C | ISPA
Center for IT-Security, Privacy and Accountability

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Saarland University
Information Security & Cryptography Group
Prof. Dr. Michael Backes

# Android Security Lab WS 2014/15

# Course Project: Secure Inter-App Communication

M.Sc. Sven Bugiel

Version 1.0 (October 13, 2014)

# 1 Organizational Matters

## General Topic

In this course project you will work with selected aspects of Android's application layer security from the app developer's point of view. Focus of this work is secure inter-app communication (i.e., between app components). Requirement is that you are able to independently consult the Android developer documentation, examples, and Internet sources in order to accomplish the tasks described in Section 2.

## Teams

Each team consists of 2 students and will work on the topic explained in Section 2. Register your team on Monday, 13/10/2014 via email to the lab supervisor at `bugielcs.uni-saarland.de` In your email state the name, (student ID), and email address of each team member.

## Custom Topics

It is generally possible for you to propose a custom project idea or deviations from the proposed scenario. However, this has to be discussed with and approved by the lab supervisor!

## Android Version

In general, the teams are free to chose the Android version on which they implement their architecture. A minimum of version 2.2 is recommended to target more common versions[1].

## Reporting

Although the work is to a big extent conducted autonomously by the teams, you are advised to keep in contact with the lab supervisor to ensure your work is on track and satisfies the requirements for a (very) good grade.

In addition, each team has to submit a final report (approx. 10 pages), that is mandatory for successful participation. The final submission also has to include the source code of the team's solution in addition to the report. The teams should clearly state which authors were responsible for which part of the work and report. The style of the report can be freely chosen, but should be reasonable w.r.t. margin width, font size, etc. to ensure readability.

The final report should be written for an audience that is familiar with the topic (i.e., no extensive background sections, but focus on the project related facts). The final report should contain the following content:

1. a short introduction to the chosen topic which explains the addressed problem/vulnerability, motivation of the work, and proves that you understood the problem set
2. a design section, explaining the architecture/attack and design decisions made w.r.t. the Android design as well as *briefly* discussing possible alternative approaches
3. an implementation section briefly describing technical aspects in order to realize the design and to ease the code review process
4. a *short* conclusion summarizing the results, lessons learned, and potential open problems/future work you identified

## Deadlines

**Final report and code submission:** Friday, November 2014, 2014 at 11:59 PM (local time)

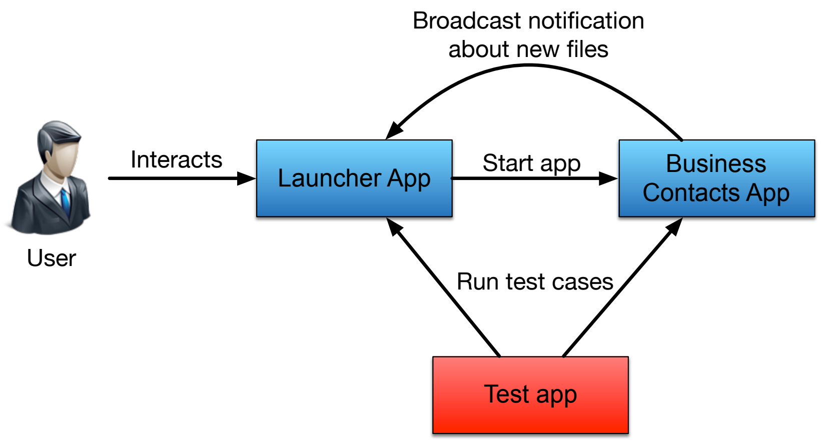
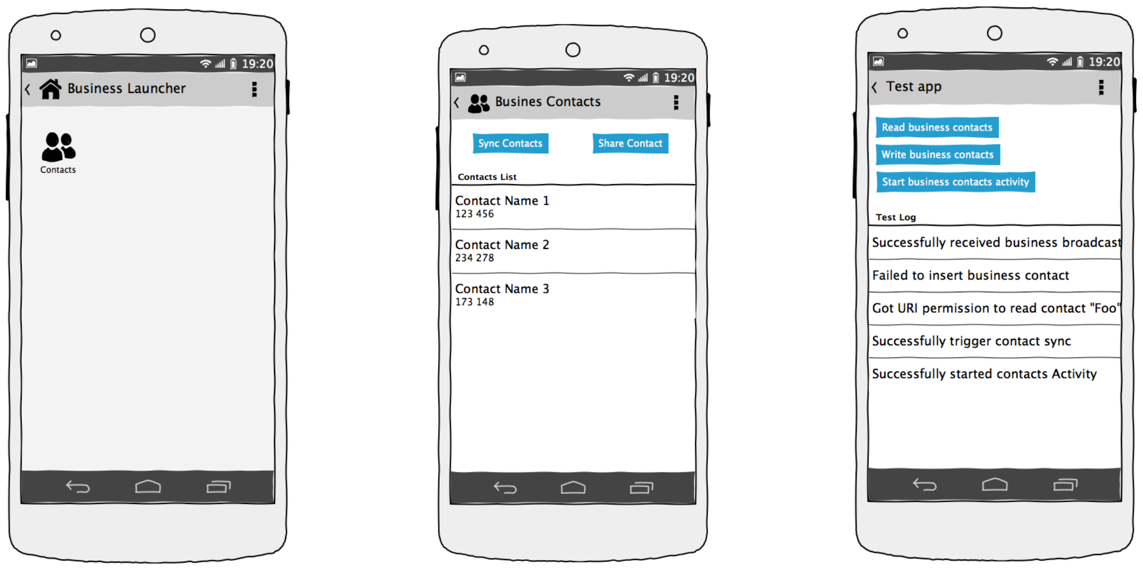


Figure 1: High-level design of the business apps suite.



Contact Name 1
123 456
Contact Name 2
234 278
Contact Name 3
173 148

Successfully received business broadcast
Failed to insert business contact
Got URI permission to read contact "Foo"
Successfully trigger contact sync
Successfully started contacts Activity

(a) Mock interface of the launcher app.

(b) Mock interface of the business contacts app.

(c) Mock interface of the test app.

Figure 2: Mock interfaces of the business and test apps.

## 2 Scenario

The scenario chosen for this course project deals with creating a simple suite of business apps that should communicate securely with each other in order to protect their business data and information. Figure 1 illustrates the high-level design. This design involves two business apps and one test app.

### 2.1 Launcher App

The launcher app is the entry point for the user to interact with the business apps. It should have an Activity similar to the mockup shown in Figure 2a, where it show entries/icons for the other two business apps. By clicking on one of the entries, the Activity of the clicked app should be started (i.e., `startActivity`). The launcher entries can be statically hardcoded into the launcher app or, preferably, the launcher app uses the PackageManager to search for other business apps and to retrieve information (like icon and name) from the PackageManager to create the app entries dynamically.

[1]`http://developer.android.com/about/dashboards/index.html`

Additionally, the launcher app should listen for broadcast Intents from the other business apps, which send broadcasts to inform the launcher app about new events (see below). Those events can be simply logged with `Log`.

## 2.2 Business Contacts App

The business contacts app should represent an app that syncs contacts stored locally in a ContentProvider component of this app with contacts stored on the company server. For simplicity, you should **not** implement a real contacts sync with a web server, but instead the contacts sync app can create fake contacts: it simply creates a new random name and phone number and inserts it into its ContentProvider.

The app should have a simple Activity similar to the one mockup in Figure 2b, where it lists the contacts currently stored locally in the ContentProvider and provides buttons to start syncing contacts and to share contact with another app. This should be implemented as a simple `ListActivity` with an adapter to the local ContentProvider.

The contacts sync functionality (i.e. creating and inserting a new fake contact) should be implemented as a service. This service should provide two functions: `void syncContacts()` to start the contacts sync and `String getVersion()` to get information about the contacts sync protocol version. The latter function can return any String of your liking, like “smartseclab-v0.1” or so. When the contacts sync has finished (i.e. finished inserting a fake contact), it should send a broadcast Intent to inform the launcher app about this event.

Sharing a contact should be implemented in this case by simply sending the URI of a randomly selected contact in the ContentProvider via Intent to the main Activity of the test app.

## 2.3 Test App

The test app represents a private, i.e., not-business app. It implements some test cases to check the security of the business apps, but also acts as a benign app to receive shared data (e.g., a shared contact).

The app should have an interfaces similar to the mockup in Figure 2c. Test cases should be started by pressing a button. In addition, results of the tests should be presented in a simple log, e.g., by appending to a `TextView`.

The test cases the app should implement are:

- Querying contacts information from the business contacts app’s ContentProvider
- Inserting contacts information into the business contacts app’s ContentProvider
- Directly starting the Activity of the business contacts app
- Triggering the contacts sync functionality of the business contacts app’s Service
- Receiving broadcast Intents by the business contacts app (Does not require a button, but only a BroadcastReceiver)
- Successfully reading contacts information of a shared contact URI received via Intent from the business contacts app

## 2.4 Security Requirements

While implementing the three apps described above, your task is to use Permissions and information provided by the PackageManager to secure the business apps communication with each other and their interfaces. This includes the following requirements:

- The business contacts app should only be started by the business launcher app or other business apps, but no other app including the default Android launcher.
    - Use a custom Permission to ensure that only business apps can start the Activity of the business contacts app.

  - Use a custom Intent filter for the Activity of the business contacts app to prevent this app from showing up in the Android default launcher.

- Use a custom Permission to ensure that only business apps can receive the Broadcast by the business contacts app that informs about new events.
- The Service of the business contacts app should be accessible by any app. However, in the `void syncContacts()` function use `Binder.getCallingUid()` and the PackageManager to implement access control that ensures that the caller to `syncContact()` has been signed with the same key as the business contacts app.
- Use a custom Permission to ensure that reading the ContentProvider of the business contacts app is not possible for non-business apps, except when using the share contact functionality. Use `URIPermission` to securely share a single contact with the test app, i.e., allow the test app to temporarily read a single contact from the ContentProvider of the business contacts app.
- Use a custom Permissions to ensure that writing to the ContentProvider of the business contacts app requires consent of the user (i.e., during installation of an app, the user must confirm that this app might insert new business contacts).

## 3 References

In the following are some example references on the Internet that can help you getting started with implementing the apps:

- Using OnClickListener for Buttons
  `http://developer.android.com/guide/topics/ui/controls/button.html#ClickListener`
- Starting another Activity
  `http://developer.android.com/training/basics/firstapp/starting-activity.html`
- Basics of using ContentProvider
  `http://developer.android.com/guide/topics/providers/content-provider-basics.html`
- Implementing a ContentProvider
  `http://www.tutorialspoint.com/android/android_content_providers.htm`
- Example for using URI permission when starting an Activity
  `http://thinkandroid.wordpress.com/2012/08/07/granting-content-provider-uri-permissions/`
  `http://stackoverflow.com/questions/13435654/how-to-grant-temporary-access-to-custom-content-provider-using-flag-grant-read-u?answertab=active#tab-top`
- Getting the Intent that started an Activity
  `http://developer.android.com/reference/android/app/Activity.html#getIntent()`
- Permissions on Activity components
  `http://developer.android.com/reference/android/app/Activity.html#Permissions`
- Binder's `getCallingUid`
  `http://developer.android.com/reference/android/os/Binder.html#getCallingUid()`