XML input. Each parser is susceptible to attacks by means of XML External Entity injection (XXE) if the correct measures have not been taken.

U/MA.18 XML External Entity injection

Criterion *(Who and what?)*	The app must limit the possibility of manipulating by protecting all **external XML input** from **entity injection**.
Objective *(Why?)*	Preventing information leakage or DoS attacks by manipulating the external XML-based input.
Risk	An attacker reads out internal data from the app or makes it unavailable to the user.
Classification	Low

Measures

External XML input

01 Validate all input that does not come from a trusted source.

02 Reject incorrect, invalid or prohibited input.

Entity injection

03 Turn off the the entity resolver when calling the parser for external XML sources. As a result, namespace parsing and document definitions are disabled.

The entities in XML can be endlessly nested. With XXE, endless nesting can lead to a denial-of-service (DoS) of the parser. The DoS is caused by the fact that the problem is not solved locally in the parser but leads to additional network traffic as a result of constantly contacting the XML source.

U/MA.19 Up to date apps

Older versions of apps can contain vulnerabilities that are often widely known to attackers and that can be abused. Keeping the apps up to date is therefore an important condition for keeping the app safe.

U/MA.19 Up to date apps

Criterion *(Who and what?)*	The vendor applies **lifecycle management** to the apps that it delivers, allowing users always to have the **safest version** available.
Objective *(Why?)*	Restricting the use of unsafe versions.
Risk	Known vulnerabilities in old versions are abused by an attacker.
Classification	High

Measures

Lifecycle management

01 The client must make agreements with the developer on keeping the app up to date.

When doing so, the expected service life of the app is taken into account.

02 Design the app's lifecycle management: from identifying or detecting a threat to the installation and cleaning up of the app as well as related information.

The quality of the lifecycle management process is reflected in the time which elapses from the moment that a threat is detected to the release of a patch. An app that has been developed according to the applicable requirements can become unsafe as a result of new developments. A forced update mechanism can ensure that users will always have the latest version.

03 Ensure a risk-based lifecycle management process. Remember to consider the levels of confidentiality and integrity that the app must meet.

The levels of confidentiality, and the robustness with which they are achieved and maintained, determine the actual security of the apps and services in the risk management process. Unsafe services, APIs and apps are deleted (or the app key is removed), while services and apps are replaced by new ones. Remember that the app store will not necessarily set higher requirements for publishing apps that must comply with a high level of confidentiality. For such apps, it is therefore necessary to check beforehand whether this requirement necessitates additional measures. Check the functioning of the risk management process with regular audits.

04 Preferably force updates if an update is available. At a minimum, the user must receive a message when an update is available.

For example, show a warning and an update request when the app is opened.

Safest version

05 Use an up-to-date version of all libraries (third-party and otherwise), in which no vulnerabilities are known.

06 Actively affiliate yourself with platforms where threats are monitored in order to anticipate and prepare for attacks as well as to combat them.

The race between hackers on the one hand and security experts and secure programmers on the other hand consists of reactive measures taken in response to attacks or expected attacks by hackers. Although the security experts/ programmers try to anticipate attacks or prevent damage, 'reactive' usually also means 'too late' in practice. Still, in theory, secure programmers play an important role in protecting apps by programming safely.

07 Force users to use the latest version of the app itself as well as the third-party services, APIs and apps used for the app in question.

Safety is guaranteed by looking at:

- *contractual guarantees;*
- *compliance with security requirements;*
- *access (least privileges);*
- *information that is shared (the transactions through the APIs).*

Appendix A **References**

Please note that some resources may not be available in English.

[1] NCSC: IT Security Guidelines for Web Applications.
https://www.ncsc.nl/actueel/whitepapers/ict-beveiligingsrichtlijnen-voor-webapplicaties.html

[2] NCSC: Guidelines for Mobile Devices.
https://www.ncsc.nl/actueel/whitepapers/beveiligingsrichtlijnen-voor-mobiele-apparaten.html

[3] NCSC: Policy and management guidelines for the development of secure software.
https://www.ncsc.nl/actueel/whitepapers/beleids--en-beheersingsrichtlijnen-voor-de-ontwikkeling-van-veilige-software.html

[4] W. Tewarie: SIVA – Methodology for the development of audit reference frameworks.

[5] CIP: 'Grip op Secure Software Development (SSD)' ('Grasping SSD').
https://www.cip-overheid.nl/downloads/grip-op-ssd/

[6] NCSC: IT Security Guidelines for Transport Layer Security (TLS).
https://www.ncsc.nl/actueel/whitepapers/ict-beveiligingsrichtlijnen-voor-transport-layer-security-tls.html

Publication
National Cyber Security Centre (NCSC-NL)
P.O. Box 117, 2501 CC
The Hague, The Netherlands
Turfmarkt 147, 2511 DP
The Hague, The Netherlands
+31 70 751 55 55

More information
www.ncsc.nl/english
richtlijnen@ncsc.nl
@ncsc_nl

June 2018