

DD2525 – Lab 3

Android Security

Deadline: May 21, 2021

1 Introduction

This assignment introduces you to some of the key concepts in Android application security. At its core, the Android platform relies on Linux-based access control and controlled inter-app communication via the permission system to enforce security. As this lab will show, the current security mechanisms of Android are not always sufficient to protect the privacy of users. Common malpractices in Android development may allow an attacker to access users' sensitive data. In this lab, you will investigate the root cause of some of these vulnerabilities, exploit them by implementing attacks, and suggest and implement countermeasures to fix them.

2 Getting Started

2.1 Install and configure Android Studio

First, make sure you have Java Development Kit (JDK) – at least – version 1.8 installed. Download Android Studio and follow the installation steps for your preferred operating system.


When running Android Studio for the first time, let it go through the standard initialization process to install the latest SDK and platform tools. Next, you will have to create an emulator by clicking on `Configure > AVD Manager > Create Virtual Device` on the start screen. Choose `Nexus 5 Template` and click next, **do not** use the recommended target, instead go to `x86 Images` tab and use as target `Android API 28, ABI x86`.

Select a system image

Recommended x86 Images Other Images

Release Name	API Level ▼	ABI	Target
Q Download	29	x86_64	Android 10.0
Q Download	29	x86	Android 10.0
Pie Download	28	x86_64	Android 9.0 (Google Play)
Pie Download	28	x86	Android 9.0 (Google APIs)
Pie Download	28	x86_64	Android 9.0 (Google APIs)
Pie	28	x86	Android 9.0
Pie Download	28	x86_64	Android 9.0
Oreo Download	27	x86	Android 8.1 (Google APIs)
Oreo Download	27	x86_64	Android 8.1
Oreo Download	27	x86	Android 8.1
Oreo Download	26	x86	Android 8.0 (Google APIs)

Pie



API Level
28

Android
9.0

Android Open Source Project

System Image
x86

Recommendation
Consider using a system image with Google APIs to enable testing with Google Play Services.

2.2 Starting code for the assignment

Download and extract the Lab's zip file from Canvas. There is a directory for each task where you will find the vulnerable apps as well as the template of exploit apps related to that task. You can open each of these apps as an Android project.

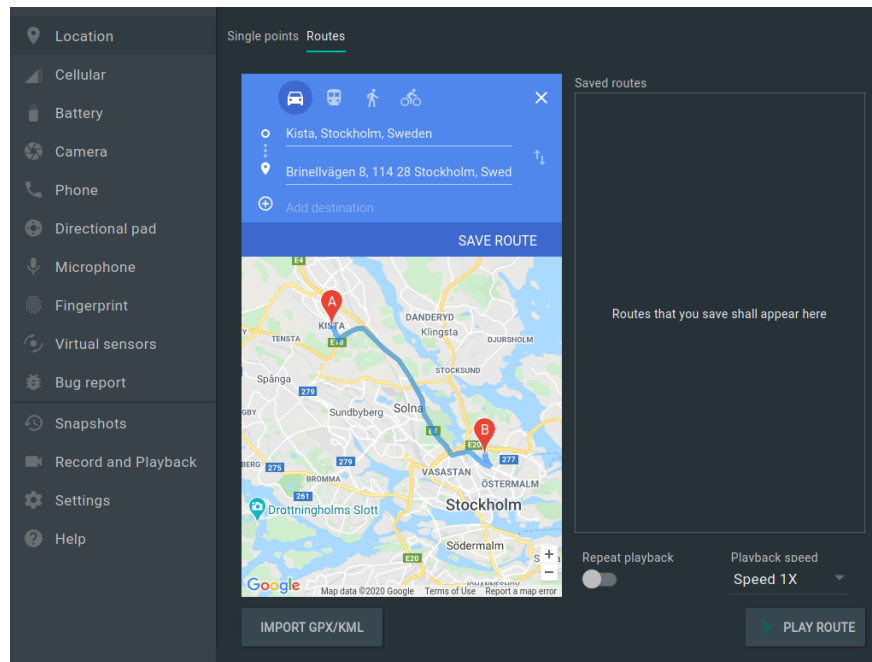
3 Android App Vulnerabilities

3.1 Intent Sniffing

Android uses Intents to facilitate inter-app communication as well as interaction between different components of a single app. However, when a component is initiated by using a broadcast intent, the data passed in the intent may be read by other apps. The scenario here is that a service in *LocationApp* reads the user's location, sends it to an activity in *LocationApp* in order to show the user's location on a map, and it also sends this data to an external app *WeatherApplication* to show the weather. In this task, you should implement an intent sniffer that captures the location data passed between the internal components of *LocationApp*, as well as the data passed between *LocationApp* and *WeatherApplication*.

Preparation:

1. Take a look at Broadcasts in Android and study the limitations of Implicit Broadcasts since Android Oreo.
2. Get familiar with the *LocationApp*'s `ForegroundLocationService`. Pay special attention to how it uses broadcasts for inter-app and intra-app communication.
3. For this task, you should have an active route in you emulator. Go to the emulator setting > location > Routes. Here you can search for different locations and then use directions button to create a route, just like in Google Maps application.



When working on this task, you should save a Route and Play it. This way you will get constant location updates.

Implementation:

1. Implement the functionality in *IntentSniffer* app to capture both inter-app and intra-app broadcasts, extract the location data from both and show it in `LatitudeTextView` and `LongitudeTextView`.

Report:

1. Describe how the broadcast system works, and the use cases of implicit broadcasts.
2. Explain your code and how you can get these broadcast intents.
3. Propose and discuss at least **two** solutions to overcome intra-app broadcast sniffing.
4. But you can also sniff inter-app broadcasts. Can you explain why?
5. What is your solution for inter-app broadcast sniffing?? Propose and discuss at least **two**.
6. Implement at least one of the solutions for each case and verify that the exploits no longer work.

3.2 Confused Deputy Attacks

A confused deputy attack happens whenever a benign but privileged program is tricked into mis-using its authority on the system. In this task you should implement the *NotesExploit* app in a way that it tricks the *Notes* app to modify its own database.

Preparation:

1. Read Felt et al.'s paper on permission re-delegation in Android.
2. Make sure you know what Intents are, specially get familiar with Explicit Intents.
3. Make sure you understand how the **Notes** app, specially the `DatabaseActivity`, works.

Implementation:

1. There are 3 methods in the *NotesExploit* app's `MainActivity`
 - `addClicked`
 - `removeClicked`
 - `showClicked`

These are button event handlers, each corresponding to a button in the *NotesExploit* app's UI. The first two handlers should send an Intent to the *Notes* app and trick it to add or remove an item from its database. The last handler should trick the *Notes* app to send it the text of a note in return. For remove and show, it is assumed that we know the ID of the note we want to remove or see.

Report:

1. Describe the functionality of `DatabaseActivity`, specifically explain how it modifies the database.
2. What is the root cause of the issue? Why can you modify the database contents?
3. There are several ways to fix this issue, propose at least **two** solutions for it.
4. Explain your solutions, specifically describe the pros and cons of each approach.
5. Implement at least one of the solutions and verify that the exploits no longer work.

3.3 Leaky Content Provider

Content Providers in Android act as a central repository to store data and facilitate sharing it with other apps. However, if a content provider is not implemented correctly, it can leak sensitive data. In this task we focus on leaky content providers. You should misuse the content provider of the *Notes* app in a way that allows you to read the contents of a text file in the SD-Card. Obviously, your exploit app should not have permission to read from external storage.

Preparation:

1. Make sure you understand the basics of Content Providers – we don't use the database-related methods here – pay special attention to `openFile` method.
2. Learn about Path Traversal Attacks.
3. For this task, you have to manually create a text file inside of your emulator's SD-Card. The easiest way is to create a `txt` file in your own computer and then use `adb push` command to push it to the emulator. (`adb` is inside your Android SDK installation directory, in a sub-directory called `platform tools`). *Note:* For this task you should directly query the content provider, avoid using URI permissions and/or the actions of `DataBaseActivity`.

Implementation:

1. Implement `queryContentProvider_onClicked` method inside *ContentProviderExploit* app's `MainActivity`. This method should use the Content Provider to read the text file which is inside of the emulator's SD-Card. It should show the returned text in the `resultTextView`.

Report:

1. Describe the usage of content providers, and why do you think we used one in this scenario?
2. What is the problem with this implementation, why can you access SD-Card through it?
3. Explain your implementation of `queryContentProvider_onClicked` method.
4. Propose and implement a solution for this issue and demonstrate that the exploit no longer works.

3.4 Content Providers and URI Permissions

In this task we focus on the combination of incorrectly implemented content provider and confused deputy attack. You should misuse the content provider of the *Notes* app in a way that allows you to read the contents of a text file in app's cache directory.

Preparation:

1. Make sure you understand the basics of Content Providers – we do not use the database-related methods here – pay special attention to `openFile` method.
2. Read about URI Permissions.
3. For this task, you have to create a text file inside app's cache directory. The easiest way is to just create a note in the *Notes* app and then try to share it. A temporary `SharedFile.txt` file will be created in the cache directory of the app.

Implementation:

1. Implement `accessContentProvider_onClicked` method inside *ContentProvider-Exploit* app's `MainActivity`. This method should use the Content Provider to read the text file `SharedFile.txt` which is inside the app's cache directory, and it should show the returned text in the `resultTextView`.
2. Can you combine this issue with the previous task (Leaky Content Provider) to read the text file stored in SD-Card? Does it work even if the provider is not exported?

Report:

1. Describe the usage of URI permissions in content providers.
2. What dangers the incorrect implementation of this feature have? What is the problem with the current implementation?
3. Explain your implementation of `accessContentProvider_onClicked` method.
4. Propose a solution for this issue and implement it (your solution cannot be removing the URI permission, imagine the app needs it for other functionalities). Demonstrate that the exploit no longer works.

4 Requirements and Grading

4.1 Report

A zip file to be submitted via Canvas that should include the following:

1. Source code to your solutions.
2. A detailed report answering the questions in the report section of each task. Specifically, you should fully explain the issues, the exploits, and the proposed solutions.
3. Clear statement of contributions of each group member.

4.2 Grading (Max 20 points)

- You get 5 points for completing each task in Section 3. Thus, you can get a total of 20 points.
- To pass the lab you should at least solve 2 of the tasks.
- This lab does **not** have any extra points.

In this lab special attention will be given to the **reports** and **presentation**. Please make sure that the report is satisfactory and every group member is able to explain the solution and answer questions during the live presentation of the lab. Failure to do so may result in deduction of points.