Mobile Security - an overview

Fakulty for Informatics and Mathematics
of University of Applied Sciences Munich

Handout

written from

Felix Strobel, André Braunersreuther

supervised by

Prof. Dr. Gudrun Socher

June 2019

Contents

1 Introduction

1.1 Definition

IT-Security is a generic term for a lot of different topics. Mobile security is a particular part of security with the same concerns but with focus on mobile devices. The rising of mobile devices started with the first mobile phones and handhelds. With arriving the first touchscreen devices and a much higher usability the number of mobile devices rised to a smartphone-penetration of 66% in 2018. [1]

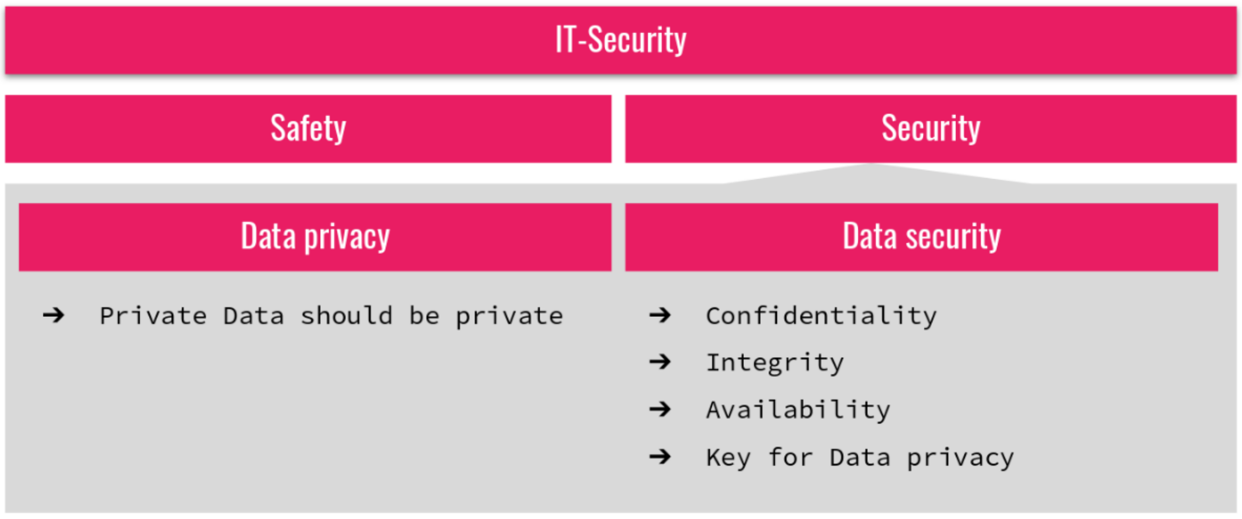

Figure 1.1: Parts of IT-Security

1.1.1 Safety

The main question in safety is: *Does my application what it is supposed to do?.*
So it is about functionality and often directly defined by customers or stakeholder. One strategy to avoid problems with safety topics are automated test. For example unit tests and integration test running by a continuous integration (CI).
But safety is not the focused topic.

1.1.2 Security

Security is about information and data. Like in Fig. 1.1 it can be split into privacy and security.

Privacy

Privacy' subject is to make sure that personal data like age, name or address are accessible only by people who are allowed.
Since GDPR came into force, it is not only a ideological question it is also a lawful question. And not fulfill this law can result in very high fines.
The goals, to keep private data private, can only get achieved with security.

Security

Security has three general targets.

Confidentiality is about making sure to restricted the access to a level where only people with the necessary privileges are allowed to see, change or delete these data.

Integrity makes sure that every unauthorized change can be detected and changes are verifiable.

Availability Means that the information should be always available to the user.

But there are also some more specific target. These does not fit in every use case.

Authenticity aims on reliability. It should be clear that a message comes from the specific sender.

Anonymity In some cases it should not be possible to connect a network package to a specific person.

In a lot of cases it is a trade of between these targets. In this case it should be clear why and what it means to not fulfill the target to 100%.

1.2 Motivation

Mobile devices get used for a lot of topics. Manage contact, doing phone calls and messaging are only the simplest examples. Nowadays the smartphone is used for nearly everything and became a personal assistant. That means a lot of personal data are stored on the device. Just to mention some of these data:

- Identities, for social media but also for credit/customer cards or online banking.
- Histories, like browser history or location history
- Payments
- Location

Implementing secure mobile applications can be more difficult than for a regular desktop environment because the user wants a simpler UI and an easy and fast access to their data. So it should be avoided to ask for long inputs or complicated authentication methods.

A further difference to the desktop environment is the mobility of these devices. Because of that an smartphone can easily get in the wrong hands. And the connectivity of these devices is higher since they support nearly every modern interface, which can be wired but in the most time is wireless. All these interfaces open a new attack vector, which can be used to get some data or control over the device.

But away from the technical background there are more reasons to secure a mobile application.

- The store provider check their apps for common bugs which can lead to secure gap
- Laws like *GDPR* force us to secure user data
- Corrupt data inside the app can destroy the user experience
- Loosing data can lead to reputational damage
- A exploited security issue can cost a lot of money
- Your own standards

2 Topics to be aware of

2.1 Permissions

As the Android OS is built on top of the Linux Kernel it also comes with the permission approach of Linux. This way an application can only access a limited range of system resources by default and every access to a resource is managed by the OS. To get more access than the standard provided by the basic sandbox, an application must define the resources it wants to have access to in its manifest. The user gets asked to grant these permissions the first time an app wants the access. This gets saved to the device for later usage and the user doesn't have to grant it again. Like in Linux the permission model is a user based model and as every application is its own user every application has its own permissions. This also isolates the user resources from one another. In addition every application has to explicitly define which resources it shares with other applications. A good way of requesting permissions is to minimize the number of requested permissions. Simply because if an app can't do more than it should, unexpected situations won't arise. So if a permission is not required it should not be requested. Also self created permissions should be as few as possible and rather system defined ones should be used as while granting the permissions a user could get confused by a big list of unknown permissions.

2.2 Networking

Networking with any device is always risky just because of the fact, that the data that has to be transmitted is potentially private data of the user. Any loss or "publication" of this data can harm the user and/or the trust a user has in the application. The highest endeavour should always be to keep user data secure at all times. For this reason it is important to use secure connections to any network an app wants to send data to or receive from. The key to secure network traffic is to use appropriate protocols for the connections. For trivial example would be to use HTTPS over HTTP whenever the server provides it. On a mobile device any network connection covers an additional possible security threat, as it is frequently connected to unsecure networks via Wi-Fi. Here the threat is first that the network itself could be unsecure and second it is not known which other users are in the network and possibly have bad intentions. Another point is that some developer tend to use localhost ports for sending data over Inter Process Communication (IPC). This is not a good approach as these interfaces are accessible by other applications on the device and so the data could be read by the wrong process. The way to solve this is to use IPC mechanisms provided by android.

2.3 Input Validation

This topic is the most common security problem [https://developer.android.com/training/articles/securit tips] if not done correctly or not done at all. Every data an app is receiving over the network, as input from the user or from an IPC is potentially threatening even if it is not meant to be harmful in first place. The most common problems are buffer overflows, use after free and off-by-one errors.[] This threat can be reduced if the data gets validated. Type-safe languages already tend to reduce the likelihood of input validation issues. Also pointers should always be handled very carefully so that they don't point on the wrong address and when using buffers they should always be managed. When using queries for an SQL database there could be the issue of SQL injection that should be taken care of by using parameterized queries or limiting permissions to read-only or write-only. Another way is to use well formated data formats and verify each expected format.

2.4 Handeling User Data

Nowadays users get more and more aware that their personal data is important and that it should be handled with care. For this it is necessary to not lose the trust of the user while using an app because he wouldn't come back to it then anymore. One good approach to accomplish this to minimize the use of APIs where the data is sent to. If this is necessary for the functionality of the app the way to go would be to not send the plain information but a hash or a non-reversible form of the data. This way the data doesn't get exposed to third parties for services or advertisement. This approach is also valid when storing the data on the phone. The data could also get leaked to other apps through permissions to IPCs meaning that one app has the data and another app normaly has no permission for accessing this data but it gets it through a connection to the first app. Third party services or advertisement often require personal information of users to work for not obvious reasons. Here a good thing to do is not to provide that data if the developer is not sure why the service would need that information.

2.5 Cryptography

Next to other security features that were mentioned, Android also provides a wide array of algorithms for protecting data using cryptography. So there is no need to implement your own algorithms. A good approach for this is to always use the highest level of the existing implementations that still support the usecase in the app. This way the developer can make the data as save as possible because the levels of security differentiate in how strong they are. While using these algorithms it is also important to use secure random number generators to initialize these algorithms. If in this case the developer only uses a non secure random number generator this weakens the cryptographic algorithms significantly.

2.6 Credentials

A good approach for handeling user credentials is to minimize the frequency of asking for the user credentials because asking for it a lot makes phishing attacks more likely to be successful each time. What is quite obvious is that the developer should never store user names or passwords on the device itself. Instead the way to go should be to initially authenticate a user and then use an authorization token.

2.7 Interprocess Communication

This communication appears when sending data from one app to another or also from process to process within an app. For this Android provides a functionality using intents. These intents get sent indirectly to another app via Reference Monitor which takes care of the security in this case. When the sending process or application specifies a permission that is needed to receive the intent the Reference Monitor checks if the receiver actually has this permission and if not it cancels the communication. So the permission gets mandatory to receive this specific intent. These intents can be broadcasted so that every process has the chance to get it or sent directly to a receiver. If the data that has to be sent is sensitive the better approach would be to send it directly for obvious reasons.

2.8 Dynamically Loaded Code

This is loading executable code from somewhere else which is obviously a security risk as the source is not always fully transparent and so it may be harmful. It is a big threat for code injection or code tampering which then can do harmful activities as it eather adds or manipulates the code of the apk. Dynamically loaded code can make it impossible to verify the behaviour of an app which then in the next step can make it prohibited in some environments as the code is more or less unpredictable. One important thing to keep in mind is that dynamically loaded code runs with the same security permissions as the app itself.

3 Example

This example is about Google and Apple Pay. It explains how these two paying applications work and why it is done how it is done.
Before diving into the algorithm behind the paying methods, some terms should be explained.

Payment Networks are credit institutions which managing credit cards and the transfer from money. But also Paypal and VisaCheckout are payment Networks.

Token Service Provider are providing a very secure environment to map credit card information to individual token. [2]

Secure Enclave is the build in security chip in Apple devices. All keys are stored in here.

Secure Element is also apple specific and is paired with the Secure Enclave via hardware in the factory. It is located in the NFC-Chip to emulate the credit card.

Apple Pay and Google Pay are similar but different in detail. Both have four main steps.

1. Adding a card
2. Initiate
3. Authorize
4. Finish

3.1 Google Pay

Google Pay is using the process of tokenization [3] like a lot other mobile payment application and is standardized together with card issuer and token service provider (TSP).

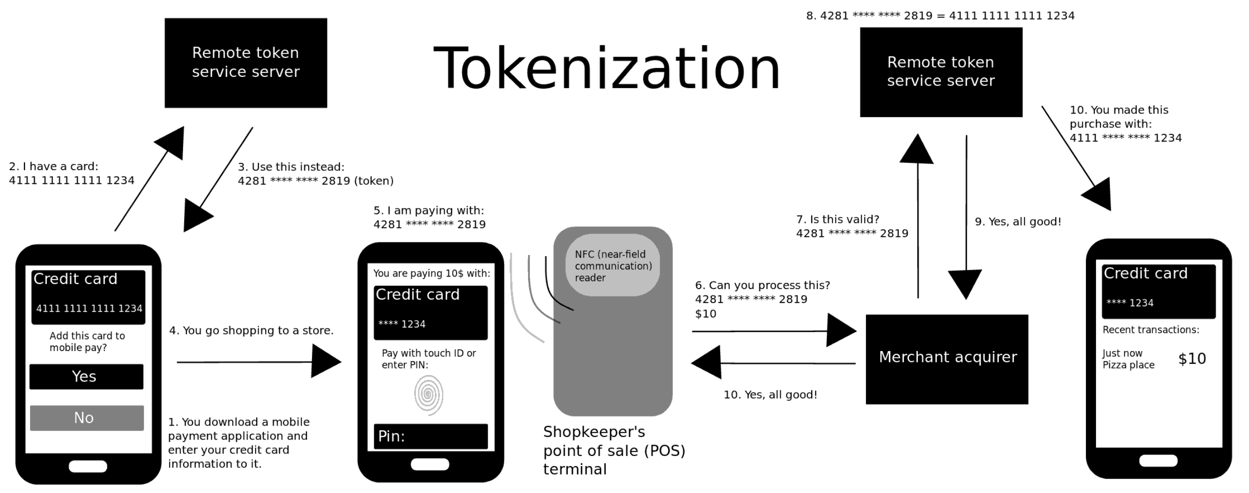
Tokenization
Remote token service server
2. I have a card: 4111 1111 1111 1234
3. Use this instead: 4281 **** **** 2819 (token)
Credit card
4111 1111 1111 1234
Add this card to mobile pay?
Yes
No
1. You download a mobile payment application and enter your credit card information to it.
4. You go shopping to a store.
5. I am paying with: 4281 **** **** 2819
You are paying 10$ with:
Credit card
**** 1234
Pay with touch ID or enter PIN:
Pin:
NFC (near-field communication) reader
Shopkeeper's point of sale (POS) terminal
6. Can you process this? 4281 **** **** 2819 $10
10. Yes, all good!
Merchant acquirer
7. Is this valid? 4281 **** **** 2819
9. Yes, all good!
8. 4281 **** **** 2819 = 4111 1111 1111 1234
Remote token service server
10. You made this purchase with: 4111 **** **** 1234
Credit card
**** 1234
Recent transactions:
Just now Pizza place $10

Figure 3.1: Mobile payment tokenization

3.1.1 Adding a card

After downloading the payment app (Fig. 3.1 Step 1) the user gives the card number. In the background the app ask for a token to represent the card. (Fig. 3.1 Step 2 and 3)
This token, provided from a TSP, is stored at the device and encrypted with a single or limited used key provided from the Payment Network.
This is an elegant way to not safe user data on the device. Security of the token is finally guaranteed by encryption. So this step applies two principles of security. And it also use appropriated protocols to exchange the card information.

3.1.2 Initiate payment

The customers taps their device on a NFC Terminal (Fig. 3.1 Step 4). With this action the application start transmitting the token, a token expire date and the cryptogram (Fig. 3.1 Step 5). The cryptogram is generated by the token, timestamp and an Application Transaction Counter which is increased at every transaction and prevent the multiple use of one message.
This is a good example how appropriate protocols can be a secure way to communicate even if everybody could listen.

3.1.3 Authorize

The merchant receive the message from NFC terminal and sends all information including the price to his card network (Fig. 3.1 Step 6 and 7). The card network validates the cryptogram with help of TSP and matches the token to the real card number (Fig. 3.1 Step 8).
This step also fulfills one principle of security - validation.

3.1.4 Finish

Billing details get decrypted and the acquiring bank completes the transaction with customers bank Merchant and customer get information about validation (Fig. 3.1 Step 9 and 10).
All with appropriated protocols.

3.2 Apple Pay

Apple Pay also use the tokenization. But Apple also provides a special place to store the token.

3.2.1 Adding a card

After downloading the payment app (Fig. 3.1 Step 1) the user gives the card number. In the background the app downloads the wallet pass file. This file includes some meta data and the token. All these data getting bind to the Secure Element [4]. This is encrypted by the Secure Enclave. (Fig. 3.1 Step 2 and 3)
This is an elegant way to not safe user data on the device. Security of the token is finally guaranteed by encryption. So this step applies two principles of security. And it also use appropriated protocols to exchange the card information.

3.2.2 Initiate payment

The customers taps their device on a NFC Terminal (Fig. 3.1 Step 4). Then the user needs to authenticate the transaction. With this action the application decrypt the token from Secure Element and starts transmitting the token, a token expire date and the cryptogram (Fig. 3.1 Step 5). The cryptogram is generated by the token, timestamp and an Application Transaction Counter which is increased at every transaction and prevent the multiple use of one message.
This is a good example how appropriate protocols can be a secure way to communicate even if everybody could listen.
Additional to Google Pay, Apple Pay asks for permission to pay.

3.2.3 Authorize

The merchant receive the message from NFC terminal and sends all information including the price to his card network (Fig. 3.1 Step 6 and 7). The card network validates the cryptogram

with help of TSP and matches the token to the real card number (Fig. 3.1 Step 8).
This step also fulfills one principle of security - validation.

3.2.4 Finish

Billing details get decrypted and the acquiring bank completes the transaction with customers bank Merchant and customer get information about validation (Fig. 3.1 Step 9 and 10).
All with appropriated protocols.

Bibliography

[1] https://www.publicismedia.de/wp-content/uploads/2017/10/2017-10-16-mobile-advertising-forecasts-2017-de.pdf - Visited on 01.07.2019

[2] https://blog.rsisecurity.com/what-is-a-token-service-provider/ - visitied on 29.05.2019

[3] https://developer.samsung.com/tech-insights/pay/tokenization - visited on 30.05.2019

[4] https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf - visited on 30.05.2019