jamk.fi

Application security

In Android application development

Aleksi Vepsäläinen

Bachelor's thesis
May 2016
Technology, communication and transport
Degree Programme in Software Engineering

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Vepsäläinen, Aleksi	Bachelor's thesis	5 2016
		Language of publication: English
	Number of pages 49	Permission for web publication: x

Title of publication

Application security

Case: Android application development

Degree Programme

Software Engineering

Supervisor(s)

Väänänen, Olli

Assigned by

Digia Finland Oy

Abstract

Digia Finland Oy needed guidelines for secure mobile application development in the case of developing mobile application products in the future. The objective was to find the most common pitfalls in the mobile application development to avoid most obvious security holes in the early development of the application to minimize costs of implementing them later on into the ready application.

Because there are several mobile operating systems on the market and covering specific all of them would have been huge effort, it was decided to focus on the most common OS in the market, the Android. Additionally, it was decided that in the Android system the focus would be on the native applications, leaving web applications and web view components out of the scope of the paper.

It was discovered that most Android applications are bad from security aspect, and even important and trusted applications, such as banking applications could have severe security holes in them.

Most of the security issues are not actually Android application specific, and they do cover also native web applications and mobile applications of other operating systems. Even as there were very Android specific issues, none of them were unsolvable, which lead to the conclusion that Android system itself is not bad bearing in mind the security aspect but rather developers of mobile applications do not focus on security issues.

Keywords/tags (subjects)

Cyber security, Mobile devices, Software development

Miscellaneous

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Vepsäläinen, Aleksi	Opinnäytetyö, AMK	5 2016
	Sivumäärä 49	Julkaisun kieli Englanti
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Application security

In Android application development

Tutkinto-ohjelma

Ohjelmistotekniikka

Työn ohjaaja(t)

Väänänen, Olli

Toimeksiantaja(t)

Digia Finland Oy

Tiivistelmä

Digia Finland Oy tarvitsi selvityksen siitä, mitä tietoturva-ongelmia voisi tulla mahdollisesti vastaan mobiililaitteille ohjelmistoa kehittäessä. Työn tarkoituksena oli selvittää yleisimmät mobiilipuolen tietoturvariskit niiden eliminoimiseksi jo ohjelmistokehityksen alkuvaiheessa, vähentäen tietoturvaratkaisujen implementoinnin aiheuttamia kuluja myöhemmissä tuotteen kehitysvaiheissa.

Koska mobiiliohjelmistoja voidaan kehittää lukuisille eri käyttöjärjestelmille, työtä päätettiin rajata käsittämään vain kaikkein käytetyin mobiilikäyttöjärjestelmä, Android. Työtä päätettiin vielä rajata käsittämään vain Androidin ”natiivit”-applikaatiot, jättäen web-aplikaatiot, sekä webview-komponentit pois työstä.

Tutkimustyön aikana kävi selväksi, että suurin osa Androidin applikaatioista sisälsi vakavia tietoturvariskejä, myös usein luotetut ja turvallisina pidetyt applikaatiot, kuten jotkut mobiilipankit saattavat sisältää useita ja vakavia tietoturvariskejä.

Suurin osa tunnetuista ja yleisimmistä tietoturva ongelmista ei ollut vain Androidille ominaisia ongelmia, vaan yleisiä tietoturvaongelmia, jotka koskettavat myös web-applikaatioita, Windows-sovelluksia, sekä muiden mobiilikäyttöjärjestelmien applikaatioita.

Myös Android-applikaatioille erityisiä tietoturvaongelmia löytyi, mutta ei mitään, mille ei olisi ollut olemassa helpohkoja ratkaisuja jo olemassa. Tästä voimme päätellä, että Android ei itse järjestelmänä ole turvaton, vaan kehittäjät eivät ole panostaneet applikaatioissaan tietoturvaan, vaan ennemminkin uusien ominaisuuksien kehittämiseen tietoturvan kustannuksella.

Avainsanat (asiasanat)

Kyberturvallisuus, Tietoturva, Mobiililaitteet, Ohjelmistokehitys

Muut tiedot

Contents

Figures

Terminology

Android system The Android Operating system is an open source OS designed mainly to mobile devices. Android is written in Java and it is based on Linux. Google purchased it from Android Inc. in 2005. (*Android OS*, N.d.)

Android Debugging Bridge Android Debug Bridge, abbreviated as ADB, is a client side command line tool for debugging and development purposes. It allows communicating with an emulator or with a connected Android device. (*Android Debug Bridge*, N.d.)

API Acronym from words Application Programming Interface. API is a program that manages interaction with other programs. API allows a developer to communicate the underlying application with callable functions. (*Application Programming Interface (API)*. N.d.)

Attacker Synonym for hacker and adversary. Person or organization who exploits weaknesses in the application in order to achieve control over the system for personal gain. (*Hacker*. N.d.)

Attack Synonym for Cyber-attack. Exploitation of computer systems, mobile applications and networks. Attacks are often executed by injecting malicious code to alter program logic to leak data or gain control over the computer system. (*Cyberattack*. N.d.)

Authentication The process that confirms and ensures user's identity to the system. (*Authentication*, N.d.)

Authorization The process that allows or denies user's access to the system, server or web API. (*Authorization*, N.d.)

Certificate	a Digital Certificates are used to confirm identity of a sender of an encrypted message to a client who then knows that the message came from a trusted source. (*Digital Certificate*, N.d.)
Certificate Authority	A certificate authority, abbreviated as CA, is a trusted entity that issues Digital Certificates that verifies the certificate holders identity. (*Certificate Authority*, N.d.)
Client	Refers to a person or organization who ordered the developed application or service.
Credentials	Credentials are proof of identity that is used to authorize access to the system and to authenticate the user. For example, credentials could be username and password or fingerprint of the user. (*Credentials*, N.d.)
Denial-Of-Service	A Denial-Of-Service attack, abbreviated as DoS, is a type of an attack where attacker attempts to prevent users from accessing the service. (*Denial-of-Service Attack*, N.d.)
Developer	A developer is an individual, who designs, builds and creates the application. (*Developer*, N.d.)
Device	A Mobile device is a handheld device designed to be portable, lightweight and small in size. (*Mobile Device*, N.d.)
Emulator	A Software that is used to emulate/simulate a mobile device. It is meant for developers to test their applications without fuzz of transferring them and installing them in to a physical device. (*Mobile Emulator*, N.d.)
Encryption	The process of converting data into an unreadable form and making it impossible to read for unauthorized users. (*Encryption*, N.d.)
HTTP / HTTPS	Hypertext Transfer Protocol, abbreviated as HTTP, is a protocol used to communicate in World Wide Web. Its secure

	version is known as Hypertext Transport Protocol Secure, HTTPS. (*Hypertext Transport Protocol Secure*, N.d.; *Hypertext Transport Protocol*, N.d.)
Man-In-The-Middle	A Man-In-The-Middle attack, abbreviated as MITM, is an attack where attacker gets between communications of two parties. Attacker could then be eavesdropping communication or modify data that is being sent through. (*Man-in-the-Middle Attack*, N.d.*)*
Malicious application	A Mobile Malware is malicious software designed specifically to mobile devices, such as tables and smartphones. Malicious application can exploit security flaws in the operating system or in other application installed in to the contaminated device. (*Mobile Malware*, N.d.)
Obfuscation	The process of obscuring the program code to prevent reverse engineering and copying the source code illegally. (*Obfuscation,* N.d.)
Phishing	Phishing attack is an act of trying to get user password or user's other sensitive information by masquerading as trusted website or application, or by sending lure emails. (*Phishing*, N.d.)
Root	Root, also known as Superuser, is the administrator and the most privileged account in the Android system. Root account can only be accessed by rooting the device. (*Superuser*, N.d.)
Signing	Code Signing is a technology of verifying the authenticity of an application publisher in order to avoid installing malware masqueraded as the legitimate application. (*Code Signing*, N.d.)
TLS	Transport Layer Security, abbreviated TLS, is a cryptographic protocol used to provide secure communication

between applications over the internet. TLS is the successor of Secure Socket Layer, SSL protocol. (*Transport Layer Security, N.d.*)

Token Access Token is an object that is used to encapsulate the user's security credentials and other user information to authorize the use of the system, server or application. Access token cannot be used to authenticate user, it is done with the usage of Authentication Token, which encapsulates user information to authenticate the user in the system, server or application.

1 Introduction

1.1 Security of mobile applications

According to NowSecure's mobile security report (*2016 NowSecure mobilesecurity report*, 2016),

- The number of mobile devices on Earth has surpassed the number of people living on it
- Forrester predicted people would download more than 226 billion apps in 2015
- 24.7 percent of mobile apps include at least one high risk security flaw
- The average device connects to 160 unique IP addresses every day
- 35 percent of communications sent by mobile devices are unencrypted
- Business apps are three times more likely to leak login credentials than the average app
- 8 out of every 10 phones in the world use the Android operating system
- Android currently has an estimated 1.6 million apps available on Google Play
- Only 43.8 percent of Android users have adopted Android Lollipop according to NowSecure mobile security intelligence

As it can be seen from the numbers above, the security of mobile applications is not very good even when the number of released mobile applications is growing fast.

Mobile applications can nowadays do many tasks compared to the times of Nokia 3310 and its Snake game. Mobile phones can do so much more than just make calls and send SMS messages: both personal and work email can be synced to the device, and there are several instant messaging applications to keep in touch with friends; some of those applications are able to offer voice and video calls over the internet. Mobile devices can be now used to shop online and pay bills as well as do bank transfers. Some applications and devices even allow users to pay contactless payments in shops or allow phones to be used as keys.

If the application security is not good enough, an adversary could exploit the application's weaknesses to achieve control over those matters: to read user's private, and work messages and emails, eavesdrop their internet behavior, phish credit card information or user credentials to various websites and services, or even worse, to steal money directly from user's bank account.

1.2 Digia Finland Oy

Digi Finland Oy is the customer organization for this work. Digia Finland Oy is the Finnish branch of multinational Digia Oy operating in several countries, including Sweden, Norway, Germany, Russia, China, South Korea and the United States. Digia has several domains working in very different areas, including commercial, logistics, and industrial sectors, the public sector, banking and insurance. This thesis was assigned by Digia Financial Solutions operating in the banking and insurance fields.

Digia Financial solutions has focused on offering wide area of solutions for financial sector; including banks, asset managers, mutual fund companies and other investment management companies and institutions. Normally these solutions are systems of grand scale used to run core processes of the client company.

As mobile devices and applications are evolving, so are their possible usages. In the future, some of these offered solutions might contain mobile applications that could be used to automate simple processes. Because of the nature of the financial sector, these applications and devices must then be very secure and reliable.

1.3 Assignment and objectives

Because cyber security plays an important role in the financial sector's solutions, the assignment was to study the possible weaknesses in application development. As there are several mobile platforms, such as IOS and Windows phone, the focus was decided to be placed on the most popular operating system, Android OS.

Applications fall into three main types,

1. Native applications launched from an icon in the start menu.
2. Web applications that are just websites and accessed through the internet browser in the device.
3. Hybrid applications which are like Native applications, however, do contain web elements through WebView component.

In this work the focus is on Native applications, leaving Webview out of the scope of the thesis.

The idea was to keep the view in application development for an enterprise, making security standards higher than in small games or utility applications, like torchlight

application or Tic-Tac-Toe game. Additionally, the idea was to solve mobile specific issues rather than universal problems in application development. For example, in a web application it is normal to type username and password to the application to start a new session, however, in the mobile environment this would be a nuisance.

2 Application Security

2.1 Creating secure connection to the server

According to OWASP Mobile top 10 vulnerabilities (2015), insecure Transport Layer Protection (TLS) transmission is the third most common vulnerability in mobile applications; the attacker might exploit vulnerabilities to intercept sensitive data while it is traveling across the wire, which might expose an individual user's data and can lead to an account theft. If the attacker intercepts an administrative account, the entire site could be exposed. Poor SSL setup can also facilitate phishing and man-in-the-middle attacks. (*Mobile top 10 vulnerabilities*, 2015)

2.1.1 TLS protocols and ciphers

There are several versions of SSL and TLS protocols, however, only two versions should be used, TLS 1.1 and TLS 1.2. The other versions, SSL 2.0, 3.0 and TLS 1.0 suffer from several vulnerabilities, such as Poodle, Crime, Beast and CBC; so those versions are no longer safe to use. (Green, 2015; *PCI Security Standards Council*, 2015)

Unsafe protocols can be disabled from connection negotiations by creating a custom SSL Socket Factory class in the application. Note that TLS 1.1 and 1.2 are not in Android API versions prior 16, Android 4.1, Jelly Bean. (*SSLSocket*, N.d.)

Unsafe or weak ciphers should be removed from the application to ensure safety of encrypted data. Even in Android 5.0 there are some unsafe ciphers enabled by default, like TLS_RSA_WITH_RC4_128_MD5, which uses outdated MD5 cryptography. (*User agent capabilities: Android 5.0.0*, N.d)

It would be most optimal from the security perspective to use ciphers that use Forward Secrecy. In most common key exchange mechanisms, RSA session keys are created from server's private key. Should a server's private key fall into hands of an attacker, it could decrypt not only all future communication but also all encrypted data the attacker has gathered before obtaining the server's private key.

Instead of RSA-based key exchange, there is ephemeral Diffie-Hellman algorithm, which is slower and generates session keys in such a way that only the two parties involved in the communication can obtain them; even with the access to server's private key. After the session is complete, and both parties destroy the session keys, the only way to decrypt the communication is to break the session keys themselves. This protocol feature is known as forward secrecy. (Ristic, 2013)

Breaking session keys is clearly much more difficult than obtaining the server's private key. Now the attackers can no longer obtain just one key to decrypt communications but they have to compromise the session keys belonging to every individual conversation. (Ristic, 2013)

SSL supports forward secrecy using two algorithms, the Diffie-Hellman, from now on DHE, and the adapted version for use with Elliptic Curve cryptography, from now on ECDHE; however, there are two problems in using them. DHE is significantly slower than common RSA-based algorithms. ECDHE is slightly faster than DHE, yet still much slower than RSA. Also, both cipher types are quite new and the older Android versions will not support them. DHE is supported on Android 2.3 Gingerbread, API level 9+ and ECDHEs are supported on, Android 4.4.4 Kitkat, API level 20+. (*SSLSocket,* N.d.; Ristic, 2013; Bernat, 2011)

Disabling and enabling used ciphers requires the creation of custom SSLSocketFactory.

Note that some libraries, e.g. third-party analytics companies and social network addons might use their SSL versions to establish connections when the application runs, causing mixed SSL sessions and thus might expose the user's session ID. (*Transport Layer Protection Cheat Sheet*, N.d.)

2.1.2 Certificates

SSL Certificates are used in TLS protocol to encrypt transferred data, provide authentication of the server to the client and to ensure data integrity. Anyone can create own key pair, and then have it signed by a Certificate Authority, often shortened as CA.

Certificate Authorities are trusted entities that issue certificates. A person who wants their server to be trusted by the Android system needs to have their public key signed by a CA. Even if self-signed certificates would be just as good to encrypt the data, the end-user device would not know if the server's certificate would indeed be the real one or not, thus causing vulnerability to a man-in-the-middle attack.

From SSLShopper site, article "*Why SSL? The Purpose of using SSL Certificates*" (N.d.):

> *The biggest problem with a self-signed certificate is a man-in-the-middle attack. Even if you are 100% sure that you are on the correct website and you completely trust the site, you could have someone intercept the connection and present you with their own self-signed certificate. You would think that you are using a secure connection with your server but you are really using a secure connection to an attacker's server.*

Android system has a build-in list of trusted CAs' root certificates. If, for some reason developer would not want to use a trusted CA to sign their key, they could create their own CA and import its root certificate to the device. Of course, this would be a huge effort if the application is deployed to several users, however, this can be avoided if Public Key Pinning is used, see chapter 2.1.3 for more details about public key pinning. (*When are self-signed certificates acceptable?*, N.d.)

Sometimes an Android system might not recognize a certificate returned from the server, which might be due to the certificate being self-signed, or the server is not returning the whole certificate chain, thus the Android system misses the intermediate CA returned from the server. (*Security Tips*, N.d.)

According to official Android developer site (*Security with HTTPS and SSL*, N.d.), most public CAs do not sign server certificates directly. Instead, they use their main CA certificate, referred to as the root CA, to sign intermediate CAs. They do this so the root CA can be stored offline to reduce the risk of compromise. However, operating systems like Android typically trust only root CAs directly, which leaves a short gap of trust between the server certificate—signed by the intermediate CA—and the certificate verifier knowing the root CA. To solve this, the server does not send the client only its certificate during the SSL handshake but a chain of certificates from the server CA through any intermediates necessary to reach a trusted root CA.

If the server does not provide a full certificate chain to the application, it should not be trusted.

2.1.3 Public key pinning

Because trusted CAs has been targets of security breaches and they have issued certificates for unqualified names, such as “localhost” and “webmail”, their trustworthiness has been compromised. In security breaches attackers managed to issue fraudulent certificates to several sites, like Windows Update server and Gmail. These forged and misleading certificates could be used to spread malware and eavesdrop users. This kind of nationwide man-in-the-middle attack against Google Gmail users was uncovered in Iran after Google started using public key pinning in Google Chrome web browser. (Schoen, and Galperin, 2011; Elenkov, 2012)

Normally in SSL connection, the client only checks that the server’s certificate has a verifiable chain of certificates and it matches the hostname, however, it does not check if the certificate is indeed the one that the developer installed into the server. In application development the developer most likely knows the host and therefore can make sure that the certificate that the server returns is the original one. (*Certificate and Public Key Pinning*, N.d.)

The application can withhold public key hashes of the expected certificate and then parse the public key from the certificate and compare the hashes; if they are equal the connection can continue, otherwise it should be dropped. Pinned certificate’s public key could be from the server’s certificate, or if their own CA is used, it could be its public key. (*Pinning Cheat Sheet*, N.d.)

Pinned certificates should be saved in a secure location to avoid attackers from extracting them from the device in case of losing the device or reverse engineering the application’s installation package.

2.1.4 Client-side certificates

Client side certificates can be used with TLS to prove the identity of the client to the server. This method requires the client to provide their certificate to the server, in addition to the server providing theirs to the client.

Even when client certificates bring an additional layer of security, this has several problems, such as certificate generation, safe distribution of the certificate, client side configuration, certificate revocation and reissuance, and clients can only authenticate to servers with the client's certificate installed. Due to the sheer complexity of this implementation, the client side certificates should only be used in high-security applications with small volume of users. (*Transport Layer Protection Cheat Sheet*, N.d.)

For instance, if client-side certificate is preinstalled in the application's keystore and it has predefined password access, an attacker could reverse engineer the keystore's password and so obtain the client-side ceritificate. For each client to have a different certificate, physical installation of the certificate would probably be needed to be sure that it is not compromised during the transmission to the device.

2.2 Authentication and authorization to the backend

2.2.1 OAuth 2.0 with OpenID Connect

In a normal web application, users are requested their credentials every time they access the service. This works just fine in applications that are not used often or are used only in a single session, however, in mobile environment it is very inconvenient to the users to submit credentials every time they want to access the service. (Bray, 2013)

In HTTP Basic Authentication, API key must be presented in every call, and Google recommends (*Security Tips*. N.d.), that user password should not be saved in the device and API key should not been directly implemented into application. (See Chapter 2.2 Obfuscation) OAuth 2.0 protocol does not require saving API keys into an unsafe environment. Instead, it will generate access tokens that can be stored in an untrusted environment temporarily. (Degges, 2015; Godfrey, David, Raghav & Onur, 2014, 181)

Simplified OAuth workflow is explained below. For more detailed information, see Appendix 1.

1. User opens the app.

    a. App checks if token exists and it is valid.
2. App asks credentials from the user.
    a. App needs to ask the user for their username and password to retrieve first token.
3. App sends requests to the API service.
4. API Server authenticates the user.
    a. Service validates retrieved username and password.
5. API server generates a token and returns it to the app.
6. App saves the token to a secure place, like in Shared Preferences or in Account manager.
7. App Makes Authenticated Requests to API server using retrieved token.

As OAuth is just an authorization protocol, it needs some additional elements to actually authenticate the user. According to OAuth site itself (*User Authentication with OAuth 2.0*. N.d.), user should never authenticate using only OAuth protocol, as it is not sufficient. For instance, Facebook used to use wrongly implemented plain OAuth to authenticate its users and it was found to be vulnerable to “Covert Redirect”-attack. (Goldshlager, 2013)

To authenticate the user securely when using OAuth 2.0, OpenID Connect must be implemented on top of that. OpenID Connect is an authentication layer that operates on top of OAuth 2.0. Too see how OpenID Connect changes the workflow of OAuth 2.0, see Figure 1. (*User Authentication with OAuth 2.0.* N.d.; *Welcome to OpenID Connect*. N.d.; Sakimura, NRI, Bradley, Ping Identity, Jones, Microsoft, de Medeiros, Google, Mortimore, & Salesforce. 2014.)

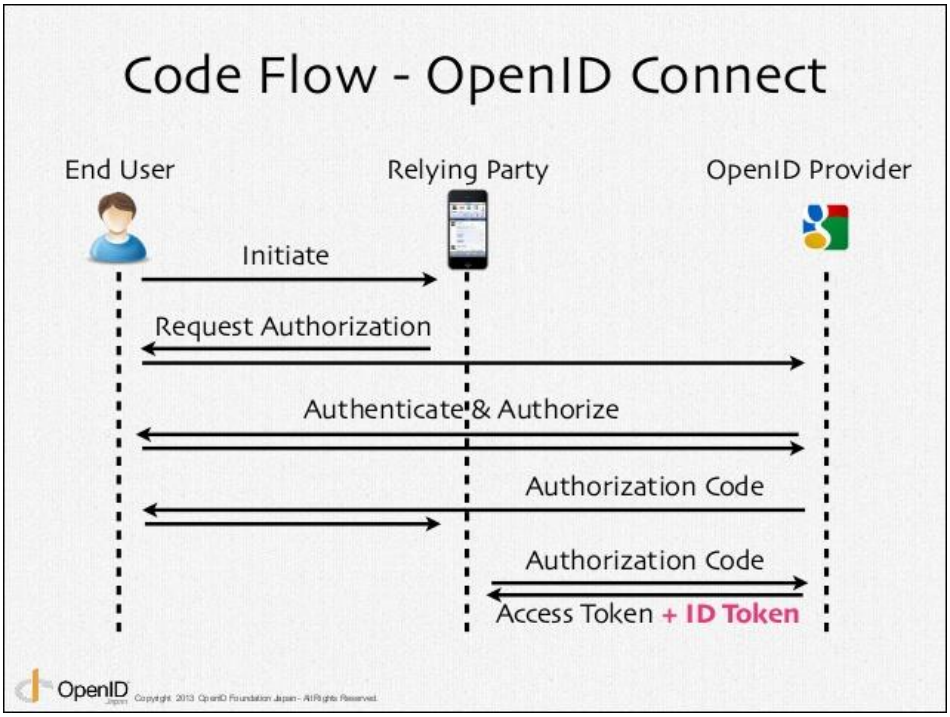


Figure 1. Workflow of OAuth 2.0 with OpenID Connect where Google acts as OpenID Provider (adapted from Matake, 2014, 31)

Benefits of using OAuth 2.0 with OpenID connect are but are not limited to the fact that user's password is not saved to the device, and thus it cannot be extracted from there; changing the user's password would not deprecate all the user's saved credentials in every application that user has given their credentials, and user does not need to submit their username and password every time they need to access the application, making phishing attacks a lot harder. (*Security Tips*. N.d.)

2.2.2 Using Google Sign-in

Google Sign-In is a secure authentication system that uses Google accounts, the very same account that user already uses to access Google services, like Google play, to securely authenticate with application's back-end server. Google Sign-in can be integrated with the application and its backend to quickly authenticate users, to easily manage accounts and to integrate Google services, e.g. access user's calendar and contacts. (*Google Identity Platform*. N.d.)

Google sign-in is based on OAuth 2.0 and OpenID Connect; (See chapter 2.2.1 OAuth 2.0 with OpenID Connect) where Google's servers acts as authorization endpoints.

When user taps the sign-in button in the application, user is prompted to choose an account to use with the application and give application permission to use it for authentication purposes; then the application only needs to send the ID token through HTTPS to the server, where the server then validates and verifies the token using Google API Client Libraries. See Figure 2 for more detailed view in this workflow. (Moroney, 2016; *Authenticate with a backend server*. N.d.)

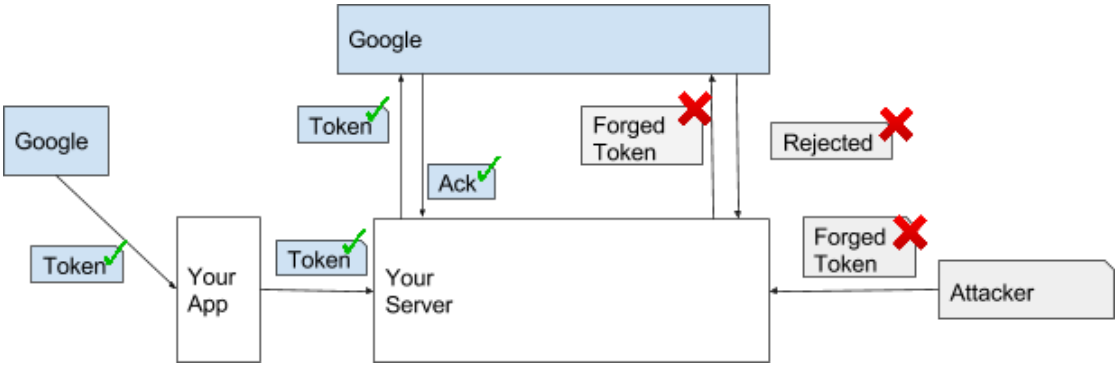


Figure 2. Google Sign-In Workflow (adapted from Moroney, L. 2016)

As seen above, Google Sign-in is a very simple and effortless way to implement authentication to the server and application, however, it requires user to have a Google account and developer to trust Google's services.

2.2.3 Using Smart lock for passwords

Google offers to users the possibility to save their login credentials to other sites and applications in their Google accounts, the idea being that Google account would act as a password vault for user's passwords. Developers can use this feature to automatically login users in to their applications and websites to achieve an effortless and fast login experience for users. See Figure 3 to details of Smart Lock login experience. (*Smart Lock for passwords on Android.* N.d.)

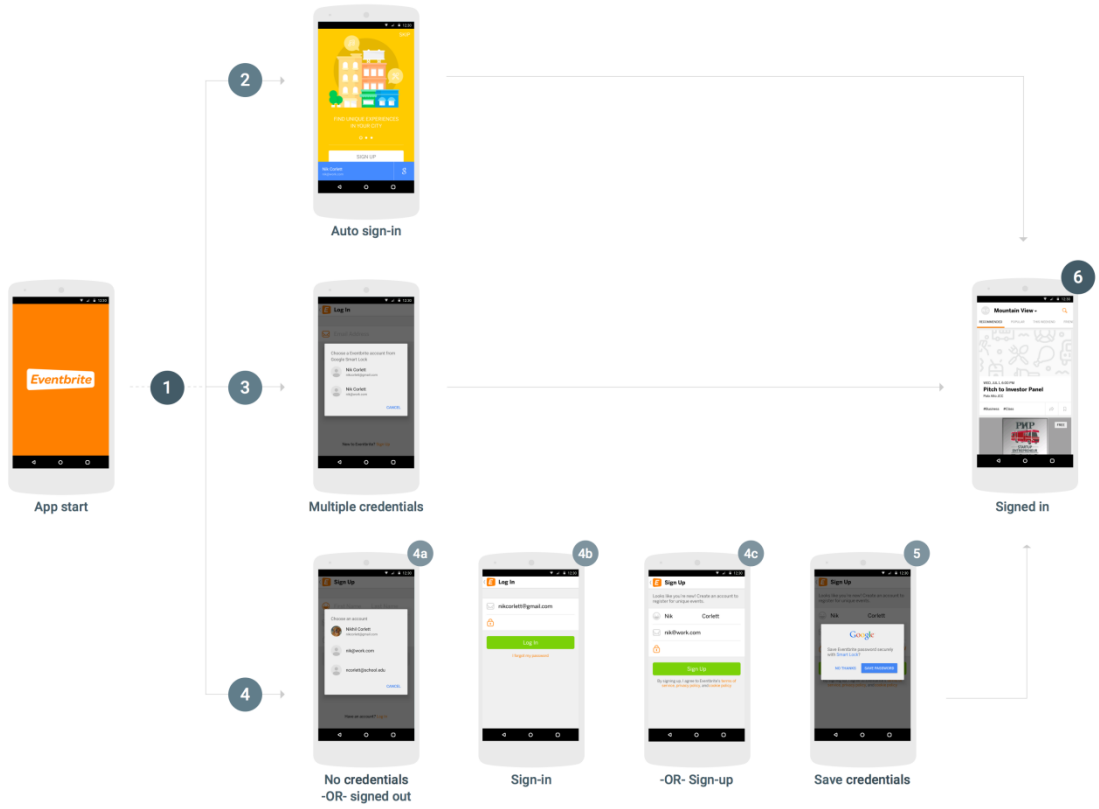


Figure 3. Smart lock for passwords on Android workflow. (Adapted from *Smart Lock for passwords on Android.* N.d.)

Users can manage the saved credentials in their Google account site and see their saved passwords from there, which causes a security issue should the user lose their account to an attacker that would allow the attacker to steal, not only user's Google account, but also all accounts linked to it. (Wallen, 2016.)

Google does provide its users with multiple security features to protect their accounts, like two factor authentication but it is up to the user to get them in use. Google's smart lock for passwords is an excellent solution to small scale applications where fast access is required and no financial loss could happen. (Wallen, 2016.)

2.2.4 Using Identity Toolkit

Google identity toolkit works in a similar way than Google Sign-In; (See chapter 2.2.2) it also uses OAuth 2.0 and OpenID Connect: it allows developer to choose a set of identity providers to authenticate the user, and to users a login with email, in case if

user does not have account in any offered identity provider, or does not trust them. (*Google Identity Platform*, N.d.; *Account Chooser*. N.d.; *Sign-in flow.* N.d.)

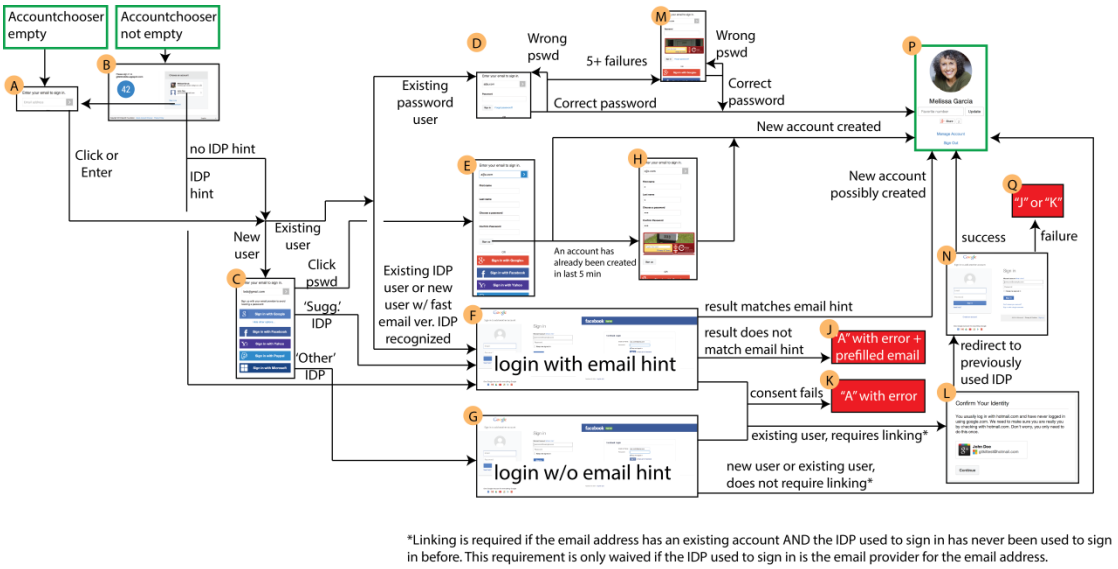


Figure 4. Identity toolkit workflow. (Adapted from *Sign-in flow.* N.d.)

As seen in the figure above (Figure 4), the account chooser workflow is very user friendly and supports several handy features from the developer's perspective, such as account migration, in case user logins in with another identity provider after the first login. (*Google Identity Platform*. N.d)

Usage of the Google identity toolkit has the same benefits as using Google sign-in and plain OAuth, 2.0 but has also to offer different identity providers than Google and developer's own. Using the identity toolkit is probably the most suitable of wide-spread applications in public that does not require bank like security, due to its easy implementation and fast sign-in/sign-up flow in user's perspective.

2.2.5 Using custom authenticator

Most OAuth 2.0 and OpenID Connect solutions require using external authentication- and authorization-endpoints. If the developed application was a high security application it would be undesirable to trust external companies and services to authenticate users. For example, who would like to use mobile banking application with Facebook login? (Cohen, 2013.)

If developer decides to create their own implementation of authenticator class in the application, then developer needs to make sure to implement all functionalities to offer same kind of seamless login and token verification as other offered solutions do, like in Google's Identity toolkit. (Cohen, 2013.)

2.3 Secure data storage

2.3.1 Shared preferences

Shared preferences are application specific XML files in the application's data directory and they consist of a list of name-value pairs. Shared preferences can be used to save small amounts of primitive data, Booleans, floats, integers, longs, and strings. Data in the file will persist across user sessions; even when the application is killed or the device is rebooted. Typically user preferences and high scores from games are saved in the Shared preferences and sometimes even user credentials. (*Storage Options*, N.d; *Insecure Local Storage: Shared Preferences,* N.d.)

Shared preferences are protected with UID permissions, i.e. applications with the same UID are only able to access those XML files. Malicious applications can only access those files it they are granted root level permissions on a rooted device. (Six, P. 2012, 21-22)

As Shared preferences are just XML files in the Android file system, an attacker can access those files upon physical access to the device. The file could be extracted from the device using Android Debugging Bridge, from now on ADB, using its backup functionality. Or if the device is rooted, files can be retrieved by just transferring them using the ADB. (Xiao & Olson, 2014; *Insecure Local Storage: Shared Preferences,* N.d.)

As stealing data from the Shared preferences would require root level permissions on a malicious application or physical access to the device, developer can be sure that Shared Preferences is a relatively safe place to save user data, but should still keep in mind that it is not fool proof and user's password still should not be saved there, see chapter 2.2 "Authentication and authorization to the backend" for more information about this topic. (Six, P. 2012, 21-22; Xiao & Olson, 2014; *Insecure Local Storage: Shared Preferences,* N.d; *Storage Options*, N.d)

2.3.2 Internal storage

Android system offers the possibility to applications to save files directly in the device's internal storage. These saved files are only accessible to the original application via UID permissions and inaccessible to the user through file system. Upon uninstallation of the application, all files saved in the application's internal storage will be deleted, so developer should make sure not to save any files belonging to the user there. (*Storage Options*, N.d.)

As described in chapter 2.4.1 "Shared Preferences", also internal storage files can be viewed by a malicious application with root level permissions or with physical access to the device by using ADB. (*Insecure Local Storage*, N.d.; Six, P. 2012, 21-22)

Developer can also share application's data saved in the Internal storage by using file modes, MODE_WORLD_WRITEABLE and MODE_WORLD_READABLE. According to Google (*Security Tips*, N.d.), this should not be used because they do not provide the ability to limit data access to particular applications, nor do they provide any control on data format. Instead, developers should be using Content Providers, which offer read and write permissions to other applications and can make dynamic permission grants on a case-by-case basis.

2.3.3 External storage

External storage is the removable media in the device, like SD cards, however, it can also be a partition in the built-in memory. All data written to the external storage is world readable, meaning that any application in the device can access them and user can view and edit saved data when user connects the device to a computer and enables USB mass storage to transfer files on a computer. This is because Linux permission based access control is not present in the External storage, as it is normally formatted using incompatible file system. (*Storage Options,* N.d; Six, P. 2012, 21-22)

All data saved in the External storage is public, so the application should not save any confidential data there, nor should it trust any data read from the External storage, as any application could insert malicious data there. If application writes to the External storage and it is needs to be sure that no other application could read or modify

those files, then encryption and checksums should be used to verify data integrity. (Six, P. 2012, 21-22; *Security Tips*, N.d.)

2.3.4 SQLite databases

Android system offers applications possibility to create application specific SQLite databases to save repetitive or structured data. By default, created SQLite databases are only accessible to the application that created them because also SQLite databases are protected by the UID permissions. (*Saving Data in SQL Databases.* N.d; Six, P. 2012, 23.)

Just like the Shared preferences and Internal storage files, also SQLite databases can be created with MODE_WORLD_WRITEABLE and MODE_WORLD_READABLE modes but as explained in chapter 2.4.2 “Internal storage”, this should be avoided and Content Providers should be used instead to share the data. (Six, P. 2012, 23; *Security Tips*, N.d.)

SQLite database files are saved in the application’s data folder and are also extractable with the usage of the ADB and viewable to a malicous application with root level permissions. (*Insecure Local Storage*, N.d.; Six, P. 2012, 21-22)

To avoid possible SQL injections, queries to the database should always be validated, and usage of the prepared statements is encouraged, especially if other applications are allowed to query the database thorugh a Content Provider. (*Security Tips*, N.d; *Mobile Top 10 2014-M7*, 2014)

2.3.5 Application logging

Android applications normally log application events using the Log class, and its output can be viewed with Logcat tool. Developers can use logging during the development phase of the application to debug it but might forget to disable it upon application release, or they might fail to understand why it should be disabled; or if logging is required, the developer could forget from limiting what is being logged. (*Reading and Writing Logs*. N.d.)

In 2014 Sanchez, (2014) researched black box and static analysis to worldwide mobile home banking applications. Most of the log files from tested applications logged sensitive information, like user credentials.

Prior Android version 4.1 Jelly Bean, Android API level 16, it was possible for any application with READ_LOGS permission to read logs of any application in the device. Even though this is now fixed, devices that will never get the needed update, or malicious applications with root level permissions could still read those logs. Also with physical access to the device, it is possible to extract and read all log files from the device using Eclipse IDE or ADB. (*Exploiting Unintended Data Leakage (Side Channel Data Leakage),* N.d.)

2.4 IPC Security

2.4.1 IPC endpoints

Android's inter-process communication, from now on IPC, is a mechanism that allows communication between Android services, Activities, Broadcast Receivers and Content Providers. Each Android component can be either public or private. If the component is public, other components can interact with it. If it is private, the only components that can interact with it are those that are part of the same app, e.g. ones that run with the same UID. (Six, 2012; Drake, Fora, Lanier, Mulliner, Ridley & Wicherski, 2014, 89; *Security tips*, N.d.)

Consequences of improper security of the IPC endpoints vary depending on the type of the endpoint. For example, Content Providers could be vulnerable to data injection and data leakage and Activities could be vulnerable to the UI redressing attack. (Drake, J. 2014, 89)

For example, André Moulu found vulnerable IPC endpoint in Samsung Kies software. Kies had "INSTALL_PACKAGES" permission and an unprotected Broadcast Receiver with registered action "KIES_START_RESTORE_APK". Calling this with Intent would cause the broadcast receiver to start KiesService with the data from the Intent and the service then tries to install all APK packages from /sdcard/restore/ directory. He then proceeded to save his own application in that directory by exploiting another

application which had “WRITE_EXTERNAL_STORAGE” permission and unprotected service that listened for an Intent “CLIPBOARD_SAVE_SERVICE”. Broadcasting that type of Intent allowed him to copy an APK package to the SD card. (Moulu, 2012; Drake, 2014, 90-91)

2.4.2 Android permission system

Accesses to the IPC components are restricted with the usage of Android permission system. Developer can create and implement custom permissions in developed components, and the calling application must have these permissions in order for call to succeed. The application can check required permissions either programmatically or by declaring them in application’s manifest file. (Six, 2012; *Permission*, N.d.)

There are four different levels of android permissions:

1. **Normal**. Permissions that cannot do any real harm to the user. Such as changing wallpaper. Application must specifically request for these permissions but they are automatically granted. User can still view these permissions during application installation. (*Permission*. N.d.)

2. **Dangerous**. Permissions that can cause real harm and loss of money to the user. Such as send SMS and open internet connections. Applications must request for these permissions and user must accept them. (*Permission*. N.d.)

3. **Signature**. Custom permissions that are automatically granted for the requesting application if it signed with the same certificate as the application that created the permission. These are used to share data between related applications. (*Permission*. N.d.)

4. **Signature or System**. Permissions that are granted only to applications that are in the Android system Image or that are signed with the s me certificate as the application that created the permission. According to Android Developer site, these permissions should be only used in special situations where multiple vendors have applications built into a system image and need to share specific features explicitly because they are being built together. (*Permission*. N.d.)

Applications that use Signature-type permissions should not rely completely on the permission because Android system keeps track of installed permissions only by the name, so if a malicious application is installed with permission of same name as in a legitimate app, the legitimate application would use malicious permission that was declared by the malicious application. (Weichao, 2014.)

According to Weichao, (2014) from the Trend micro security, developers should not rely exclusively on the protection levels when their IPC endpoints are accessed. Several functions, such as getCallingUid and getCallingPackage are provided by the operating system and can be used to identify any applications making the call and implement access control as needed.

While installing the application, a list of required permissions is shown to the user who then either accepts them, or the application is not installed. From Android 6.0 or higher, and app's target SDK is 23 or higher, applications must specifically request dangerous permission while the application is running. Also from Android 6.0, API level 23, users can revoke permissions from any app at any time, even if the app targets a lower API level. In the name of reliability, the application should test if it still has permission for an action before executing it because user might have revoked the needed permission. (*Requesting Permissions at Run Time*, N.d.)

2.4.3 Intents

Intent is a messaging object that is used in IPC communication to request an action from another component. Intents are mainly used for three things, with Activities to show a screen, with Services to do a single operation, e.g. to execute a file download and to broadcast a message to other applications. (*Intents and Intent Filters*, N.d.)

There are two types of Intents, Explicit and Implicit Intents. Explicit intents specify the component to start by name and are therefore often used to activate a component in the same application. Implicit intents declare a general action to perform, which allows a component from another application to handle it. (*Intents and Intent Filters*, N.d.)

In the figure below (Figure 5), when Implicit Intent is launched by Activity A (1), the Android systems finds matching Intent Filters from installed applications' manifest files (2). If a match is found, the system starts that component and passes the Intent to it (3). If multiple components are found, then user is shown a dialogue to choose the used application. (*Intents and Intent Filters*, N.d.)

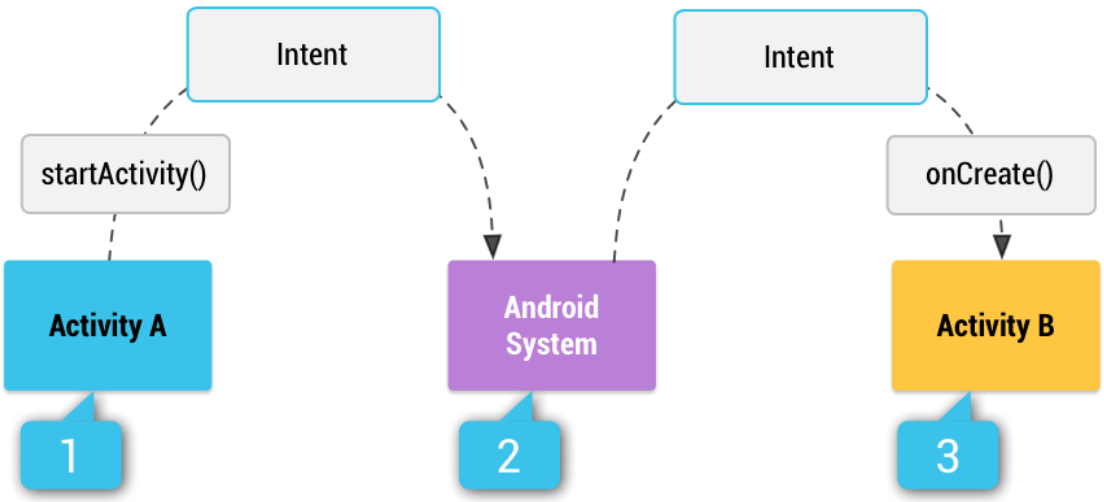


Figure 5. Implicit Intent flow. (Adapted from *Intents and Intent Filters*, N.d.)

2.4.4 Intent Hijacking

Because Implicit Intents are broadcast to the whole system, they are prone to be hijacked. Access to Intents can be restricted with Android permissions and Intent Filters. If “Signature” or “SignatureOrSystem” permissions are not used to restrict access to the launched Intent, then any application with matching Intent Filter could access it, and even worse if more accurate Intent Filter is used by the malicious application, choosing dialog is not shown to the user. Even if choosing dialogue is shown to the user, user might accidentally choose a malicious application to be the default application for that type of Intents. (Security Tips, N.d.; *Intents and Intent Filters*, N.d.; Medianero, D. 2016)

Malicious applications could “consume” the broadcast Intent in order to prevent legitimate applications from getting it to execute denial-of-service attack, execute man-in-the-middle attack or to show malicious Activity for phishing user credentials. (*Security Tips*, N.d.; *Intents and Intent Filters*, N.d.; Medianero, D. 2016)

According to Google, (*Intents and Intent Filters*, N.d.), Services should only be started with Explicit Intents and no Intent Filters should be declared for them. It cannot be known what service is started with an Implicit Intent and user cannot see which service is started. From Android 5.0, API level 21, the system will throw an exception if a service is called with Implicit Intent.

As described in Chapter 2.5.2 Android permission system, a malicious application could still be installed, and granted permissions to the Intent.

2.4.5 Intent Spoofing

An installed malicious application could be broadcasting forged Implicit Intents to the Android system in order to cause legitimate applications to show Activities and start Services with malicious intentions; or simply to crash applications, causing denial-of-service. Consequences of accepting forged Intents varies greatly between applications and are often very different, as shown in the example in Chapter 2.5.1. (Medianero, 2016.)

To protect an application from forged Intents, the developer should think if that Intent does indeed need to be implicit, or if possible, set custom “Signed” permission, and validate data from the Intent carefully. (*Intent Spoofing Vulnerability in Android Apps – OWASP Top 10*, 2015)

2.4.6 Content Providers

Content Providers are used to sharing private data between processes and applications in the Android system. Content provider also acts as an abstractor between its repository and the data. Just like other IPC endpoints, accesses to Content Providers are restricted using permissions. Content provider permissions have on top of normal permissions URI permissions that are used to limit access to the data in a specified URI. (*Content Provider Basics*. N.d.)

For example if an image attachment is received in a mail application that is protected with permissions and a user wants to view the received image with an image viewer application, the image viewing application should not get full permissions to the mail application as it contains private data, therefore the image application should only receive URI permission to view that one image in the Content Provider. (*Content Provider Basics*, N.d.; *System Permissions*, N.d.)

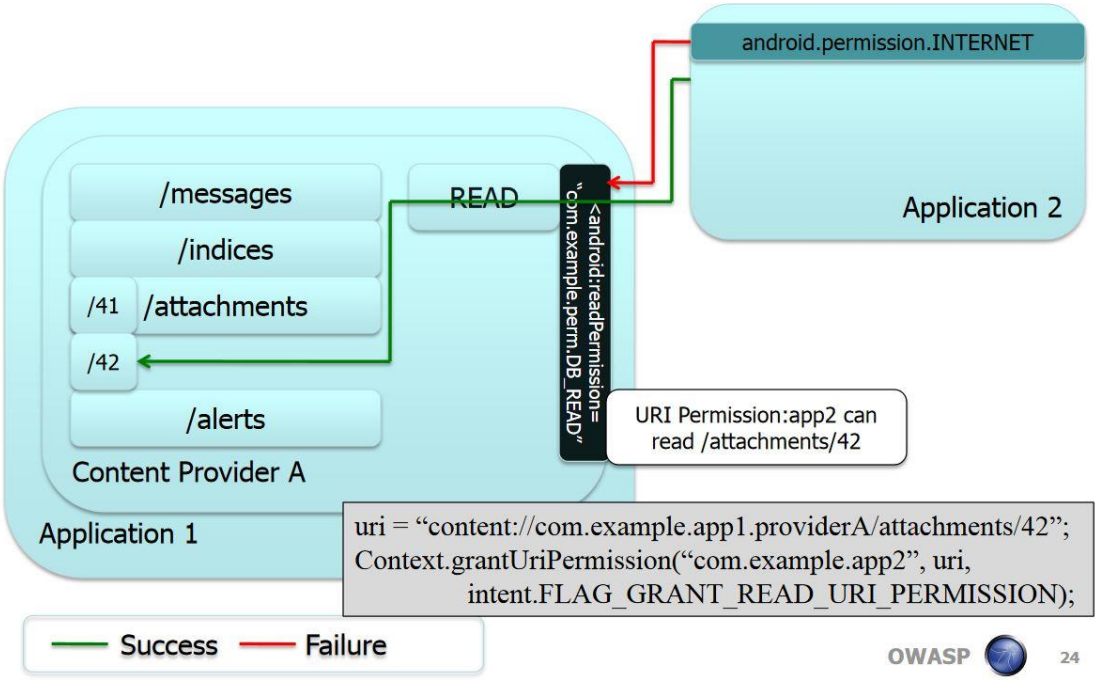


Figure 6. Example of URI permissions. (Adapted from Six, J. 2012.)

As seen in Figure 6, Application 2 has URI permission only to the /42 directory under the /attachments directory in Application 1 and therefore is only able to read the contents of that specific folder. If a Content Provider is for some reason made public, then it should not contain any sensitive data. (Six, 2012; Shòu, 2015)

If access to a Content Provider is not limited, and untrusted user input is not sanitized before execution, the Content Provider could be vulnerable to SQL injection attack. A malicious application could be inserting SQL queries that form a logical tautology, causing Content Provider to return the whole content of a table. Therefore parameterized SQL should always be used instead of raw SQL. By parameterizing queries developer does not only protect the Content Provider from SQL injection, but also increase its performance as the Content Provider does not need to parse every statement before execution. (Shòu, 2015; Makan, 2013)

3 Application management

3.1 Android Application Packages

It is not enough that an application is secure itself. Because of the nature of Android Application Packages, from now on APK, any code that is installed to an end-user's mobile device can be reverse engineered back to debuggable source code. Therefore, the developer must be sure that code itself does not contain anything the developer wants to keep as a secret, such as passwords, keys and certificates. Additionally, the company should also try to avoid that no one who does not actually need the software can access its installation packages. (Fora, P., 2014)

Because APKs can be easily reverse engineered back to the original source code, any application can be read, debugged and their operation understood. An attacker can use this to find new attack vectors to the software, create malicious version of the original application, or another company could create a copy for themselves to develop and sell, thus avoiding major development costs that the original developer had.

3.2 Source code safety

As mentioned before, code can be reverse engineered and easily modified. APK can be opened with very little effort with tools like APKtool. If any kind of obfuscation software were used when the package was built, all class and variable names are reversed back to original ones, making understanding of code much easier.

No matter how well application data is encrypted, data will still appear unencrypted in the memory. A rooted device can read memory from the device that contains data and possibly even decryption keys.

Attacker who runs an application in an emulator or uses a custom kernel can use tools such as Volatility and techniques such as loadable kernel modules to intercept data and keys used to encrypt the data. (Judge, N.d.)

3.2.1 Signing the package

Android system prevents users from installing unsigned applications, so all APK package must be signed with the developer's private key. Installation packages are signed using public-key cryptography, meaning that the original private key is impossible to extract from the signed package.

The security of the developer's private key is critical and it should never be kept in an unsecure or public place where several persons could access it. If the private key were to fall in the hands of an attacker, that person could sign and distribute apps that maliciously replace original apps or corrupt them. The attacker could also sign and distribute apps under developer's identity that attack other apps, the system itself, corrupt or steal user data. (*Signing Your Applications*, N.d.)

3.2.2 Obfuscation

Event though security through obscurity is generally a bad idea if used alone, it can still be used to make cracking the code harder by making it more time consuming and frustrating. Obfuscation will not stop motivated and dedicated attacker, however, it may stop persons with little motivation, time or interest. (*Avoid security by obscurity*, N.d.)

There are several tools to automatically obfuscate java code, and nowadays ProGuard tool is integrated in Android development studio, making obfuscating easy and fast. Tools will obfuscate code by removing unused code and renaming classes, fields, and methods with obscure names. See example of code obfuscated by ProGurad in Appedix 2. (*ProGuard*, N.d.)

According to Simon Judge (Use the Android NDK for security Sensitive code, N.d.), the most vital parts of an application, such as tamper detection could be written in Android NDK. NDK is a tool set that allows implementation of native-code languages such as C and C++. When the application is compiled with maximum optimization, the parts that were written in native-language are much harder to decompile and understand. (*Android NDK*, N.d.)

3.2.3 Tamper detection

By adding tampering detection to the application, the developer can try to prevent an attacker from making changes to the application's code and stop it from debugging the application in a emulator, or at least make it unbearably hard. The idea is to read device variables, application data and other environment variables to determine if the code is run in debug mode, emulator or if it has been altered.

Tamper detection should also be made in the server side code, so tamper detection code cannot be altered or removed so easily. (Alexander-Bown, N.d.)

Issues to check include, however, are not limited to:

1. Applications public key.
2. Install source.
3. Envrionment variables, eg. hardware.
4. Manifest file's debuggable field.
5. Root status.

By checking the public key, the developer can be sure that the application is the original one deployed, and by checking the install source from Android package manager it can be secured that it was installed from the original source.

Checking the device's environment variables and manifest file's debuggable flag the application can try to determine if it is run in an actual device and not in a emulator.

Of course, an attacker can try to find all tamper detection code, alter or remove it and so fool both the application and server, however, when combined with methods mentioned in the previous chapter, it will be extremely hard and time consuming.

3.3 Distribution of application and updates

The application and its updates must be distributed to the customers. Because a company does not want the application to be public, there is a need to choose the distribution and update channels and ensure their safety.

The default appstore in an Android system is Google Play, and by default it is the only trusted source that is allowed to install applications straight away. Depending on the used device, it might have other pre-configured appstores, such as Amazon market

and Yandex; however, installing from any other source than Google Play, the user needs to accept installing applications from “Unknown Sources”, which could allow installing harmful applications without user’s consent.

Customers have a huge impact on how applications should be deployed. If the customer company has a Mobilde Application Management Solution that allows distibuting applications and their updates, then of course that should be used, since using it is propably company policy and customer would not even allow any other way.

3.3.1 Google Play

Google play is the default app store in the Android system and the only trusted source allowed to install applications without “Unknown Sources”-setting on, which is a positive feature from the security point of view.

Google play offers effortless distribution for applications and its updates with features that allow the creation of private distribution channels. Users can be added and removed from private channels and after adding users into one, users can dowload applications that are published there. (*Developer Console Help*, N.d.)

While Google Play allows an easy distribution of applications to a private channel, there are no ways to uninstall applications from end-user’s device. After the user has been dropped out from the private channel, the installed application and its installation package cannot be uninstalled remotely. (*Google Apps Administrator Help,* N.d.)

3.3.2 Mobile application management solutions

While Google Play allows developers to restrict access to the application, customers might want to restrict its employees’ access to application stores in order to prevent them to accidentally install harmful or unproductive applications, or customers might not want to trust Google to keep their software, which is when Mobile Application Management (from now on MAM) tools step in.

MAM solutions can enable pushing applications and updates directly to the end-user or show an application in Google Play or in the company's own app store where users can install apps for themselves. This way the application and updates are easy to distribute and there is no way that users might install a wrong, possibly malicious app from a store. (Steele, N.d.; Rouse, N.d.)

Nevertheless, a customer company might not want to adopt MAM solutions for various reasons, such as cost of the service, required work to keep it running and employee's privacy concerns, if "Bring your own device"-policy has been applied.

3.3.3 Other methods

In some cases the customer company might not want to trust Google Play services and apply Mobile application management solution due the small scale of the project. Then the application and its updates must be distributed outside the app stores.

After allowing installation from "Unknown sources", users can install the application from anywhere, e.g. an email attachment or a company's website. While this is obviosly the easiest way to distribute apps, it also has some serious issues.

1. It is hard to keep track who has installed the application and it might be not easy to restrict access to installation file if it is distributed from a website.
2. No steady source to install updates. Google play and some MAM solutions can push updates to end-user's device, but when distributed from a website the users must check for update packages themselves or someone must email them.
3. Allowing installing from "unknown sources"setting must be on.
4. Effort to manually do all the things that other solutions would have already.
5. No way to uninstall application from users that no longer need it.

It is possible to create auto-update feature inside the application itself, where it would dowload the latest APKs from a server; however, this is undesirable because it would ask for an effort to create a code to securily dowload the updated APK and check automatically for updates, while all these services are already made in other solutions. (Strumpflohner, 2011.)

4 Conclusions

Even as mobile applications are insecure in general, it seems that Android system itself is not insecure but rather developers are more focused on creating new features rather than improving applications' security. Also, some degree of rushing and floppiness can be seen in security issues like unintended data leakage through application's log files.

Some categories, like Intents in IPC controls are something that most un-professional developers might not understand completely, and thus they cause security holes that way. Google has done a good job the in their recently released Android API versions to increase security in the Android system, for instance, by fixing that any application with READ_LOGS permissions could read any log of any application installed in to the device, or setting exported flag to false in IPC endpoints by default. These alone make the burden of the developer not only easier but also almost completely remove security issues regarding those issues.

Finding optimal security solution is always about balancing between usability and security. If a client requests that they must be able to login to the application offline, this could compromise credentials of that specific user in case of losing the device to an adversary, yet, it could still be an important requirement in a map application. Or, writing excessive tamper detection would be quite redundant in a very small scale notepad application with not much functionality versus big scale banking application.

Yet even the best security implementations in an application cannot save physical data in the device if the user is careless enough. For instance, on rooted device, if the user grants root permissions to a malicious application, then the malicious application could access all resources in the device, or if an attacker gets physical access to the device and manages to bypass the device's lock screen. But if an application is made secure, the attacker cannot gain anything by doing so, i.e. the attacker could only get user id and token but not password. Or, in a case with files, the attacker would only get encrypted files but not encryption keys if they are stored in the server.

This paper includes only the tip of the iceberg regarding secure mobile application development. Each and every topic contains so much more information and details that it would be possible to write a thesis of every chapter. Also, there was no way that it would have been possible to include every security aspect in this work, as every technology and process includes its own flaws. This work contains the most common pitfalls of mobile application development, however, that alone covers most of the security flaws in native Android applications.

References

2016 NowSecure Mobile Security Report. 2016. Security company NowSecure's security report about mobile applications in 2016. Accessed on 24.4.2016. Retrieved from https://info.nowsecure.com/rs/201-XEW-873/images/2016-NowSecure-mobile-security-report.pdf

Account Chooser. N.d. OpenID foundation's advertisement site for account chooser. Accessed on 3.4.2016. Retrieved from http://accountchooser.net

Alexander-Bown, S. N.d. *Android Security: Adding Tampering Detection to Your App*. Airpair micro consult service. Accessed on 20.3.2016. Retrieved from https://www.airpair.com/android/posts/adding-tampering-detection-to-your-android-app

Android Debug Bridge. N.d. Article in the official Android developer site. Accessed on 25.4.2016. Retrieved from https://developer.android.com/tools/help/adb.html

Android NDK. N.d. Article on Official Android Developer site. Accessed on 20.3.2016. Retrieved from https://developer.android.com/tools/sdk/ndk/index.html

Android OS. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/14873/android-os

Application Programming Interface. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/24407/application-programming-interface-api

Authorization. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/24130/authentication-authorization-and-accounting-aaa

Authenticate with a backend server. N.d. The official Android developer site. Accessed on 3.4.2016. Retrieved from https://developers.google.com/identity/sign-in/android/backend-auth

Authentication. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/342/authentication

Six, J. P. 2012. *Application Security for the Android Platform.* Sebastopol: O'Reilly Media, Inc.

Avoid security by obscurity. N.d. The OWASP site. Accessed on 20.3.2016. Retrieved from https://www.owasp.org/index.php/Avoid_security_by_obscurity

Bernat, V. N.d. *SSL/TLS & Perfect Forward Secrecy*. Vincent Bernat's blog. Accessed on 20.3.2016. Retrieved from http://vincent.bernat.im/en/blog/2011-ssl-perfect-forward-secrecy.html

Bray, T. 2013. *Verifying Back-End Calls from Android Apps*. Google's official Android developer blog. Accessed on 28.3.2016. Retrieved from http://android-developers.blogspot.fi/2013/01/verifying-back-end-calls-from-android.html

Certificate Authority (CA) . N.d. Definition in Search security dictionary. Accessed on 25.4.2016. Retrieved from http://searchsecurity.techtarget.com/definition/certificate-authority

Certificate and Public Key Pinning. N.d. The OWASP site. Accessed on 20.3.2016. Retrieved from https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning

Code Signing. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/24876/code-signing

Cohen, U. 2013. *Write your own Android Authenticator*. Udi Cohen's blog. Accessed on 28.3.2016. Retrieved from http://blog.udinic.com/2013/04/24/write-your-own-android-authenticator/

Content Provider Basics. N.d. Article on the official Android developer site. Accessed on 18.4.2016. Retrieved from https://developer.android.com/guide/topics/providers/content-provider-basics.html

Credentials. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/10259/credentials

Cyberattack. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/24748/cyberattack

Degges, R. 2015. *The Ultimate Guide to Mobile API Security.* Blog of Stormpath, management and authentication service. Accessed on 28.3.2016. Retrieved from https://stormpath.com/blog/the-ultimate-guide-to-mobile-api-security/

Denial-of-Service Attack (DoS). N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/24841/denial-of-service-attack-dos

Developer. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/17095/developer

Digital Certificate. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/1775/digital-certificate

Distribute Android apps in your organization. N.d. Google Apps Administrator support site. Accessed on 20.3.2016. Retrieved from https://support.google.com/a/answer/2494992?hl=en

Distribute apps in your organization. N.d. Google Developer Console Help site. Accessed on 20.3.2016. Retrieved from https://support.google.com/googleplay/android-developer/answer/2623322?hl=en

Drake, J., For a, P., Lanier, Z., Mulliner, C., Ridley, S., & Wicherski, G. P. 2014. *Android hacker's handbook.* Indianapolis: John Wiley & Sons, Inc.

Elenkov, N. 2011. *Using ECDH on Android*. Blog of Nikolay Elenkov. Accessed on 20.3.2016. Retrieved from http://nelenkov.blogspot.fi/2011/12/using-ecdh-on-android.html

Elenkov, N. 2012. *Certificate pinning in Android 4.2*. Blog of Nikolay Elenkov. Accessed on 20.3.2016. Retrieved from http://nelenkov.blogspot.fi/2012/12/certificate-pinning-in-android-42.html

Enable Unknown Sources. N.d. Amazon Help and Customer service. Accessed on 20.3.2016. Retrieved from http://www.amazon.com/gp/help/customer/display.html?nodeId=201482620

Encryption. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/5507/encryption

Exploiting Unintended Data Leakage (Side Channel Data Leakage). N.d. Article on InfoSec Institute's information security training. Accessed on 24.4.2016. Retrieved from http://resources.infosecinstitute.com/android-hacking-security-part-4-exploiting-unintended-data-leakage-side-channel-data-leakage/

Goldshlager, N. 2013. *How I Hacked Facebook OAuth To Get Full Permission On Any Facebook Account (Without App "Allow" Interaction).* Nir Goldshlager's secuirty blog. Accessed on 3.4.2016. Retrieved from http://www.nirgoldshlager.com/2013/02/how-i-hacked-facebook-oauth-to-get-full.html

Google Identity Platform. N.d. The official Android developer site. Accessed on 3.4.2016. Retrieved from https://developers.google.com/identity/

Godfrey, N., David, T., Raghav, S. & Onur, C. P. 2014. *Android Best Practices.* Apress. https://www.eff.org/deeplinks/2011/08/iranian-man-middle-attack-against-google

Google Play for Work. N.d. Article on Official Android Developer site. Accessed on 20.3.2016. Retrieved from https://developer.android.com/distribute/googleplay/work/about.html

Green, A. 2015. *SSL and TLS 1.0 No Longer Acceptable for PCI Compliance*. Varonis Security Suite blog. Acessed on 20.3.2016. Retrieved from https://blog.varonis.com/ssl-and-tls-1-0-no-longer-acceptable-for-pci-compliance/

Hacker. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/3805/hacker

Handling SSL Certificates in Android. 2013. KdHairy's blog about mobile security. Accessed on 20.3.2016. Retrieved from http://kdehairy.com/handling-ssl-certificates-in-android/

How ProGuard protects Android applications from reverse engineering. 2015. Blog about application development. Accessed on 24.4.2016. Retrieved from http://www.flyingtophat.co.uk/blog/2015/11/05/how-proguard-protects-android-applications-from-reverse-engineering.html

Hypertext Transfer Protocol. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/2336/hypertext-transfer-protocol-http

Hypertext Transport Protocol Secure. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/5361/hypertext-transport-protocol-secure-https

Insecure Local Storage. N.d. Article on InfoSec Institute's information security training. Accessed on 23.4.2016. Retrieved from http://resources.infosecinstitute.com/android-hacking-security-part-10-insecure-local-storage/

Insecure Local Storage: Shared Preferences. N.d. Article on InfoSec Institute's information security training. Accessed on 23.4.2016. Retrieved from http://resources.infosecinstitute.com/android-hacking-security-part-9-insecure-local-storage-shared-preferences/

Intents and Intent Filters. N.d. Article on offical Android developer site. Accessed on 17.4.2016. Retrieved from https://developer.android.com/intl/es/guide/components/intents-filters.html

Intent Spoofing Vulnerability in Android Apps – OWASP Top 10. 2015. Appvigil security blog. Accessed on 18.4.2016. Retrieved from https://www.appvigil.co/blog/2015/04/intent-spoofing-vulnerability-in-android-apps/

Judge, S. N.d. *Incorporate Tamper Detection.* Informative site about secure Androis application development. Android security site. Accessed on 20.3.2016. Retrieved from http://www.androidsecurity.guru/incorporate-tamper-detection/

Judge, S. N.d. *Take Care With Logging*. Informative site about secure Androis application development. Accessed on 24.4.2016. Retrieved from http://www.androidsecurity.guru/take-care-with-logging/

Judge, S. N.d. *Use the Android NDK for security Sensitive code*. Informative site about secure Androis application development. Accessed on 20.3.2016. Retrieved from http://www.androidsecurity.guru/use-the-android-ndk-for-security-sensitive-code/

Makan, K. 2013. *Knowing the SQL-injection attacks and securing our Android applications from them.* An excerpt from book, Android security cookbook. Accessed on

18.4.2016. Retrieved from https://www.packtpub.com/books/content/knowing-sql-injection-attacks-and-securing-our-android-applications-them

Man-in-the-Middle Attack. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/4018/man-in-the-middle-attack-mitm

Matake, N. 2014. *OpenID Connect 101.* Matake Nov's slideshow from OpenID Tech-Night vol.11. Accessed on 20.30.2016. Retrieved from http://www.slideshare.net/matake/technight11

Medianero, D. N.d. *OASAM-IS: Intent Spoofing*. Article on open Android security assessment methodology site. Accessed on 18.4.2016 Retrieved from http://oasam.org/en/oasam/oasam-is-intent-spoofing

Medianero, D. N.d. *OASAM-UIR: Unauthorized Intent Receipt*. Article on open Android security assessment methodology site. Accessed on 18.4.2016 Retrieved from http://oasam.org/en/oasam/oasam-uir-unauthorized-intent-receipt

Mobile Device. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/23586/mobile-device

Mobile Emulator. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/30676/mobile-emulator

Mobile Malware. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/29477/mobile-malware

Mobile Top 10 2014-M3, 2015. The OWASP Mobile Top 10 threaths. Accessed on 20.3.2016. Retrieved from https://www.owasp.org/index.php/Mobile_Top_10_2014-M3

Mobile Top 10 2014-M5. 2015. The OWASP Mobile Top 10 threaths. Accessed on 28.3.2016. Retrieved from https://www.owasp.org/index.php/Mobile_Top_10_2014-M5

Mobile Top 10 2014-M7. 2014. The OWASP Mobile Top 10 threaths. Accessed on 24.4.2016. Retireved from https://www.owasp.org/index.php/Mobile_Top_10_2014-M7

Moulu, A. 2012. *From 0 perm app to INSTALL_PACKAGES on Samsung Galaxy S3*. Article on André Moulu's home page. Accessed on 18.4.2016. Retrieved from http://sh4ka.fr/android/galaxys3/from_0perm_to_INSTALL_PACKAGES_on_galaxy_S3.html

Moroney, L. 2016. *Using Google Sign-In with your server*. The official Android developers blog. Accessed on 3.4.2016. Retrieved from http://android-developers.blogspot.fi/2016/01/using-google-sign-in-with-your-server.html

Obfuscation. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/16375/obfuscation

OpenID Connect Basic Client Implementer's Guide 1.0 - draft 37. 2015. OpenID technical documentation. Accessed on 28.3.2015. Retrieved from https://openid.net/specs/openid-connect-basic-1_0.html

Oliva Fora, P. 2014. *Beginners guide to reverse engineering Android Apps.* RSA confrerence slide-show. Accessed on 20.3.2016. Retrieved from https://www.rsaconference.com/writable/presentations/file_upload/stu-w02b-beginners-guide-to-reverse-engineering-android-apps.pdf

PCI Security Standards Council. 2015. *Migrating from SSL and Early TLS*. Accessed on 20.3.2016. Retrieved from https://www.pcisecuritystandards.org/documents/Migrating_from_SSL_Early_TLS_Information%20Supplement_v1.pdf

Phishing. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/4049/phishing

Requesting Permissions at Run Time. N.d. Article on official Android developer site. Accessed on 17.4.2016. Retrieved from

https://developer.android.com/intl/es/training/permissions/requesting.html

Permission. N.d. Article on Official Android Developer site. Accessed on 17.4.2016. Retrieved from https://developer.android.com/guide/topics/manifest/permission-element.html

Pinning Cheat Sheet. N.d. The OWASP site. Accessed on 20.3.2016. Retrieved from https://www.owasp.org/index.php/Pinning_Cheat_Sheet

ProGuard. N.d. Article on Official Android Developer site. Accessed on 20.3.2016. Retrieved from https://developer.android.com/tools/help/proguard.html

Reading and Writing Logs. N.d. Article on official Android developer site. Accessed on 24.4.2016. Retrieved from https://developer.android.com/tools/debugging/debugging-log.html

Ristic, I. 2013. *SSL Labs: Deploying Forward Secrecy*. Security professional community. Accessed on 20.3.2016. Retrieved from https://blog.qualys.com/ssllabs/2013/06/25/ssl-labs-deploying-forward-secrecy

Rouse, M., Phifer, L. 2012. *Mobile application manager (MAM)*. TechTarget global network of technology-specific websites. Accessed on 20.3.2016. Retrieved from http://whatis.techtarget.com/definition/mobile-application-manager-MAM

Sakimura, N., NRI, Bradley, J., Ping Identity, Jones, M., Microsoft, de Medeiros, B., Google, Mortimore, C. & Salesforce. 2014. *OpenID Connect Core 1.0 incorporating errata set 1.* Final specifications from the official OpenID site. Accessed of 3.4.2016. Retrieved from https://openid.net/specs/openid-connect-core-1_0.html

Saving Data in SQL Databases. N.d. Article in the official Android developer site. Accessed on 24.4.2016. Retrieved from https://developer.android.com/training/basics/data-storage/databases.html

Sanchez, A. 2014. *Personal banking apps leak info through phone*. IOActive secuirty advisor's blog. Accessed on 24.4.2016. Retrieved from http://blog.ioactive.com/2014/01/personal-banking-apps-leak-info-through.html

Schoen, S., & Galperin, E. 2011. *Iranian Man-in-the-Middle Attack Against Google Demonstrates Dangerous Weakness of Certificate Authorities*. Accessed on 20.3.2016. Retrieved from https://www.eff.org/deeplinks/2011/08/iranian-man-middle-attack-against-google

Shòu, j. 2015. *Android Content Provider Security.* Artice on Shòu jiāo wǔ's blog. Accessed on 18.4.2016. Retrieved from http://en.wooyun.io/2015/07/01/android-content-provider-security.html

Security with HTTPS and SSL. N.d. Official Android Developer site. Accessed on 20.3.2016. Retrieved from https://developer.android.com/training/articles/security-ssl.html

Security Tips. N.d. Official Android developer site. Accessed on 28.3.2016. Retrieved from https://developer.android.com/training/articles/security-tips.html#UserData

Sign-in flow. N.d. The Official Android developer site. Accessed on 3.4.2016. Retrieved from https://developers.google.com/identity/toolkit/web/federated-login

Signing Your Applications. N.d. Article on official Android Developer site. Accessed on 20.3.2016. Retrieved from https://developer.android.com/tools/publishing/app-signing.html

Six, J. 2012. *An in depth introduction to the android permission model and how to secure multi component applications*. OWASP's slideshow from AppSecDC 2012 conference. Accessed on 17.4.2016. Retrieved from https://www.owasp.org/images/c/ca/ASDC12-An_InDepth_Introduction_to_the_Android_Permissions_Modeland_How_to_Secure_MultiComponent_Applications.pdf

Smart Lock for passwords on Android. N.d. Article on official Android Developersite. Accessed on 9.4.2016. Retrieved from https://developers.google.com/identity/smartlock-passwords/android/

SSLSocket. N.d. Official Android developer site. Accessed on 20.3.2016. Retrieved from https://developer.android.com/intl/zh-cn/reference/javax/net/ssl/SSLSocket.html

Steele, C. N.d. *Mobile device management vs. mobile application management*. Tech Target global network of technology-specific websites. Accessed on 20.3.2016. Retrieved from http://searchmobilecomputing.techtarget.com/feature/Mobile-device-management-vs-mobile-application-management

Storage Options. N.d. Article in the official Android developer site. Accessed on 23.4.2016. Retrieved from https://developer.android.com/guide/topics/data/data-storage.html

Strumpflohner, J. 2011. *Coding an update functionality for your Android app*. Juri Strumpflohner's blog. Accessed on 20.3.2016. Retrieved from http://juristr.com/blog/2011/02/coding-update-functionality-for-your/

Superuser. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/9588/superuser

System permissions. N.d. Artice on official Android developer site. Accessed on 18.4.2016. Retrieved from https://developer.android.com/guide/topics/security/permissions.html

Transport Layer Protection Cheat Sheet. 2016. The OWASP site. Accessed on 20.3.2016. Retrieved from https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

Transport Layer Security. N.d. Definition in Technopedia dictionary. Accessed on 25.4.2016. Retrieved from https://www.techopedia.com/definition/4143/transport-layer-security-tls

User Agent Capabilities: Android 5.0.0. N.d. Qualys SSL Labs site. Accessed on 20.3.2016. Retrieved from https://www.ssllabs.com/ssltest/viewClient.html?name=Android&version=5.0.0

User Authentication with OAuth 2.0. N.d. OAuth official site. Accessed on 28.3.2015. Retrieved from

http://oauth.net/articles/authentication/

Wallen, J. 2016. *How to use Android Marshmallow's Smart Lock for Passwords*. Article on Tech republic site. Accessed on 16.4.2016. Retrieved from http://www.techrepublic.com/article/how-to-use-android-marshmallows-smart-lock-for-passwords/

Weichao, S. 2014. *Android Custom Permissions Leak User Data*. Tren micro's Tren labs's official blog. Accessed on 17.4.2016. Retrieved from

http://blog.trendmicro.com/trendlabs-security-intelligence/android-custom-permissions-leak-user-data/

Welcome to OpenID Connect. N.d. The OpenID official site. Accessed on 3.4.2016. Retrieved from https://openid.net/connect/

What is the "Unknown Sources" setting?. N.d. Android Enthusiasts. Accessed on 20.3.2016. Retrieved from https://android.stackexchange.com/tags/unknown-sources/info

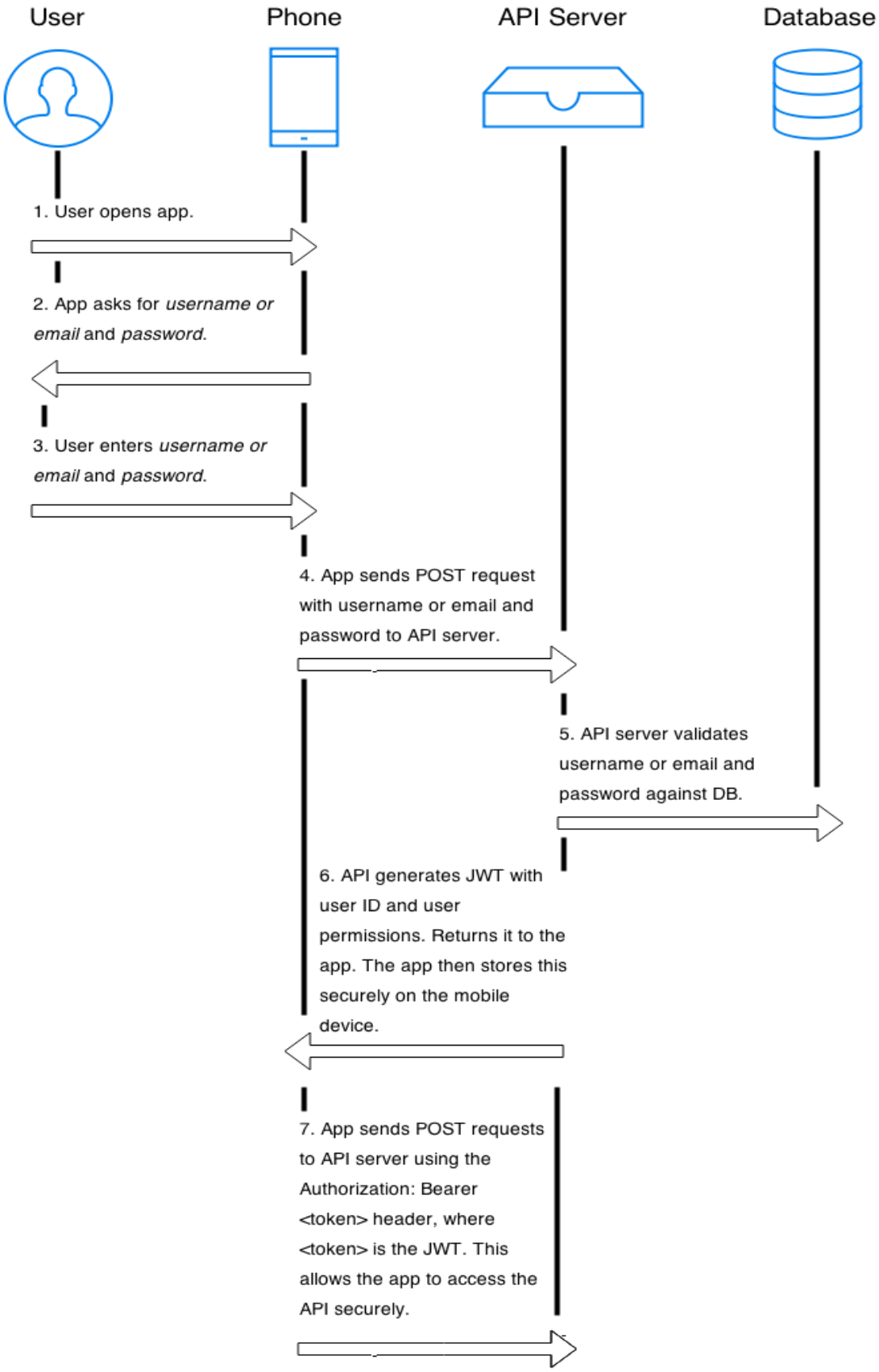
When are self-signed certificates acceptable? N.d. SSL shopper article. Accessed on 20.3.2016. Retrieved from https://www.sslshopper.com/article-when-are-self-signed-certificates-acceptable.html

Why SSL? The Purpose of using SSL Certificates. N.d. SSL shopper article. Accessed on 20.3.2016. Retrieved from https://www.sslshopper.com/why-ssl-the-purpose-of-using-ssl-certificates.html

Xiao, C. & Olson, R. 2014. *Insecure Internal Storage in Android.* Artice on Palo Alto Networks resarch center site. Accessed on 23.4.2016. Retrieved from http://researchcenter.paloaltonetworks.com/2014/08/insecure-internal-storage-android/

Appendices

Appendice 1. OAuth 2.0 workflow. (Adapted from Degges, R. 2015)



Appendice 2. An example of code obfuscation. (Adapted from How ProGuard protects Android applications from reverse engineering, 2015.)

Before code obfuscation

```
public class RandomNameGenerator {

    private final Random random = new Random();

    private String[] mFirstNames;
    private String[] mLastNames;

    public RandomNameGenerator(String[] firstNames, String[] lastNames) {
        mFirstNames = firstNames;
        mLastNames = lastNames;
    }

    public String[] getName() {
        int firstNameIndex = random.nextInt(mFirstNames.length);
        int lastNameIndex  = random.nextInt(mLastNames.length);

        return new String[] { mFirstNames[firstNameIndex], mLastNames[lastNameIndex] };
    }

    // This method isn't used, so let's see what ProGuard does with it
    public void dummyMethod() {}
}
```

After code obfuscation

```
public class b
{
  private final Random a = new Random();
  private String[] b;
  private String[] c;

  public b(String[] paramArrayOfString1, String[] paramArrayOfString2)
  {
    this.b = paramArrayOfString1;
    this.c = paramArrayOfString2;
  }

  public String[] a()
  {
    int i = this.a.nextInt(this.b.length);
    int j = this.a.nextInt(this.c.length);
    return new String[] { this.b[i], this.c[j] };
  }
}
```