Grip on Secure Software Development (SSD)

Security requirements for Mobile Apps

Version: 1.0

Translated by: IBM

Author	Marcel Koers	CIP

Classification	Public
Status CIP category	Commented practice
Date	February 25, 2016
File name	Grip on SSD Mobile apps Security requirements v1_0

Previous

As customers, organizations frequently find it a challenge to provide secure IT services for which the software used satisfies the security requirements. In order to have secure software, starting with the development phase, control must be given to IT projects and the implementation of changes on existing systems. Outsourcing development, maintenance and management to multiple external suppliers makes this control issue exceptionally complex. Unspoken expectations regarding information security and privacy protection come up again and again. The "Grip on Secure Software Development (SSD), Security requirements for (web) applications" (Grip on SSD) method, published by CIP explains how the customer can exercise control, make expectations explicit and monitor the execution[1]. One important aspect of control is the use of a manageable number of security requirements. This includes the security requirements for developing applications for mobile devices: the mobile apps. Undoubtedly, this could be more complete; it could certainly delver deeper, but the intention for the time being is to first get a grip on the mobile material.

The SSDm security requirements are *a supplement* to the SSD "Grip on SSD" security requirements for server applications. They focus on being able to control expectations among the parties involved, even when development, maintenance and offering the app in an app store are outsourced. The security requirements take into account the mutual expectations between the parties involved and designate the mutual responsibilities that must be exercised in order to be able to satisfy the security requirements.

The initiative for the development is taken by the SSD practitioner community and is further detailed by a workgroup consisting of the following members:

- David Vaartjes (Security)
- Willem van Deel (Tax Authority)
- Leendert Versluijs (Tax Authority)
- Jan Laan (SIG)
- Arjan Nieuwenhuis (Capgemini)
- Arnd Brugman (Sogeti)
- Eric Zuidweg (Clockwork, Ordina)
- Dennis Brocker (Ministry of Justice)
- Ewout Vochteloo (Police Service ICT)
- Taeke de Jong (Dictu)
- Henk Ameling (UWV)
- Marcel Koers (CIP)
- Roger Vikdazir (CIP, secretary)
- Wiekram Tewarie (CIP)
- Ad Kint (CIP, workgroup chairman)

During a 6-month learning and development period, knowledge was intensively exchanged, a great deal of dialogue was conducted and documents were reviewed. The result is the initial version of SSDm, which could not have been created without the help and collaboration of a large number of people. We would like to thank CIP director Ad Reuijl, who enabled us to deliver a high quality product.

Amsterdam, 26 January 2016

- CIP thanks IBM for making this translation from Dutch to English possible.

Amsterdam, 8 June 2016

[1] http://www.cip-overheid.nl/

Content

1 Introduction and Reading Guide

The following paragraphs contain important information regarding the scope and intention of this document. In no case may the title "Security Requirements for Mobile Apps" be taken to mean that this is the definitive answer regarding all threats. First of all, this is because the field and the application of mobile apps itself is in considerable flux; secondly, it is because the scope of this document is limited.

The layman may not always comprehend all the contents of this document. The norms at the criteria level (the criterion, the objective and the risks), are written in such a way that - for those who work in IT, such as information managers - they can be used as a norm for acquiring secure apps. Those who have knowledge of ICT can delve deeper into the subject matter and can thus verify that the criteria are satisfied. For this target group, criteria are further detailed in the form of so-called indicators. The explanations regarding the indicators have also been added for this target group.
This approach for arriving at a description of the security requirements for mobile apps has been chosen in order to make this broad target group aware of the aspects that play a part in getting a grip on app security.

1.1 Incentive

Mobile apps are also increasingly used for business applications. These apps can then frequently process confidential information. However, risks are involved in this that can be many times greater than the risks involved with server applications.
Applications on a server run in a better protected environment than apps on a mobile device. This is primarily because mobile devices can be used at non-secure locations and connections may be routed via non-secure networks.
This document describes the most important security requirements for organizations that apply when developing and purchasing so-called (mobile) apps; the document entitled “Security Requirements for (web) Applications” describes the SSD requirements for applications on the server side. The document entitled "Grip on SSD" describes how an organization can get a grip on app security using these documents (in the customer role). These documents offer a complete solution for creating secure software.

1.2 A few important remarks

The requirements are restricted to a system's application layer. Security requirements that are imposed on the infrastructure, the workplace or the personnel, for example, are not included. Existing norm frameworks for information security can be used for this, such as ISO 27002.

Maintenance of this document is extremely important if readers are to be able to continue to deal with the most important threats. Thus, the list of security requirements or security norms is and will be collectively kept current by customers and suppliers who develop the software.

The norms list is limited and thus kept clear and organized, making good governance possible. How governance is made possible is specified in the 'Grip on SSD' method. We do, however, want to warn you to be alert and to remain attentive to possible threats and appropriate measures.

1.3 The scope: native apps (mobile apps) on the client side

This document deals with mobile apps: applications that are designed to run on mobile devices, such as smartphones and tablets.
In terms of defining the scope, the scope is first of all limited to apps that use web standards, such as Hypertext Transfer Protocol (HTTP) or its encrypted form: HTTP Secure (HTTPS). Applications such as legacy applications, which are based on the classic client/server protocols and are generally found on classic desktops, are not part of the norms handled in this document.

Applications can be installed and run as 'apps' within the mobile device's operating system, or come as part of the web page and run within mobile browsers (such as responsive web applications[2]). We distinguish the following three types of applications:

a. Web apps: these are applications that only run in a web browser. The application logic is written in HTML, Javascript and CSS. The application logic can be located in the app itself (client) or is accessed using a URL (server). Frameworks[3] for Webapps frequently have an API so that, using Javascript, they can also use the device hardware.
b. Native apps: run within the operating system on the mobile device.
c. Hybrid apps: hybrid apps are a combination of web apps and native apps.

This document describes the security requirements *for the native apps and the native part of the hybrid app*, which is referred to as a mobile app or app for short. These are thus the apps that are or will be installed *on the device itself* (client side). All server side parts of the chain, such as REST API, authorization mechanisms and storage on the server, fall outside the scope of this document. The requirements regarding the server side are described in "Secure Software Development, Security Requirements for (Web) Applications."

Native applications are generally downloaded from a central store: the app store. It is generally possible to download apps via a direct link (URL), as well, thus outside the app store.

The so-called Internet of Things (IoT) also falls outside the scope of this document. The Internet of Things consists of (everyday) devices that are connected to the Internet, to control devices remotely or the read from sensors. These devices have their own configuration and their own standards, and thus comprise an expansion of the standards mentioned in this document. The security requirements in this document do not provide an answer to possible security risks for the individual configuration and individual standards that apply for the IoT. This means that you must develop your own security requirements for the specific security risks involved for software for the IoT.

It is indicative of mobile devices that they can be used in any environment, whether this is a secure workplace via a secured network or a non-secure environment via an unprotected network. When specifying the security measures, we assume the most risky situation(s). Assuming the worst case scenario means that use of the software is not limited exclusively to secure environments and the user is not restricted in terms of mobility.

1.4 Operating systems

As criteria, the security requirements are described independently of the operating system. This is in contrast to other norm frameworks that are specifically tailored to HOW security must be incorporated on a specific device. The explanations regarding a criterion do deal with the meaning of this within a specific device. The SSD requirements comprise a bridge between WHAT is required of an application and HOW the software supplier must construct the software. However, the latter does not prescribe to the software supplier HOW this must be done.

In the explanations to the norms, the meaning of a criterion on the following mobile platforms is discussed:

[2] If an application or web site can be designated 'responsive', it adjusts itself to accommodate the device on which it is displayed and also adjusts to accommodate the size of the screen. The advantage of this is that the application or web site works on multiple platforms in as user-friendly manner.

[3] An application framework consists of a development, and generally, a corresponding runtime environment. Another (comparable) runtime environment can also be used. The development environment consists of (standard) libraries with software components that can be deployed. Agreements regarding what code standards and libraries are used and how the components are used are part of a framework. The runtime environment offers applications an environment, within which the application can run. Because all services are offered within this environment, it can be used and no longer has to be developed.

a. iOS: iOS is the operating system for Apple's iPod, iPad and iPhone. Native apps are generally written in Objective C or in Swift; however, C++ is also used a lot.
b. Android: the open source Google operating system. Native Android apps are generally written in Java and C#. This is used by many manufacturers. These manufacturers frequently create modified versions of Android to distinguish themselves. This has led to a long list of more or less distinguishable versions and derivatives; one striking aspect of this is that improvements that are introduced in the latest version are not incorporated into the original and previous versions. Combined with the fact that the devices themselves are frequently bound to a specific version and cannot always 'upgrade' to the newest edition, this represents a serious (additional) security risk for devices for which updates are no longer issued.
c. Windows for smartphone and tablets: Windows is the Microsoft operating system. Until Windows 10 was introduced, Microsoft maintained different operating systems for the different types of devices (such as tablet and mobile), such as Windows RT and Windows Phone. Microsoft does invest in cross-platform. Windows 10's goal is to make simple applications available for the types of hardware on which a Windows platform runs (PC, Tablet/Smartphone, Xbox, IoT devices, etc.). Native Windows apps are developed in C# based on the .NET Framework.

1.5 Threats

Apps face different forms of threats:

1. **Remote:**
   Attacks that originate from the network. These attacks focus on entry points to an app that can be accessed via the network. Think, for example, of webview, JSON client, XML client, and attacks from MitM or a compromised server.
2. **Local:**
   Attacks that originate from your own device, because someone has acquired electronic access to the device. Think, for example of malware that is installed on the device, or a piece of malicious code with access to the operating system (jailbreak).
3. **Physical:**
   Access in the event of loss or theft of the device, whether or not it is locked or unlocked. Locked or unlocked, the device can fall into the hands of the wrong person. The fact that the device is locked does not always guarantee (continued) protection.

1.6 Risk analyses

All modern norm frameworks for information security (maintained by the government), such as BIR and ISO 27000, propose a risk-based approach. This means that a decision regarding whether to take a measure or not must be based on a weighing of the risks.
Thus, no detailed risk classification is used in the security requirements. In terms of a decision to apply a measure or not, classification is only indicative to a limited degree. By definition, a mobile device is a non-secure device, or in any event a device that can only be secured to a limited extent. 'No', 'limited' or 'moderate' risk categories are simply not applicable and tell you nothing about the nature of the threat. Every decision to apply an SSD requirement must be based on a conscious weighing of the risks involved. Mobile apps must *always* be developed with the hostile environment and the ignorant user in mind.

The choice regarding whether or not to apply an SSD security requirement or to apply such a requirement to limited extent is determined based on a risk analysis.

1.7 Comply or Explain

In terms of the security requirements imposed, the 'impose or explain why not' principle applies. A measure that applies for a security requirement is *not applicable*, if it can be demonstrated that:

- the costs of the solution are not proportionate to the analyzed (determined) risk;

- other measures are or will be implemented that mitigate the risk.

It is important that the measures taken and the risks that are accepted are clear and fit in with the customer's "risk appetite". This requisite transparency and a field that is constantly in flux require that the relationship between risks and measures is monitored in a governance process.

The security requirements described in this document are a best practice; they specify how the measure can be implemented. Depending on the situation, possible alternative measures may be more appropriate. The proposed measures are thus not hard and fast requirements in themselves. However, the risks specified with the requirements must be covered using measures and/or acceptance of the risk by the customer.

1.8 Explanation regarding this version

The security requirements are based specifically on Android and iOS. However, the requirements are described in such a way that they can also be used for Windows. When aggregating the requirements, platform-independent terms and descriptions are used to the extent possible.

1.9 References

[1] Reference architecture for mobile applications, DICTU, version 1.1; November 11, 2015
[2] Update #22 and #23 of Madison & Gurkha; Mobile applications: Risks, vulnerabilities and attack sectors
[3] OWASP: Application Security Verification Standard (ASVS); https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project
[4] SSD requirements; version 2.0; CIP; http://www.cip-government.nl/
[5] Whitepaper 'iOS Application (In)Security'; May 2012; Dominic Chell; MDSec Consulting Ltd; www.support.office.com
[6] Grip on Privacy; Privacy Baseline; 23 November 2015; CIP; http://www.cip-government.nl/grip-on-privacy/

2 Explanation of the design of the security requirements

2.1 Template used

The SSD security requirements are based on the SIVA method [Tewarie, 2014]. How this is used is described in the appendix. To provide a *structured* answer to the question "Who does what and why?" the method prescribes a template:

SSDm-no. Subject of the norm					
Criterion (who and what)	What (xxxxxx) <verb>xxxxxkey wordsxxxxx				
Objective (why)	The reason the norm is used.				
Risk	The risk that is the incentive for maintaining the norm.				
Reference	Source 1	Source 2	…		

Each keyword constitutes an indicator, which must be satisfied. That is why the keyword is detailed. The template used for keywords is:

SSDm-no. Subject of the norm	
	Indicators
/01	keyword
/01.01	indicator 1.1
/01.01	indicator 1.2
…	…

The keywords (/01, /02, etc.) and the perspectives are numbered (/01.01, /01.02, etc.), so that they can be referred to in the subsequent explanation.

To the extent relevant, for references we specify where additional information can be found in the following standards and guidelines:
ISO27002: NEN-ISO/IEC 27002 *Code for information security*, 2005. For the relevant references, you can also use the 'Baseline Information Security for Civil Government Service' (BIR).
SSD: CIP *Grip on Secure Software Development (SSD), Security requirements for (web) applications* version: 2.0;
ASVS: OWASP, Application Security Verification Standard (2014)

2.2 The parties involved

For the security requirements, the following roles are described for the parties involved:

- **The customer for an app:**
  This party is responsible for the functional management of the application. From this role, the customer initially specifies the app and defines the security requirements. Remember that in the absence of a customer, the software fulfills this role.

- **The internal or external software supplier:**
  This party handles the design, the development and the testing and ensures that the application can function on the intended device. No distinction is made here between application management and

technical management[4].
The primary responsibility for implementing the requirements described lies with the internal or external software supplier. A software supplier is regarded as an internal software supplier, if both he and the customer belong to one organization.
As part of the delivery of a release that is ready for production, the supplier reports to the customer regarding how the imposed requirements are satisfied.

- **The (de)central app store:**
  The app is made available from an app store. An app store can be an app store that is affiliated with an operating system, for example for Android: Google Play, for iOS: iTunes and for Windows: the Windows Store. However, applications can also be offered from an enterprise app store or simply via a link.

  It is important that the store monitors the interests of both the user and the customer. Remember that each app store and platform can maintain its own acceptance criteria. Generally speaking, the acceptance policies that Apple and Microsoft maintain are stricter than those maintained by the Android app store. It is also easier to install apps from other app stores on an Android device – including apps that are not trusted by the operating system.

  Before publishing an app, the following items must be verified at a minimum:

  - The app works in conformance with the description;
  - The developer is a trusted developer;
  - The app (to the extent that this can be verified) does not contain any bugs or cause crashes;
  - The app does not contain any undesirable functionality or any functionality that is not described (malware).

  In practice, any checks the app store runs will be minimal. The fact that the app has been published is no guarantee that it is secure. Thus, it is important to determine whether or not the app satisfies the minimum security requirements, as these are described in this document. See the principles described in the paragraph entitled Comply or Explain (1.7) for more information.

- **The user:**
  The user is responsible for the management of the device. Because in practice the user generally has only limited knowledge regarding keeping the device secure, the device is regarded as inherently unsafe. The app itself or an extra provision such as Enterprise Mobile Management (EMM), Mobile Application/Access Management or Mobile Device Management (MDM) must ensure the app's security. If you are certain that the app runs in the MDM[5] environment, this environment is viewed as an extension of the app. It is important that the responsibility for satisfying the requirements that apply for the app is never left up to the user.

[4] The reason for this is that the user is regarded as a party who does not have the knowledge to perform technical management activities.

[5] For the remainder of this document, this will be referred to as MDM. However, in place of MDM, you can also read EMM.

3 Security requirements for the mobile apps

3.1 SSDm-1: Secure server side applications

Together with the application on the server side, the app on the mobile device comprises a chain. Secure use of the app is only possible in combination with a securely configured server side application.

Lack of a secure server side application or an unsafe server makes it impossible to create a secure chain. This results in an environment that is not appropriate for saving and exchanging confidential information.

SSDm-1: Secure server side applications					
Criterion (what)	When constructing an app, the developer must respect the security requirements that apply to server.				
Objective (why)	When developing the app, only those configuration choices that do not jeopardize the security of the server (application) are made. This guarantees the (end-to-end) security of the chain from the app all the way through the server side application.				
Risk	Building an app makes secure configuration of the server side application impossible or more difficult (more expensive).				
Reference	ISO27002	SSD	ASVS		
		All SSD requirements			

Conformity indicators

/01 Security requirements

Apps on the mobile device and the applications on the server collectively comprise a chain. The SSDm requirements describe the requirements for apps on the mobile device. The SSD requirements describe the requirements for server applications.

SSDm-1: Secure server side applications	
	Indicators
/01	Security requirements
/01.01	The app does not impose any requirements on the server side applications that conflict with the SSD security requirements for (web) applications.
/01.02	The customer ensures that the server side applications used by the app satisfy the SSD requirements.

Explanation

/01. Paying attention to the information security of the server application prevents requirements from being imposed on the server application during construction of the app that conflict with the information security of the total chain from mobile device to server.

/01.02 Maintaining the SSD requirements for the applications on the server is one requirement for ensuring a secure chain.

3.2 SSDm-2: Secure operating system

Mobile devices run on operating systems that can be manipulated. Jailbreaks or taking advantage of weaknesses in unsafe (obsolete) versions, makes it possible to read or modify files protected by the operating system. This makes it possible to view, modify or change confidential information, to install malicious code, to install software or to expand existing software to include malicious features.

When constructing an app, as the developer weights the risks and selects security measures, he must assume the security (measures) set by the user and the operating system. If these security measures are not taken, the operating system does not satisfy the security requirements.

SSDm-2: Secure operating system					
Criterion (who and what)	The app checks to determine whether or not the operating system satisfies the security requirements on which the security of the app is based and sends a message to the user.				
Objective (why)	Prevent the operating system from constituting an unsafe environment for the app against which the app has not taken or cannot take any measures to protect itself.				
Risk	Unsafe operating system settings can be abused or exploited by an attacker.				
Reference	ISO27002	SSD	ASVS		
			V17.7		

Explanation

In iOS jargon, the ability to alter the operating system is called jailbreaking; for Android devices this is called rooting and for Windows Phone it is called unlocking. Android is easier to change than iOS and Windows Phone because Android is an open platform.

Conformity indicators

/01 Security requirements

SSDm-2: Secure operating system	
	Indicators
/01	Security requirements
/01.01	The developer of an app or parts of an app is familiar with the (standard) requirements the operating system must satisfy.
/01.02	The security requirements the operating system must satisfy are monitored and forced by MDM, if this is used. Control by MDM supplements the app's own control.
/01.03	If the security is based (in part) on protection by MDM, the app must force the use of MDM.

Explanation

/01.01 The requirements imposed on the operating system are described in documentation for the system version(s) and the settings for this version (or these versions).

/01.01 There are a wide range of mobile devices, each with its own specific (setting of the) applicable operating system and security model.

/01.01 There is also the problem that the supplier frequently only keeps the device-specific version of the operating system up-to-date during the first year (or two) after it appears on the market. This is generally more of a problem for Android devices than it is for the other commonly used platforms.

/01.01 For example, each user can maintain his own security settings. In contrast to iOS, for Windows Phone and Android it is generally the device manufacturer who is responsible for keeping the operating system up-to-date. These devices frequently lag behind in terms of implementing (security) updates.

/01.02 Forcing the use of VPNs, patch management, web filtering, Web Application Firewall (WAF) and whitelisting increases the mobile device's information security. A Mobile Device Management (MDM) system makes it possible to monitor and enforce these measures.

/02 Message

SSDm-2: Secure operating system

	Indicators
/02	Notification
/02.01	The user is notified that the operating system does not satisfy the security requirements before he/she uses the app.
/02.02	The notification alerts the user of the risks and the requisite measures to be taken.
/02.03	The notices are restricted to alerts of those risks that the app has proven to be unable to prevent, for which the risk involved is so great that the user must be alerted to this situation.

Explanation

/02. Detecting and/or responding to the weaknesses of the operating system and notifying the user of this can prevent damage for the users because the user is alerted to the risks.

/02. Remember that there are many ways to detect or to penetrate the operating system safeguards and that, in theory, any detection method can be circumvented.

/02.02 The risks are described in a manner that is clear to the user, so that the user can make an informed decision regarding whether or not to take measures to prevent them. The notice is attuned to the user group.

/02.02 The usefulness of the notification can be demonstrated in a User Acceptance test or based on good practices.

/02.03 Risks can only be prevented by using libraries that cover the risks for that version of the operating system. Knowledge regarding whether or not libraries are suitable is exchanged on platforms where developers meet one another.
If the developers on these platforms are not aware of any library that covers the risk for that version of the operating system, it is appropriate to send the user a notice that the operating system he/she is using does not satisfy the requirements.
(For keeping the app up-to-date, see: SSDm-3: Up-to-date apps).

/02.03 Based on a risk analysis, in collaboration with the customer the developer decides whether or not other measures must be taken to mitigate the risks.

3.3 SSDm-3: Up-to-date apps

Older version of apps can contain weaknesses, which can be abused and/or exploited. Attackers are generally familiar with these weaknesses. Keeping apps up-to-date is one of the important conditions for keeping them secure.

SSDm-3: Up-to-date apps					
Criterion (who and what)	The supplier conducts life cycle management on the apps that he delivers, so that users (can) always have the most secure app version.				
Objective (why)	Prevent unsafe versions from being used.				
Risk	Familiar weaknesses in old versions are abused and/or exploited by an attacker.				
Reference	ISO27002	SSD	ASVS		
			V17.17		

Conformity indicators

/01 Life cycle management

SSDm-3: Up-to-date apps	
	indicators
/01	Life cycle management

SSDm-3: Up-to-date apps	
	indicators
/01.01	The customer has made agreements with the developer regarding keeping the app up-to-date.
/01.02	Life cycle management is set up for each app - from ascertaining or detecting a threat to installing and cleaning up the app and the corresponding information.
/01.03	The life cycle management process is risk-based. The levels of confidentiality with which the app must comply are taken into account.
/01.04	Preferably, an update is forced for the app as soon as it becomes available. At a minimum, the user receives a notice stating that the update is available.

Explanation

/01.01 The (anticipated) life expectancy of the app is taken into account. Grip on Security describes what agreements are involved in the chapter on purchasing contracts.[6]

/01.02 The quality of the life cycle management process is expressed in the time that elapses from the moment a threat is detected until the moment a patch is issued.

/01.02 An app that was developed in conformance with the applicable requirements can become non-secure as the result of new developments. A mechanism for forced updates can ensure that users are working with the most current version.

/01.03 In the risk management process, the levels of confidentiality and robustness with which apps are created and maintained are leading for determining the current security level of the apps and services. Unsafe services, APIs and apps are deleted (or the app key is deleted) and services and apps are replaced by new versions or apps.

/01.03 You must realize that the app store does not necessarily impose more stringent requirements on publishing apps that must satisfy stringent requirements regarding confidentiality. Thus, for such apps, you must determine in advance whether the possible consequences of this constitute acceptable risks or whether extra measures are required ("risk translated into measures").

/01.03 The functioning of the risk management process is periodically verified using audits.

/01.04 For example, when starting an app, display a warning and an update request.

/02 Most secure app version

SSDm-3: Up-to-date apps	
	Indicators
/02	Most secure version
/02.01	An up-to-date version of all (third party) libraries is used, for which no vulnerabilities are known.
/02.02	By actively participating in platforms where threats are monitored, the software supplier anticipates attacks and is prepared and able to combat them.
/02.03	The most current version is forced for both the app itself, as well as for third party services, - APIs and - apps that are used for the app in question.

Explanation

/02.02 The race between hackers and secure programmers or those who safeguard programs consists of reactively taking measures based on attacks or anticipated attacks by hackers. The

[6] http://www.cip-overheid.nl/downloads/grip-op-ssd/

safeguarding parties/programmers are supposed to stay ahead of attacks or to prevent damage, but in practice they are generally 'reactive', not to mention 'too late'. Still, in theory secure programmers play an important role in defending programs by programming them to be 'watertight' and 'free of leaks'.

/02.03 When securing safety, the following issues must be considered:

- contractual guarantees,
- conformance with the security requirements (such as the SSD security requirements),
- the access (least privileges) and
- information that is shared (the transactions via the APIs).

3.4 SSDm-4: Third party apps

Apps frequently work together with other apps, such as viewers and keyboards. These apps generally originate from other suppliers and are designated as third party apps. By virtue of their function, these third party apps process information from the app. These third party apps can have hidden functionalities, with which access to confidential information can be obtained, even if the app shields this information.
When choosing such apps, the advantages and risks must be weighed against one another. For example, one advantage can be that replicating a third party app can be expensive and does not necessarily lead to a safer app because in many situations building a secure app, such as cryptographic tools, is no small feat.

SSDm-4: Third party apps					
Criterion (who and what)	The use of third party apps is based on a weighing of the risks.				
Objective (why)	Preventing third party apps form gaining undesirable access to information that must be shielded by the app.				
Risk	Confidential information is made accessible to attackers via a third party app.				
Reference	ISO27002	SSD	ASVS		
			V17.17		

Explanation

Third party apps can request permissions or contain hidden functions with which access to confidential information is obtained, even if this information is shielded. Third party apps can also exploit unsafe APIs and vulnerabilities in apps and the operating system. The point of departure is that, in principle, third party apps must be considered unsafe unless you can inspect the creation process and the code.

Conformity indicators

/01 Weighing the risks

SSDm-4: Third party apps	
	Indicators
/01	Weighing the risks
/01.01	The developer checks the third party apps for vulnerabilities, such as hidden functionality and unsafe APIs. The risks are specified by the developer.
/01.02	The choice for third party apps is based on a risk analysis. The results of the risk analysis are recorded.
/01.03	When choosing third party apps, special attention must be paid to third party keyboards.

Explanation

/01 A choice for third party apps is based on the choice between secure operation of the app and the advantages offered by using existing third party apps.

/01.01 When determining the risk, the following points must be kept in mind:

- Can (the code of) the third party apps be tested when the app is delivered?
- Can patches to or new versions of (the code of) the third party apps also be tested right after these are put into production?
- Are agreements regarding keeping the app up-to-date contractually recorded and is there an adequate governance process for checking this?
- Does the third party app satisfy the security requirements, such as the SSDm requirements?

/01.02 When conducting a risk analysis, the results of the vulnerabilities test (/01.01) are used.

/01.02 When weighing the risks, also consider the benefits of third party apps, such as the security provisions built into the third party app, when these have been proven effective in practice.

/01.02 The information security of third party apps is generally greater on IOS than it is for Android. iOS checks its apps better. IOS imposes more restrictions on third party apps than Android. However, you should remember that even iOS permits information leaks if the user grants permission for this. With iOS, the use of 'privacy best practices' and the corresponding iOS program requirements, guidelines and license agreements is encouraged, but no guarantees are made.

/01.02 See also: http://www.windowsecurity.com/articles-tutorials/misc_network_security/third-party-software-security-threat-part1.html

/01.03 On the mobile device, in addition to the standard keyboard associated with the operating system or provided by the supplier, one can frequently use third party alternatives (third party keyboards), as well. These third party keyboards can record keystrokes and thus leak or exploit data.

/01.03 Users can frequently choose alternative third party keyboards themselves. If use of third party keyboards cannot be prevented, then restrict the entry of extremely confidential information so that it can only be entered within the app's user interface.

3.5 SSDm-5: Secure code upon delivery

When developing apps, the development can be accelerated by using (external) code (libraries). However, these can contain weaknesses, viruses or malware. Information regarding the libraries used and the operation of the app provides the attacker information regarding any weaknesses in the app.

SSDm-5: Secure code upon delivery					
Criterion (who and what)	The developer only uses trusted source code libraries from third parties.				
Objective (why)	The source code used is safe and does not provide any information regarding the app's internal operation.				
Risk	Third party source code contains weaknesses, viruses or malware or information that gives an attacker access to the operation of the app or to confidential data.				
Reference	ISO27002	SSD	ASVS		
		SSD-3	V17.11 V17.16 V17.25		

Conformity indicators

/01 Trusted source code

When using existing code or code developed elsewhere, you have assurance regarding the security of the code.

SSDm-5: Secure code upon delivery	
	Indicators
/01	Trusted source code
/01.01	The app only uses code whose origin is known and whose security has been specified.
/01.02	The code and code libraries used are up-to-date and do not contain any known vulnerabilities.
/01.03	No (download of) mobile code, whose origin and security is not specified, is required in order for the user to use the app.
/01.04	The settings in the configuration file of the app being offered are such that they guarantee optimal security.
/01.05	The software supplier publishes the hash of the downloaded binary app.
/01.06	The software supplier signs the binary with the hash, so that the origin of the app is clear.

Explanation

/01.01 The source is known and no weaknesses are known within the discipline that constitute a threat.

/01.01 Do not trust closed source components from third parties, unless you know exactly what the components do. An analysis must be conducted of open source components, which indicates that the software can be considered secure.

/01.03 For software that is downloaded when building an application, requirement /01.01 regarding knowing the origin also applies.

/01.03 The common method for guaranteeing this is 'code signing'. This is a security function on the device that prevents attempts to run unauthorized applications on the device by validating the app's signature. The validation is performed every time the app is invoked. This means that apps can only execute code that has a valid, trusted signature.

/01.04 To do this, the settings for debugging and permissions as well as the security of the configuration settings are all checked.

/01.05 A 'hash value' is calculated based on the binary. Changing the binary almost always results in a different hash value (depending on the quality of the hash algorithm). This way, it is possible to check whether the binary has been modified (undetected).

/01.06 The hash and the binary can also be secured by signing them. The binary and hash are equipped with a digital signature that is unique for that hash using a cryptographic key.

/01.06 Not every user checks the digital signature and thus the authenticity of the app. Thus, it is useful to periodically search for 'look alike' or copied and modified apps in the public/private app stores.

/02 Technical information

SSDm-5: Secure code upon delivery	
	Indicators
/02	Technical information
/02.01	No sensitive technical information is stored in the app. If storing such information is deemed necessary, this is done based on a weighing of the risks.
/02.02	Security settings/measures are not stored in files that lie outside the scope of the app signature and outside the protection of the operating system.
/02.03	Information regarding the operation of the app and relevant confidential information that must be protected is shielded in the binary to prevent reverse engineering and manipulation.

Explanation

/02.01 Think, for example, of: cryptographic keys, passwords, internal URLs, etc.

/02.01 Because a binary can never be completely shielded, security measures and security settings may never be stored in the code itself.

/02.03 Core concepts in Apple's iOS Frameworks are KeyValue Coding (KVC) and KeyValue Observing (KVO). In practice, obfuscation[7] is expanded to include scrambling by adding extra code to the binary to divert the attacker from the relevant code and information.

/02.03 As a rule: obfuscation and scrambling cannot be used as a replacement in situations where encryption is required.

/02.03 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 key (and the algorithm), so that the app store can inspect the app and, after approval, can publish it. In the Apple certificate, Apple also specifies the supplier's identity. This has given Apple the possibility to restrict the offer of apps to iOS program-enabled developers.

/02.03 For Android, tools are available that can obfuscate code, such as Proguard (see: http://developer.android.com/tools/help/proguard.html) and Dexguard (see: https://www.guardsquare.com/dexguard).

/02.03 Always remember that in a jailbroken mobile device, the binary is stored in the memory in unencrypted form.

[7] Obscuration, obfuscation

3.6 SSDm-6: Reliable operation of the app

In contrast to server side applications, apps do not run in a trusted environment, but on the mobile device itself. This allows an attacker to gain control of the complete code: the binary and the operating app. In so doing, the attacker can gain control over every bit of code during every phase of the app's program plus the information that is stored in the app's binary files, including the app's configuration files. Thus, manipulation of the operation by attackers can only be prevented by shielding the binary and the operating app electronically.

SSDm-6: Reliable operation of the app					
Criterion (who and what)	The app on the mobile device is shielded against unwanted or unintended manipulation; the outcomes of the critical business logic are always checked on the server.				
Objective (why)	Prevent the operation of the app from being influenced unintentionally.				
Risk	The confidentiality of the information, the correct operation of the app and the integrity of the output fall under the influence of an attacker.				
Reference	ISO27002	SSD	ASVS		
		SSD-3			

Conformity indicators

/01 Manipulation

SSDm-6: Reliable operation of the app	
	Indicators
/01	Manipulation
/01.01	Manipulation of the app's operation is prevented by using Position-independent Code (PIC) and Address Space Layout Randomization (ASLR).
/01.02	The decision not to include ASLR in the code or parts of the code is based on a weighing of the risks.

Explanation

/01.01 Position-independent Code (PIC) is a security function, by which parts of the code can run at any place in memory. Apps that have PIC enabled are configured as a Position-independent Executable (PIE).
Address Space Layout Randomization (ASLR) is a security function that stores code randomly in the process memory when PIE is called.
Apps in which ASLR is enabled are secured against attacks based on buffer overflows. With attacks based on buffer overflows, overflow code is included in the buffer that leads to a malicious operation in the app when the code is invoked in that part of the memory.

/01.01 /01.01 Android implemented PIC as of version 4.1; iOS implemented it as of version 4.0 (see https://and.wikipedia.org/wiki/Address_space_layout_randomization#Android) and https://developer.apple.com/library/ios/qa/qa1788/_index.html, respectively).

/01.01 In iOS, ASLR was introduced for the first time in version 4.3; in Android it was first introduced as of version 4.0.

/02 Critical business logic

SSDm-6: Reliable operation of the app	
	Indicators
/02	Critical business logic
/02.01	Decisions based on critical business logic in the app are checked in a secure environment on the server side.

Explanation

/02.01 Even though protecting minimizes the chance that the business logic will be altered, there is no guarantee that the logic in the app will remain intact in the event of a hacker attack.

3.7 SSDm-7: Location for the storage

To a large extent, the choice of location for the data storage determines the possibilities for protecting the data against unwanted access. Thus, the choice regarding storage of each bit of data or set of data must always be made consciously. Because safer locations are easier to protect and offer more certainty, the point of departure for this security requirement is that the most secure location is chosen for the storage, unless there is certainty that a less secure location can be suitably secured and this certainty has been demonstrated.

SSDm-7: Location for the storage					
Criterion (who and what)	The choice of that location at which the data and information of the app logic is stored is based on the principle of confidentiality.				
Objective (why)	Confidential information and critical logic are only stored on sites where efficient security measures can be taken.				
Risk	Confidential information or critical logic falls into unauthorized hands.				
Reference	ISO27002	SSD	ASVS		
			V17.3 V17.4		

Conformity indicators

/01 Location

SSDm-7: Location for the storage	
	Indicators
/01	Location
/01.01	The standard is to store confidential data and sensitive information regarding the app's logic on the server side in a secure environment.
/01.02	When designing the app, the point of departure is that local storage (on the mobile device) must be prevented as much as possible. A consideration is made for each bit of data or set of data to determine whether offline local availability is required or can be avoided.
/01.03	If either sensitive information regarding the mobile device or sensitive logic is stored on the mobile device, this must be absolutely necessary for the operation of the app.
/01.04	If confidential information is stored on the mobile device, *a risk analysis* is conducted to determine what protection is required per bit or set of data.
/01.05	Unless the requisite data protection (/01.04) stipulates otherwise, the storage of confidential data on the device takes place *in the app's internal storage*. The fact that the (standard) folders can be part of backups or synchronization with the cloud or a linked computer is taken into account.
/01.06	If data on the device is stored outside the app's internal storage, such data is restricted to non-confidential information and/or to information the user knows he must protect or knows that he must manage and thus must protect.
/01.07	If storage on the server (/01.01) is impossible, the offline storage of confidential information takes place in the public cloud, but only after previous weighing of the advantages and disadvantages for the customer and user, including the privacy aspects, have been specified and such consideration has led to this decision. This applies even more strongly for backups.

Explanation

/01. It is more difficult to secure data that is stored on the mobile device - using the features offered by the device - from anyone who has access (physically or via malware) to the device than it is to secure a place that is protected both electronically and physically.

/01.01 Storage of all data and logic takes place on the backend if a risk analysis determines that the consequences of public storage are unacceptable.

/01.04 The risk analysis also includes the operation of components used by the app and thus the data that is stored (imperceptibly to the user) behind the screens (caching, logging. etc.).
/01.04 The risk analysis can then lead to the encrypted saving of information, for example.

/01.05 Using a sandbox shields the app and the information in the app. A sandbox is a container on the mobile device that is shielded using encryption. Even if an attacker has access to the device, he cannot access the data as long as the key with which the sandbox is encrypted is shielded.
/01.05 The 'internal storage' on the mobile device is the storage within the folder in which the app is installed. This folder is secured by the sandbox, along with the app. Other apps have no access to these files, unless the device is rooted.
/01.05 Files can be stored in 'internal storage' on both iOS and Android devices. Files are shared from the platform via interfaces.
/01.05 By default, specific folders within internal storage are included in backups or synchronizations with the cloud or a trusted computer. Consult the platform documentation to choose the most appropriate folder within internal storage based on functional considerations and a weighing of the risks.
/01.05 Most apps on Android devices save files in SD memory. This is 'external storage' and is generally located on a removable memory card. All apps can access this memory card. Moreover, by default this storage is not secured, which means that if the device is lost or stolen, unauthorized parties can simply read the SD card. On Android devices, confidential information should exclusively be stored in internal storage.
/01.05 On Android, by default files are stored with the MODE_PRIVATE flag. The files are then only accessible to apps that write information and apps with the same user ID. On Android, files for each app have a separate user ID. The app runs under that user ID. By default, this user has no access to data belonging to other apps, provided such data is stored in internal storage. All apps can access the external storage.
/01.05 If the key to a sandbox safe is stored on a backend (see /01.01), this is also shielded if the device is jailbroken or rooted. This offers the possibility for better security.

/01.07 By default, storage on the mobile device and in the public cloud, such as social media, but also other storage at third parties is regarded as insecure for confidential information.
/01.07 See your personal organization or the CIP[8] cloud policy when choosing clouds.

/02 Confidentiality

Suitable data shielding can be offered when the sensitivity or confidentiality of the data is specified.

SSDm-7: Location for the storage	
	Indicators
/02	Confidentiality
/02.01	The customer specifies (the classification of) the confidentiality of the data in or by the app.
/02.02	If the confidentiality level is not specified, the data is regarded as confidential (by default) and is secured by the developer as confidential data.

Explanation

[8] http://www.cip-overheid.nl/

/02.01 The specification gives clarity regarding what data must be shielded.
/02.01 If classifications are used, the security requirement is specified per classification.
/02.01 The analysis also includes the data backups.
/02.01 In practice, you will not know the classification for each bit of data, certainly not if this particular bit of data is used for the app's technical operation.

/02.02 The analysis also includes the business logic in the software.
/02.02 Business logic that is included in the software applies as confidential business logic if a risk analysis determines that changes to this data can be detrimental.

3.8 SSDm-8: Storage on the mobile device

Sensitive (confidential) data can be protected using cryptographic techniques. If the information is physically accessible, cryptographic techniques are, in fact, the only efficient method for shielding information.

SSDm-8: Storage on the mobile device					
Criterion (who and what)	When saving confidential information on the mobile device, confidential data is shielded using cryptographic techniques.				
Objective (why)	Prevent unauthorized use, viewing, change and loss of confidential data in non-secure storage.				
Risk	Confidential data on the mobile device falls into unauthorized hands.				
Reference	ISO27002	SSD	ASVS		
		SSD-2	V17.5 V17.21 V17.24 V17.27		

Explanation

The organization specifies what data is sensitive or confidential. Confidentiality is specified as part of the task of determining the storage location (see: SSDm-7: Location for the storage).

Conformity indicators

/01 Cryptographic techniques

Using cryptographic techniques, data is protected against unauthorized viewing and/or manipulation.

SSDm-8: Storage on the mobile device	
	Indicators
/01	Cryptographic techniques
/01.01	If confidential information is stored, password protection is required *at a minimum*.
/01.02	A cryptographic key is not stored on the mobile device, unless a risk analysis specifies that this does not lead to undesired risks.
/01.03	If no connection with the network is available (making storage on a secure server impossible), you can use a pass phrase or delaying key stretching algorithms.
/01.04	Supplemental measures are taken for data that is on the mobile device temporarily.
/01.05	For temporary files, generate a temporary key and store the key in (internal) memory until the app closes.
/01.06	Confidential information may only be stored in an SQLite database if it is encrypted.
/01.07	Familiar, proven algorithms and implementation of these algorithms are used for encryption. No self-developed crypto functions are used.

Explanation

/01.01 Protection based on screen protection using a PIN code is rather easy to penetrate using "brute force".

/01.01 For very confidential information, such as medical data and ID data, supplemental measures are required.

/01.01 Thus, secret keys and ID data are *not* stored in the code.

/01.01 The app and the information can be shielded with a key that consists of a combination of a pin code and unique features of the device. The actual key is encrypted using this derived key. (https://www.ietf.org/rfc/rfc2898.txt)

/01.01 For more information see:
http://nelenkov.blogspot.nl/2014/10/revisiting-android-disk-encryption.html
http://nelenkov.blogspot.nl/2012/04/using-password-based-encryption-on.html
http://android-developers.blogspot.nl/2013/02/using-cryptography-to-store-credentials.html

/01.02 To protect stored data using encryption, a secret encryption key is stored at a secure location.
/01/02 If the encryption key is stored on the mobile device, its protection can be easily breached. Thus, the key must be stored at another location, for example on another one of the user's devices or on a server. For example, the key could be saved in the user's profile. Access to this profile must then be blocked; in the event of access via the Internet, two-factor authentication must be used at a minimum, *where the property authentication aspect is something other than the mobile device!*
/01.02 On iOS, the so-called "Keychain" can be used as a secure storage place for user data. Data that saves an app in the Keychain can only be accessed by that specific app. For Android, the Keystore (Keychain equivalent) is intended exclusively for secure storage of the key and certificate material. The app itself must ensure secure storage for user data.
/01.02 With iOS, if the key is stored on the mobile device, it should preferably be stored in the Keychain. Remember that the Keychain is open if the user does not use a PIN code or password/passphrase for screen protection.
/01.02 If the operating system is rooted or jailbroken, the security of the stored data is no longer assured and the security can be bypassed, including the security of sandboxes.

/01.03 The time required to bypass the security using brute force can be increased by using a passphrase or a delaying key stretching algorithm, such as PBKDF2 or bcrypt. The passphrase is the process of lengthening the password to be entered to a sentence or part of a sentence.

/01.05 For iOS this means that the key is not stored in the Keychain.
/01.05 Use random input to generate the key.

/01.06 The SQLite database is frequently used without encryption. However, it is possible to encrypt the entire database. For an example, see https://www.zetetic.net/sqlcipher/.

/01.07 Robust implementation of cryptographic functions is very complex. Generally it is wiser to utilize the cryptographic functions of the operating system, or of familiar and proven third-party crypto-libraries.

3.9 SSDm-9: Unnecessary information in the cache memory

Temporarily saving confidential information in the memory of the mobile device is unavoidable for most apps. However, there are attacks which create a memory dump, even though the device is locked. Confidential information that is in memory at the moment the dump is created (such as a PIN code), can then be seized by the attacker. Confidential information is also accessible after a crash, for example.

SSDm-9: Unnecessary information in the cache memory					
Criterion (who and what)	Storing confidential information in cache memory is kept to a minimum based on a weighing of the risks per type of data; this applies both within as well as outside the personal app.				
Objective (why)	Prevent sensitive information from becoming (and remaining) accessible via the cache memory.				
Risk	Temporarily stored sensitive information falls into unauthorized hands.				
Reference	ISO27002	SSD	ASVS		
			V17.14 V17.18 V17.20		

Conformity indicators

/01 Weighing the risks

SSDm-9: Unnecessary information in the cache memory	
	Indicators
/01	Weighing the risks
/01.01	Keeping confidential information (temporarily) available in the cache memory is kept to a minimum, based on a weighing of the risks *per type of data*.
/01.02	The level of confidentiality of information determines how long or how briefly the information is kept in the cache memory and whether or not this information is rendered illegible, for example by overwriting it using data wiping.
/01.03	Where possible, truncating is applied. (See under Explanation).

Explanation

/01.01 When weighing the risks, take into account whether the duration of the temporary storage may be extended, because a service or the access to a file or network is not available for some time, for example.

/01.02 Java has a garbage collection mechanism for cleaning up data that has been left behind. Because it can be some time before the garbage collector is activated, this function is not regarded as adequate for deleting confidential data.

/01.03 When truncating is used, some of the characters of the confidential data are not stored, so that the remaining characters do not disclose the confidential nature of the data.

/02 Keep this to a minimum

SSDm-9: Unnecessary information in the cache memory	
	Indicators
/02	Keep this to a minimum
/02.01	Only sensitive information that is required for the operation of the app is stored.
/02.02	Confidential data that is stored on a server is not (also) stored on the mobile device.

SSDm-9: Unnecessary information in the cache memory	
	Indicators
/02.03	Storage of confidential information for the autocomplete functionality is prevented.
/02.04	Http(s) traffic is not cached.
/02.05	In case of backgrounds, extremely confidential information is deleted from the screen capture or rendered illegible using an overlay, then replaced by the app splash screen. (See under Explanation).

Explanation

/02.01 No confidential information is stored in cookies, web history, web cache, property files and SQLite files, for example.

/02.01 Be aware that iOS, Android and Windows save an App's screen capture as soon as this is backgrounded (if an app goes to the background and is not yet turned off). This means that confidential data that was displayed on the screen at that moment can be viewed. Thus, we recommend that you switch off the 'screen captures on backgrounding' function. However, this setting can frequently be changed by the user and is thus the user's responsibility, as well. With MDM this can be forced.

/02.02 With Android and iOS you can also use web views and JSON parsing. In such cases, the data remains on the server to the greatest possible extent; the data is then viewed with web views and JSON parsing.

/02.02 It is also possible to avoid importing confidential data into the app itself, but to process the confidential data on the server side and *only display the results* in the app. This measure can be combined with indicator /02.01 from SSDm-6: Reliable operation of the app.

/02.02 JSON (or JavaScript Object Notation) is a standardized data format and is an alternative to XML. JSON uses text that humans can read in the form of data objects that consist of one or more bits of data with a corresponding value. It is primarily used for exchanging data between the server and the app. JSON originally came from the JavaScript programming language, but is now a language-independent data format. Libraries for reading and creating JSON files are available for a wide range of programming languages (https://nl.wikipedia.org/wiki/JSON)

/02.03 When typing in a text field, the data can be stored and used by the autocomplete function. This means that confidential information can also be 'pre-sorted' and thus rendered viewable.

/02.04 The operating system can save http(s) requests to the backend server in its own cache, making the sensitive information included in the request available.

/02.05 With backgrounding, when an application goes to the background, the operating system creates an image of the screen/window of that application. When the application is then called up again, the operating system briefly displays that image during the transition from background to foreground. You can prevent this information from being displayed by shielding the confidential information in the image using an overlay when backgrounding.

3.10 SSDm-10: Timeout user session

When the user uses an app, a user session is open with the app. During this session, information is accessible via the app. Other people can abuse this information. Session termination ensures that the session is terminated after a prescribed period of inactivity.

SSDm-10: Timeout user session					
Criterion (who and what)	The app terminates a user session after a pre-set period of user inactivity using automatic session termination.				
Objective (why)	Preventing a session from being accessible to other people for a limited time when the user has left the session (on the device) unattended.				
Risk	The open session is used by someone with malicious intent.				
Reference	ISO27002	SSD	ASVS		
	11.3.2	12B	17.8		

Conformity indicators

/01 Pre-set period

SSDm-10: Timeout user session	
	Indicators
/01	Pre-set period
/01.01	Two minutes is used as the default, unless the functionality requires a different time period.
/01.02	If it is necessary to use a pre-set period that is longer than the default, the customer substantiates this and attunes this with the developer.

Explanation

/01 Inactivity in the user session is a period in which the user does not interact with the app.

/01.01 The strictest timeout period is maintained in conformance with https://www.owasp.org/index.php/Session_Management_Cheat_Sheet#Session_Expiration. This site states: "Common idle timeout ranges are 2-5 minutes for high-value applications and 15- 30 minutes for low risk applications.

/02 Automatic session termination

SSDm-10: Timeout user session	
	Indicators
/02	Automatic session termination
/02.01	The app activates automatic session termination after a period of inactivity specified by the customer.
/02.02	Session termination is identical to the user logging out.
/02.03	After session termination, when the user re-initiates (screen) activity, the app login screen is displayed and the user must enter his credentials again.

3.11 SSDm-11: Logging

During the log-in user actions and notices regarding the operation of the app can be written to log files. The logging can be used to track security incidents and errors in the app's operation; however, sensitive information can also appear in the logs. If this information falls into the wrong hands, not only is the user's privacy jeopardized; the information can also be used to detect weaknesses in the app. Access to this sensitive log information must thus be prevented.

SSDm-11: Logging					
Criterion (who and what)	Logging for debugging is disabled before the app is put into production; the log files are deleted. When statistical logging regarding the use of the app is performed, the logs do not contains any personal data.				
Objective (why)	Prevent non-essential detailed information regarding the user or the operation of the app from being accessed by an attacker.				
Risk	Sensitive information or information regarding weaknesses (in the security) of the app fall into the hands of an attacker.				
Reference	ISO27002	SSD	ASVS		
		SSD-30	V17.10 V17.13 V17.22		

Explanation

Debug logging is a function for registering activities and events in the app. Debugging offers the possibility to track events and errors during development, testing or use of the app. Activities and the events are included in a debug logbook.

Conformity indicators

/01 Logging for debugging

SSDm-11: Logging	
	indicators
/01	Logging for debugging
/01.01	The developer disables debug logging upon delivery.
/01.02	The app does not record any sensitive information regarding the user or the operation of the app. By default, this information is considered sensitive.
/01.03	The log files are deleted by the developer.
/01.04	The functions for storing crash logs and dumps are disabled by the developer.
/01.05	The app contains no test data when it is delivered.

Explanation

/01.01 Debug logging is used to track errors. To prevent various privacy and security risks, this logging must be disabled and the log files must be deleted when the app is put into production.

/01.01 Be aware that debug logging must also be disabled for referenced libraries, such as third party libraries.

/01.02 The recommendation is *do not log any information*. If the developer does decide to log information, he pledges to the customer that no sensitive information will be included in the logged information.

/01.02 If information is logged, this information is checked to determine whether it discloses detailed technical information, which could jeopardize the app's security. Tests are run to detect any possible disclosure of privacy-sensitive information regarding the user and any such disclosure is eliminated.

/01.04 Prevent a crash log or memory dump from being created in the event that the app crashes.

/01.04 The preference is to have the app handle error messages.

/01.05 Test data can occur in the files with the extensions .ipa, .apk and .jar., for example.

/02 Before putting the app into production

SSDm-11: Logging	
	Indicators
/02	Before putting the app into production
/02.01	Apps that are offered to the user by the developer or the tester do not contain any log files or functions for logging.
/02.02	Apps that are offered for testing and acceptance only contain those logging functions that have been determined to be required for testing and acceptance. (Logging in production is kept to a minimum. See for this /01.02)
/02.03	The presence of (information in) log files is checked upon delivery by the developer and by the app store before the app is offered to the public in the app store.
/02.04	It must be verified that the keyboard used does not record any credentials, financial information or other confidential information.

Explanation

/02.01 By linking all log and debug operations *to the release configuration,* all code for logging and debug operations is automatically disabled in the app version that is built for production. This prevents debug code and logging instructions from accidentally ending up in the product version.

/02.02 In specific cases, it can be useful to have logging functions available for testing and acceptance. This is then specified in a test plan.

/03 Statistical logging

SSDm-11: Logging	
	Indicators
/03	Statistical logging
/03.01	No sensitive information is written to the log files by the app or by any third-party program library.
/03.02	Logging is minimal and contains no personal data.

Explanation

/03.01 Logging of activities and events for functional objectives for the user is bound by the requirements that apply to storage.

/03.01 Personal information must be handled in conformance with the legal framework. See the Grip on Privacy[9] for additional information.

/03.02 Statistical information regarding the use of the app never contains information that can be traced to a person.

/03.02 Keep in mind that logging seemingly innocent requests can also result in leaks of user data, via the query string, for example.

[9] http://www.cip-overheid.nl/

3.12 SSDm-12: Session encryption

Encryption of a session between the server part of the application and the workplace part of the application protects confidential data that is being transported. Mobile devices frequently use open - and thus non-secure - WIFI networks. The communication is thus susceptible to man-in-the-middle (MitM) attacks. These days, masking the session by employing encryption that uses SSL, even when the information exchanged is not confidential, is a standard requirement for securing mobile devices.

Encryption is a necessity over networks that are deemed non-secure. Non-secure networks are networks that are not protected against unauthorized access. This also includes business networks in offices that are not *demonstrably physically* (and not electronically) protected. Because mobile devices are also used via non-secure and public networks, apps are required to use communication that is secured by encryption.

SSDm-12: Session encryption					
Criterion (who and what)	The application utilizes encryption for all communication via networks.				
Objective (why)	Guarantee the confidentiality, integrity and possibly the irrefutability of data delivery and transactions.				
Risk	The information exchanged comes under the influence of (is read and modified by) an attacker.				
Reference	ISO27002	SSD	ASVS		
	10.6.1 10.8.1 10.9.1	SSD-4			

Conformity indicators

/01 Encryption

SSDm-12: Session encryption	
	indicators
/01	Encryption
/01.01	The developer only uses protocols and cryptographic techniques that are deemed to be secure.
/01.02	The app encrypts the communication between the app and the server component.
/01.03	When services are invoked, the confidential information is encrypted in the call.

Explanation

/01.01 For the versions used within the discipline, no weaknesses are known that comprise a demonstrable threat.

/01.01 See https://www.enisa.Europe.eu/activities/identity-and-trust/library/deliverables/algorithms-key-size-and-parameters-report-2014.

/01.01 The rule that encryption must be based on 128bit or greater and TLSv1 or higher (TLSv1.1 and preferably TLSv1.2) applies. SSLv3 is no longer secure[10].
Moreover, it is important that weak encryption algorithms, such as DES and RC4 are disabled. We recommend using AES instead.

/01.01 Encryption based on an X.509 certificate (or comparable encryption) currently offers sufficient protection in most cases.

[10] See: https://blog.mozilla.org/security/2014/10/14/the-poodle-attack-and-the-end-of-ssl-3-0/

/01.01 Do not use weak algorithms to sign a certificate. MD5 is such a weak algorithm. Use SHA256 instead. Moreover, we recommend using a minimum of 2048 as the key length for RSA.

/01.03 The objective of a secure flag is to prevent a cookie from being sent in clear text (over http), which a third party could intercept[11]. Thus, on the server side the secure flag must be enabled in the http cookie header. When the secure flag in the http header is disabled, the browser/app also sends a cookie over the non-secured connection, in other words over http rather than https.
/01.03 See also https://www.antagonist.nl/blog/2013/11/xss-and-csrf/
/01.03 This applies, for example, to privacy-sensitive information in URLs.

/02 Communication

SSDm-12: Session encryption	
	Indicators
/02	Communication
/02.01	All communication over the network is encrypted by default.

Explanation

/02.01 It is assumed that communication can be conducted over non-secure networks.
/02.01 Keep in mind that it is always possible that a user may unintentionally select a non-secure network.

[11] For more info: https://www.owasp.org/index.php/SecureFlag.

3.13 SSDm-13: Certificate pinning

One of the most important security measures for apps is the security of the communication with the server side using encryption (SSDm-13).
Certificates for encryption are issued via a trusted Public Key Infrastructure (PKI), by which a certificate is issued by a certificate authority (CA). Generally it is a root CA and various sub CAs who issue certificates. The ultimate certificate provided contains the specific (collection of) URL(s), for which the certificate is issued.
As long as a certificate authority is not compromised, the certificates it issues are secure. However, practice has shown that CAs can become compromised, as has happened with DigiNotar or Verisign, for example. Thus, a supplemental security measure must be applied: certificate pinning.
With certificate pinning, the app checks to see if the certificate used has been issued by a trusted certificate authority and checks to determine whether or not the certificate is unchanged (not compromised). Thus, with certificate pinning the certificate is validated against one specific CA or final certificate[12]. This prevents the risk of a MitM if COMODO, DigiNotar, TürkTrust or another non-used CA is hacked and the hacked certificate is used. If the certificate is issued by a non-trusted certificate authority, or is not used by the server/ URL specified in the certificate, the app can reject the connection.

SSDm-13: Certificate pinning					
Criterion (who and what)	When setting up an encrypted connection, the app checks to see whether the server certificate is trusted and takes the requisite measures.				
Objective (why)	Guarantee the confidentiality, integrity and, if desired, the irrefutability of data and transactions using a secure key or a secure certificate.				
Risk	The information exchanged comes under the influence of (is read and modified by) an attacker, in spite of the fact that an encrypted connection is used.				
Reference	ISO27002	SSD	ASVS		
			V17.1 V17.15		

Conformity indicators

/01 Trusted

SSDm-13: Certificate pinning	
	Indicators
/01	Trusted
/01.01	The developer equips the app with certificate pinning for all communication over the network.

Explanation

/01.01 By default, certificate pinning takes place by 'hard coding' a certificate in the app. The disadvantage is that each new 'pin' (trusted CA) must be added to the code, making switching to another CA a laborious process.

/01.01 As an alternative (with iOS and Android) you can use a TrustManager, with which hashes of certificates (hard coded or from a configuration file) are used to verify the certificates. The benefit is that certificates derived from the root certificate can easily be added, making it simpler to switch to another certificate authority. This implementation provides greater flexibility; however, there is a greater chance that errors may occur during implementation. This can result in failed (or missing) SSL checks.

[12] More about 'pinning': https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning#What_Is_Pinning.3F

/02 Measures

SSDm-13: Certificate pinning	
	Indicators
/02	Measures
/02.01	In the event of a non-secure certificate, the user is informed of the consequences and the risks.
/02.02	Based on a risk analysis, the developer specifies whether the user must just be informed or, whether in addition to informing the user, the communication must also be terminated.
/02.03	The developer has procedures ready in the event that a certificate becomes non-secure.

Explanation

/02.02 To this end, the customer specifies in advance what the consequences arise if the data becomes compromised.

/02.02 To this end, the developer specifies what consequences arise if the app becomes compromised.

/02.03 This proactively limits that the consequences if the certificate or key becomes non-secure and thus the communication becomes non-secure or the app becomes unavailable.

/02.03 In so doing, the developer can prevent the app from being used in a non-secure manner, after the certificate is no longer trusted. The developer then has an alternative certificate authority available, for example.

3.14 SSDm-14: Hardening apps

When apps are hardened[13] the app's communication facilities are restricted to a minimum (to those that are strictly required). One of the ways to achieve this is by deleting or disabling unnecessary interfaces. By inventorying the requisite interfaces and then determining the dependencies, you can compile a minimum list of interfaces which the app must have available. All other interfaces can be deleted. Be aware that even if interfaces are non-active, if they appear on a system they can ultimately lead to a vulnerable app. Thus, it is safer to keep the attack surface as small as possible. Thus, unnecessary interfaces and rights (privileges) should preferably be deleted.

SSDm-14: Hardening apps					
Criterion (who and what)	The developer ensures that access to the app is only possible via interactions that are required for the application to function properly.				
Objective (why)	The app and the data are secured against additional chance of exploitation via weaknesses, by limiting the functionality to what is strictly necessary for the app to function properly.				
Risk	An attacker breaks in via unused entry points on the app, allowing the confidential data and the app to fall under the influence of an attacker (read and modify).				
Reference	ISO27002	SSD	ASVS		
		SSD-1			

Explanation

Limiting access paths, and thus limiting services in the app, is done to increase the app's security. Each extra access path potentially offers an attacker an extra possibility to break in or to acquire information. It is possible to reduce the number of potential attacks by eliminating app functions that are not strictly required for the app to function properly. When it is impossible to delete such functions, all facilities that are not strictly required must be disabled.

Conformity indicators

/01 Interactions

SSDm-14: Hardening apps	
	Indicators
/01	Interactions
/01.01	Programming code, libraries and components that are not used are disabled or, where possible, deleted.
/01.02	Non-essential protocols or functions with which the app can be accessed (e.g. by other apps) are disabled or, where possible, deleted.
/01.03	Authorization is only granted if this is necessary for the app to function properly.
/01.04	If a service is accessible outside your own app, it must satisfy the SSD requirement: SSD 26: HTTP methods.

Explanation

Interactions can work either synchronously or asynchronously, are available in the background or offer interaction with the user.

- Services: A service is part of an app that runs (in the background) that provides functionality for interaction with other applications without requiring any interaction with the user. Services can start

[13] In computing, 'hardening' is usually the process of securing a system by reducing its surface of vulnerability. Reducing available paths of attack typically includes removing unnecessary software, unnecessary user names or logins and disabling or removing unnecessary services; https://en.wikipedia.org/wiki/Hardening_(computing).

other background processes and exchange data with other apps via so-called 'content providers'. In addition to services, within Android there are also so-called *Intents*, with which asynchronous communication is possible.

- Publish/subscribe services: These are services that - compared to ordinary services - are extremely suitable for question and answer interactions in a system in which apps/services are not offered by one party (whether or not this system is distributed across multiple devices). The questions and answers are routed via junctions that harmonize the questions and answers with one another. An Android example is a 'broadcast receiver'. With iOS, this is a 'Notification Center Service'.

/01. Deletion is preferred to deactivation.
/01.01 An app interface for communication outside the app is only available and accessible (using access rights), if the objective and thus the importance of the user is substantiated.
/01.01 Interfaces can be offered in the form of a service. With Android, asynchronous communication can go via Intents. The deployment of so-called URL schemes in iOS is somewhat comparable to this.
/01.01 In Android, the developer leaves the export flag in the manifest file set to false, so that the services are not accessible outside the (shielded) app. To determine access rights, the app looks to this flag and to who is allowed to call the service.

/02 Required

Most libraries that are used for app development contain more functionality than the app needs for the objective for which it is developed. To prevent unsafe (obsolete) protocols or functions from being active, the app keeps track of what interfaces are used. As part of "SSDm-3: Up-to-date apps", the versions of the app used are guaranteed to be up-to-date.

SSDm-14: Hardening apps	
	Indicators
/02	Required
/02.01	An interaction is exclusively used by an app if the objective for that use and the interests of the user substantiate this.
/02.02	The software supplier makes the app available with a current overview of the requisite interaction options and provides the justification.
/02.03	The software supplier makes the app available with a current overview of the requisite libraries.
/02.04	The app is equipped with the most recent, relevant versions of the communication facilities.
/02.05	By default, deprecated ('rejected') code does not appear in the app.
/02.06	If deprecated code must be used in the app because otherwise older versions of the operating system can no longer be supported, the developer conducts a risk analysis of this. The results of the analysis are submitted to the customer for review and approval.

Explanation

/02.01 After building, each modification of the app is tested to ensure that the modified app does not offer more interaction options than are strictly required by the design.
/02.01 Within the privacy domain, the guarantee of the confidentiality of the user data and thus the user's interests (here the objective for which the user utilizes the service) is referred to as purpose limitation.
/02.01 For Apple, for example, the iOS Notification Center Service first notifies the user of the situation, based on which he can grant permission in advance for the exchange of information.

In line with the Personal Data Protection Act (see Grip on Privacy[14]), the notification must notify the user in advance regarding the information to be exchanged and the objective of this exchange.

/02.04 For more information, see: http://www.jssec.org/dl/android_securecoding_and.pdf

/02.05 Deprecated code is software that is rejected by the original developer (or on the platforms for developers), for example because it contains weaknesses or leads to weaknesses in the app's operation.

/02.05 Parts of APIs (classes or functions) can be deprecated. This applies for the platform APIs, but can also apply for third party libraries. The developer of that API advises against continuing to use specific functions, generally because those functions are obsolete and involve risks. You must keep in mind that deprecated functionality is deleted after some time.

/02.05 If specific functions are deprecated, new facilities are generally available for realizing the intended functionality. Where deprecated code appears in an old version, this is either replaced by the most current version or this function is no longer used.

/02.06 Depending on the oldest version of the operating system that must be supported, new facilities may only be available as of a specific API. Maintaining the deprecated logic is the only method for being able to run on older versions. You must weigh the risks to determine whether you want to support both the old and the new method, or whether you choose the API in which the relevant new functionality is available and thus require another version of the operating system.

[14] http://www.cip-overheid.nl/

3.15 SSDm-15: Least Privilege for other apps

Risks of system abuse can be considerably reduced by limiting rights on the app. Commonly used principles for the policy are based on 'no access by default', 'least privilege' and 'need-to-know', for example. This applies to users, but also to apps mutually. In conformance with the 'least privilege' principle, the rights on an app are limited to the minimum set of rights required for the app to function properly.
In the event of a remote hack of the app, attackers can do everything the app can do. For example, if the app has (unnecessary) rights for the use of the camera, attackers can abuse this (unnecessary) right to remotely take photos and videos. Thus, you should restrict the rights to a minimum to keep the app's risk profile as low as possible. In designing the application, you must thus keep the principle of least privilege in mind.

SSDm-15: Least Privilege for other apps					
Criterion (who and what)	The app's access rights to functions and data are only issued in those cases in which they serve the user's substantiated objectives and interests.				
Objective (why)	Other apps and users only have those rights to an app that are required for the app's proper operation.				
Risk	An attacker, user or compromised app uses unnecessarily granted rights, making it possible for the operation of the app and sensitive information to fall under the influence of unauthorized parties (read and modify).				
Reference	ISO27002	SSD	ASVS		
		SSD-8	V17.9 V17.23 V17.26 V17.28		

Conformity indicators

/01 Access rights

SSDm-15: Least Privilege for other apps	
	Indicators
/01	Access rights
/01.01	In the design, the developer takes preventing the allocation of unnecessary rights to other apps into account.
/01.02	Based on a risk analysis, the developer specifies what access rights, which are required for the proper functioning of the app, must be granted.
/01.03	Using access rights, the app's user interface is exclusively made accessible to trusted apps.
/01.04	Authorization for use of an app's function and/or access to data must be explicitly granted.

Explanation

/01.01 Rights may already be unnecessarily granted to other apps in the design.

/01.03 With Android a user interface is designated as an Activity. An activity is generally used for the interaction with the user.

/01.03 With Android, rights are set with android:permission.

/01.03 For more information see: http://www.jssec.org/dl/android_securecoding_and.pdf

/01.04 The rights required by the app are tracked in an app manifest file in the <permissions> element.

/01.04 Within Android, permission can be granted (in the Android Inter Process Communication System (IPC)) at three levels:

- Private**:** With Private IPC there is no exposure outside the app, making this the most secure type of IPC.
- Partner**:** The partner IPC can only be used by trusted (your own) other apps. Other apps can be trusted because they are signed with the same key.
- Public**:** The Public IPC can generally be used by other apps and thus provides no protection (MODE_WORLD_READABLE or MODE_WORLD_WRITABLE). This setting is regarded as extremely dangerous; these days, it is no longer considered suitable for achieving information security [15].
- Least Privilege**:** Least Privilege here means: "Private, unless" and then "Partner, unless".

/01.04 See footnote[16] for more information.

/01.04 This prevents an unsafe app from gaining access to information entered into the app's user interfaces by the user.

[15] See: http://developer.android.com/reference/android/content/Context.html.

[16] http://developer.android.com/guide/topics/security/permissions.html
https://www.owasp.org/images/c/ca/ASDC12 -An_In_
Depth_Introduction_to_the_Android_Permissions_Modeland_How_to_Secure_MultiComponent_Applications.pdf

3.16 SSDm-16 Input normalization

Like all software, apps depend strongly on all sorts of input, such as input by the user, data received from external servers, other apps and local files. Apps are extremely dependent on common web technologies, such as JSON, XML, SQL, HTML and JavaScript. Input can contain characters or commands that affect the operation of the app. This input does not satisfy the rules for secure input.
Just as (web) applications on the server side must be shielded, apps must be protected as well. When the app does not handle malicious input properly, data that must be shielded by the app can be accessed using specific input.

SSDm-16 Input normalization					
Criterion (who and what)	The app prevents manipulation by normalizing all input received before validating it.				
Objective (why)	Prevent the viewing, change, loss and misuse of data by entering characters or commands in the app input that make this possible.				
Risk	Attackers can influence the operation of the app and acquire, modify or add data.				
Reference	ISO27002	SSD	ASVS		
		SSD-19			

Explanation

Normalizing contents means that the contents satisfy a number of restrictive rules. By employing normalization, the app prevents malicious requests from being able to abusively breach the filters for validation. In addition, the input is presented in such a way that it can be securely processed at all places in the app (elimination of SQL and HTML injections). Normalization is also known as anti-evasion[17] or canonicalization[18].

Conformity indicators

/01 Received input

SSDm-16 Input normalization	
	indicators
/01	Received input
/01.01	The developer maintains an inventory of all the app's entry points.
/01.02	Normalization is performed on all entry points from user interfaces, on other apps and files on the mobile device to external sources (via the network).
/01.03	All trusted sources are whitelisted in the app.

Explanation

/01. The app receives input from the user, from other apps, the backend and services on the mobile device and via the network. This input can take different forms. The app must normalize the input before the input can be validated via the filtering mechanisms.

/01.02 In addition to answers from the server side that are modified by a MitM or server hack, user input can also come from an attacker. After all, an attacker can gain physical access to the mobile device. Other apps, intents or files on the mobile device, including the SD card, or

[17] https://www.owasp.org/index.php/Virtual_Patching_Best_Practices#Anti-Evasion_Capabilities
[18] https://www.owasp.org/index.php/Canonicalization,_locale_and_Unicode

communication at public locations can also generate manipulated input. Thus normalization must be performed on all entry points.

/01.03 A check is run on the entry points to verify that the source is trusted.

/02 Normalization

SSDm-16 Input normalization	
	Indicators
/02	Normalization
/02.01	The app checks the input for suspicious characters or commands.
/02.02	Among other things, (but in any event) the app does the following: - Converts NULL characters to spaces; - Converts the code of special characters to a uniform character code, such as UTF-8; - Normalizes path changes such as '/./' and '/../'; - Deletes unnecessary spaces and line breaks; - Deletes unnecessary white spaces; - Converts backslashes '\' to forward slashes '/'; - Converts mixed case strings to lower case strings.

Explanation

/02.01 The guidelines for input handling apply to *all* input that comes from outside the app. Thus this applies not only to (end) users, but also to external systems and applications, other apps, the backend and services on the mobile device and via the network, as well.

/02.02 Also use the OWASP guideline for data validation: https://www.owasp.org/index.php/Data_Validation.

/02.02 At a minimum, use the standard services that exist for this.

/02.02 Convert all risky characters to a safe format using 'escaping'[19]. Keep in mind the differences between character sets, for example ASCII and UTF-8[20].
As an example:
The escape (\) preceding a character indicates that the character must be interpreted literally, <\"> is <">. Escaping can also be achieved using the hexadecimal value of a character, <\x22> is <">[21]. However, this does not produce the desired text if this is compiled with an ASCII[22]-incompatible character set[23].

/02.02 Characters from the input that can be processed and are not undesirable can still be risky when they are used within the program logic. Risky characters can be part of legitimate input. For example, take the city <'s-Gravenhage>. This leads to a syntactically incorrect query[24]. If you place an escape before the apostrophe, the database considers the apostrophe to be part of the input string and not part of the query. Many programming languages support standard services for escaping dangerous characters.

[19] For example by: `value.decode('string_escape')`; See: http://stackoverflow.com/questions/18219398/how-to-convert-characters-like-x22-into-string

[20] ASCII was the most frequently used character set on the Internet until December of 2007; after that date, UTF-8 became the most commonly used character set.

[21] The number 22 is the hexadecimal ASCII value for a double quotation mark.

[22] http://en.wikipedia.org/wiki/ASCII

[23] ASCII was the most frequently used character set on the Internet until December of 2007; UTF-8 became the most commonly used character set after that date.

[24] SELECT * FROM news WHERE title LIKE '%'s-gravenhage%'

/02.02 Comparable conversions apply for HTML, XML, et cetera. Perform escaping on the input after applying whitelists and eventual blacklists. Escaping must be tailored to the program components in which input is processed.

3.17 SSDm-17: Input validation

The most important rule of thumb for input in an app is that the application cannot trust any input and thus must validate all input. Thus, the app must validate all input for accuracy, comprehensiveness and validity before processing it. Moreover, at a minimum, the input must be validated for values that fall outside the valid scope (limits), for invalid characters, missing or incomplete data, for data that is not in the right format and for inconsistency of data compared to other data within the input or other data files. Non-trusted input can come to the app from all sorts of access paths, such as the intents, the services, the network traffic, binding interfaces and access to files. Input validation is *the* condition for reliable data processing; invalid input must be rejected by the app.

SSDm-17: Input validation					
Criterion (who and what)	The app prevents the possibility to manipulate input by validating all input received before that input is processed.				
Objective (why)	Prevent deliberate (or unintentional) manipulation of the app via the app's input.				
Risk	Attackers can influence the app's operation and acquire, modify or add confidential data.				
Reference	ISO27002	SSD	ASVS		
		SSD-22	V17.19		

Explanation

Uncontrolled (non-validated) input from users is a significant threat for an app. If user input is used directly in HTML output, cookie values, SQL queries, etc., there is a considerable chance that a malicious party can compromise the app. A lack of input validation can lead to XSS, command and SQL injection vulnerabilities:

- Cross-site scripting (XSS): With XSS, in addition to the traditional XSS exploits, attackers can start native code used by utilizing WebViews.
- SQL injection: When using a local database, SQL injection makes it possible to read, change and delete locally stored data.
- Command injection: Manipulating the input for a command modifies the app, making it possible to read, change and delete the locally stored data.

A number of principles apply with regard to input when developing apps. These principles are:

a. Other apps cannot be trusted; thus the input that comes from such apps cannot be trusted, either.
b. Before being validated, the input is first normalized (SSDm-16 Input normalization).
c. Input that does not satisfy one or more of the checks is deleted or rejected; only the characters that occur in a previously defined list are permitted. This approach is also known as the whitelist approach.[25]

In the app development process, the software must be explicitly studied for correct application of these principles. This requires extensive testing or targeted code reviews.

[25] https://www.owasp.org/index.php/Data_Validation#Sanitize_with_Whitelist.

Conformity indicators

/01 Received input

SSDm-17: Input validation	
	indicators
/01	Received input
/01.01	The developer has kept an inventory of all entry points for the app.
/01.02	(After normalization), at all entry points any input from user interfaces, other apps and files on the mobile device, and from external sources via the network is validated.
/01.03	Trusted sources are whitelisted in the app.

Explanation

/01.01 The app receives input from the user, other apps, the backend and services on the mobile device as well as via the network. This input can come in different forms.

/01.01 In addition to answers from the server side that are modified by a man-in-the-middle (MitM) or a server hack, user input can also come from an attacker. After all, an attacker can gain physical access to the mobile device.

/01.01 Other apps, including Android intents or files on the mobile device, which includes the SD card, and communication at public locations can also generate manipulated input. Validation must thus be performed at all entry points.

/01.02 The app must normalize the input first, before the input can be validated using filtering mechanisms.

/01.03 Even when an app conducts validation and filtering, this filtering is frequently not effective enough to block all possible attacks on the app. This is predominantly the case the moment the app uses blacklisting. With blacklisting, the dangerous sources must be known so that they can be blacklisted. With whitelisting, all the dangerous sources do not have to be known; only trusted sources are permitted.

/02 Validating

Validating the contents ensures that only valid data is processed. Validation takes place at both the protocol level (mostly HTTP) (SSD-18) and the application level. The objective is to prevent software at the application level from being abused or failing due to user input.

SSDm-17: Input validation	
	indicators
/02	Validate
/02.01	The app validates all input.
/02.02	Incorrect, invalid or forbidden input is rejected.
/02.03	WebView calls are only executed after the input has been validated.
/02.04	The permitted methods in the WebViews are whitelisted.
/02.05	If JavaScript is required, the addJavaScriptInterface () is only used for those web pages for which all input is reliable.

Explanation

/02.01 Confirm the contents of an HTTP request based on processable input (whitelist). Validate the input for malicious keywords, characters and patterns (blacklist).

/02.03 WebViews ensure that apps can open web content from online sources within the app.

/02.03 In terms of risks, WebViews are, in fact, comparable to web browsers, because they are vulnerable to all sorts of browser-related actions, such as origin policy bypasses, JavaScript parser, buffer overflows, and cross-site scripting.

/02.03 From the perspective of information security, WebViews poses risks that cannot easily be resolved. These risks mean that an attacker is able to gain control over (part of the) information loaded in the WebView. This can occur in many ways, such as man-in-the-middle (MitM) attacks, client side injection or with cross-site scripting.

/02.04 The JavaScript interface that is frequently used in WebViews and offers access to the underlying WebView code (JavaScript) and native code (Java) can cause security problems. On older Android versions, the JavaScript interface can be exploited to execute native code using the JavaScript on the device. This problem is resolved in API level 17 (Android 4.2) by adding the Javascript Interface annotation, by which permitted methods are whitelisted. However, devices are vulnerable if:

- They run on an Android version older than 4.2 or:
- The app is developed and built in API level 16 or lower (SDK) and runs on an Android version prior to 4.4.

/02.05 In general, only the JavaScript in your own app is reliable.

/02.05 Android has the possibility to limit the start-up of Java scripts, making script injection impossible. If the app does not use Java scripts, be sure that the method call setJavaScriptEnabled() never occurs. This applies for both your own app code as well as for any third party code that is included.

For more information regarding Input Validation, see:
https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet

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

To share data between apps, the model offers the recommended method with a *content provider*. The interfaces that use SQL are susceptible to SQL injections, if these interfaces are not used securely. The interfaces can consist of native code in the apps or of web components (HTML5).

SQL injection: With SQL injection, an attacker can expand the input of permissible SQL commands to include his own SQL commands. Thus, the attacker may be able to create, change, edit, read or delete data in the database. Such non-trusted input can come to the app from all sorts of access paths, such as the intents, the services, the network traffic, binding interfaces and access to files. Android content providers and the SQLite Query Builder offer complete support for parameterized queries. When parameters are used, the input only consists of the specific variables or parameters of the query. The code for the query itself is recorded in the app. Validating the variables and parameters entered prevents SQL injection.

More information regarding SQL injection and content providers can be found at:
https://www.owasp.org/index.php/SQL_Injection
http://developer.android.com/guide/topics/providers/content-provider-basics.html
http://developer.android.com/training/articles/security-tips.html#manifest

3.18 SSDm-18: HTTP methods

The web server supports the HTTP protocol. HTTP has methods, headers and error information that can be exploited. Thus, use of HTTP must be restricted to the minimum required for the proper operation of the apps that are made available.
HTTP 1.1 and 2.0 support different functions. In practice, an app only uses the GET and POST functions. For many scripts and objects, only the GET function is required. The other functions are virtually never required within traditional apps; these comprise an extra security risk.

Fout! Verwijzingsbron niet gevonden.					
Criterion (who and what)	The app only uses the HTTP functions required to communicate with other apps and services, whether or not this takes place over the network.				
Objective (why)	Prevent the use of non-essential methods that can be used to manipulate the app's logic.				
Risk	The operation of the app is manipulated, bringing it under the control of an attacker.				
Reference	ISO27002	SSD	ASVS		
		SSD-26	V17.19		

Conformity indicators

/01 HTTP functions

Fout! Verwijzingsbron niet gevonden.	
	Indicators
/01	HTTP functions
/01.01	The app only uses GET and POST as http methods. The software supplier substantiates and describes any requisite methods other than GET and POST and records this in the design documentation.
/01.02	In the configuration documentation, the software supplier specifies what HTTP methods are used.

Explanation

/01.01 The design documentation specifies:

- what HTTP methods are important for HTTP to function.
- what HTTP headers are important for HTTP to function.
- what HTTP request methods (GET, POST, et cetera) are required for the supported apps.
- what information in the HTTP headers is important for the function.
- what standard error message(s) are displayed/sent.
- the manner in which the preceding is realized; for example, think of the configuration of the web server and, if applicable, the application level firewall.

Any deviations from the preceding principles that are required because the app cannot function without these deviations must be substantiated.

/01.01 Methods other than GET and POST are virtually never necessary within traditional apps and only comprise an extra security risk (exploitation).

/01.01 In a CORS pre-flight 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 is conducted to ensure that this does not have any adverse consequences for security. More information regarding CORS and pre-flighted requests can be found at: https://developer.mozilla.org/and-US/docs/Web/HTTP/Access_control_CORS.

/01.03 When using MDM, blocking non-essential HTTP methods is recommended in all cases.

3.19 SSDm-19: XML external entity injection

Along with JSON, XML is a much used format for reading data in the app. The input based on the XML format contains data that is surrounded by codes in a defined structure: in XML, the entities (data) are displayed between an opening and a closing tag.
The codes determine the structure and meaning of your data. Because the structure and the meaning of the data is described, the data can be reused in a number of ways. This possibility for portability has made XML one of the most commonly used technologies for the exchange of data. Thus, XML is used a lot in apps. Parsers are generally used to read out data from the XML input. Android offers three types of XML parsers for this purpose: XMLPullParser, DOM and SAX. iOS offers two parsers: NSXMLParser and libxml2. Each parser is susceptible to attacks using external entity injection (XXE) if the right measures are not taken.

Fout! Verwijzingsbron niet gevonden.					
Criterion (who and what)	The app limits the possibility of manipulation by protecting all external XML input from entity injection.				
Objective (why)	Prevent a DoS attack by manipulating the external input that is based on XML.				
Risk	An attacker renders the app inaccessible to the user.				
Reference	ISO27002	SSD	ASVS		

Conformity indicators

/01 External XML input

Fout! Verwijzingsbron niet gevonden.	
	Indicators
/01	External XML input
/01.01	The app validates all input that does not come from a trusted source.
/01.02	Incorrect, invalid or forbidden input is rejected.

Explanation

/01.01 The entities in XML can be nested in one another endlessly. With XXE, this endless nesting leads to a Denial of Service for the parser. The DoS is caused because the problem is not resolved in the parser locally, but leads to extra network traffic caused by continually re-accessing the XML source.

/01.01 More information regarding XXE attacks can be found at the following location: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing

/02 Entity injection

Fout! Verwijzingsbron niet gevonden.	
	Indicators
/02	Entity injection
/02.01	When calling the parser for external XML sources, the app turns off the entity resolver. This shuts off the parsing of the namespace and document definitions.

Explanation

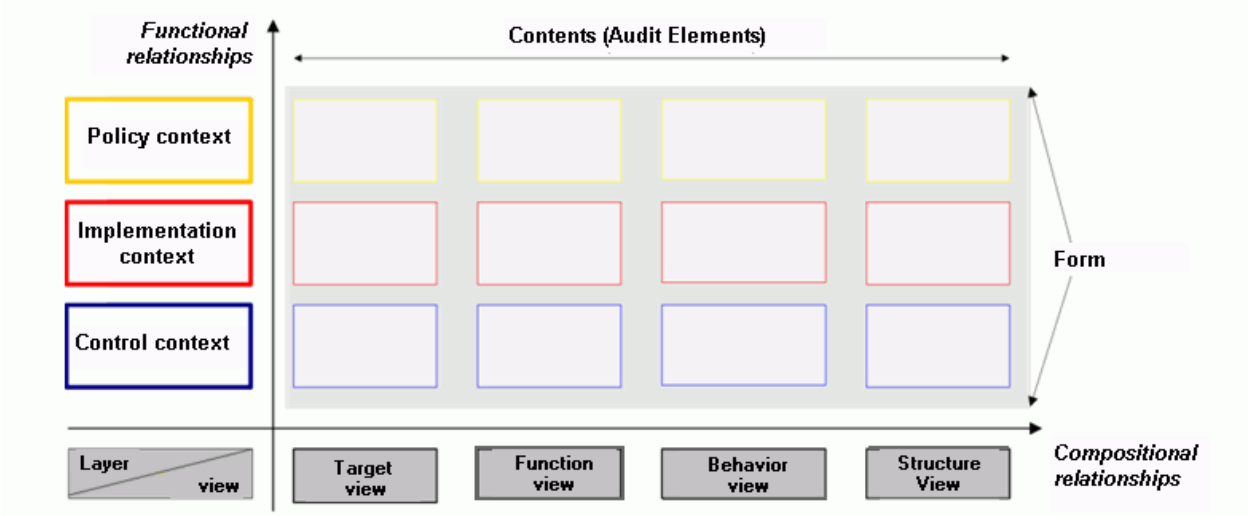
/02.01 The DoS is caused because the problem is not resolved in the parser locally, but leads to extra network traffic caused by continually re-accessing the XML source.

/02.01 More information regarding XXE attacks can be found at the following site: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing

Appendix: The SIVA method in brief

The description of the SSD security requirements for mobile apps is based on the SIVA method [Tewarie, 2014]. The SIVA method maintains a framework that is sub-divided into domains, with a separate general part that contains policy aspects and management aspects. This framework contains specific layers and columns in order to display a connection between the security measures. Having to fill in this framework helps ensure that the norms are complete.
This method for describing the norms makes it clear who must do what within a norm, while the SIVA method for each security requirement makes the context clear.
The security requirements here focus on the 'software' object (the app) and thus do not impose any requirements on the production process of an app or the policy for the app.

Framework



The SIVA framework consists of four components: *Structure*, *Contents*, *Form* and *Analysis sequence*.

These components are tools. They are described as follows:

- *Structure*: The environment, in this case the application environment, is divided into a number of domains. This promotes comprehensiveness, relevance, clarity and the relationship of the aspects that are studied.
- *Contents*: Basic elements are identified per domain from different perspectives.
- *Form*: The security requirements are formulated per element using formulation regulations (template).
- *Analysis sequence*: An iterative analysis process for the layers specified under structure.

Analysis sequence deals with the process for arriving at the security requirements and is not relevant here, because we base this on existing norm frameworks to the greatest possible extent. This also applies for the structure. The *structure* consists of three distinguished contexts: the policy context, the implementation context and the control context. Because the SSD(m) security requirements focus on the requirements regarding the implementation of applications and thus on the implementation context, no requirements are imposed from the policy context and the control context. When drawing up the SSD(m) security requirements, the *Contents* and the *Form* are used as tools.

Contents

In the SIVA method, contents are shown from perspectives: objective, function, behavior and structure (OFBS). A specific collection of basic elements (objects) is identified from each DFGS perspective. The perspectives include the following:

objective: this is the 'why' aspect, the reason for the organization's existence. Examples of basic elements from *objective* include: organization, vision, objectives, laws and policy, stakeholders and tools.
function: this is the 'what' aspect. The organizational and technological elements that must achieve the organization's objective. Examples here include: organizational and technical functions, processes, tasks and task requirements.
behavior: the 'how' aspect (behavior aspect). This involves the human and technical resources and their features that must give the organizational and technical functions their form. Examples: actor, object, interaction, state, characteristic and history.
structure: the 'how' aspect (form aspect): the method by which an organizational and personnel structure is given form. Examples include: business organization structure, business architecture, IT architecture and business-IT alignment.

The relationships between the objects from the DFGS perspectives can be read as follows: the elements from the *objective perspective* are regulated and/or are achieved by elements from the *function perspective*. The elements from the *function* perspective use or realize the elements from the *behavior perspective,* which in turn are given form by elements from the *structure perspective*.

The following checklist is used for the SSD security requirements to determine whether the security requirements cover the areas of attention (basic elements) for apps - and thus cover the areas of risk.

Implementation context		
App		
Perspectives	**Basic elements**	**Identified elements**
objective	policy	operational policy
	tools	apps
function	process	app rights
behavior	object	app
	protocol (input)	app input
	protocol (output)	app output
	link	app link (front-end) – server side (Web)application (back-end)
	connection time	app session
structure	architecture	context of the app

The security requirements in Chapter 3 cover the elements to be identified.
However, that does not mean that the areas of risk are completely covered. The SSDm security requirements are set up so as to be able to get a grip on the security of applications and not to provide a complete list, as intended with the fully implemented SIVA method.

Form

The *Form* component specifies the security requirement (the principle) as a fixed formula (syntax):

Predicate	{ object 1,	object 2,	object 3 }
Action type	{ who ,	what ,	why }

Four elements appear in the formula. The first element is the action (action type). The second and third elements are the objects that perform the action (actor, who) or on which the action is performed (what), respectively. The fourth element represents the result or objective of the action. The following diagram briefly details these elements.

Who	Actor involved. Specifically for SSD(m), the actor is described in the criterion or with the indicators. *If no involved actor is specified, the (ultimate) responsibility rests with the app's builder.*
What	Here items are expressed, such as: • what must be done to achieve/check/monitor the objectives and to be able to provide justification, • what someone must do or • what a technical function/device does.
Action type	Specific verbs related to the 'what' aspect and to the specific layer in which the aspect is described.

Template

The elements "what" and "why" are included separately. In the expression of the security requirements, the keywords that serve as indicators are used. Indicators are designated for each keyword. The indicators provide insight into how the security requirements can be satisfied. The keywords in the formulation of the security requirements ensure that only relevant criteria are designated per security requirement. The template used for the security requirements is:

SSD no. Subject of the norm					
Criterion (who and what)	What (xxxxxx) <verb>xxxxxkey wordsxxxxx				
Objective (why)	The reason the norm is used.				
Risk	The risk that provides the incentive for maintaining the norm.				
Reference	Source 1	Source 2	…		

Each keyword *constitutes* an indicator, which must be satisfied. Each keyword is further detailed with this in mind. The template used for keywords is:

SSD no. Subject of the norm	
	Indicators
/01	keyword
/01.01	indicator 1.1
/01.01	indicator 1.2
…	…