Imperial College London

COURSEWORK 2

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Advanced Computer Security

Authors:
Bianca Tazlauanu (CID: 01921394)
Theodoros Kokkoris (CID: 01347345)
Shreyas Annamraju (CID: 01339332)

Date: November 25, 2020

1 Part 1

1.1 Part 1: A.1

The `adb devices` invocation causes `adb` to list the currently attached devices, along with the **serial number** (a unique string which identifies the device by its port number). In our case we only had one emulated device, the result of the command being:

```
List of devices attached:
emulator-5554 device
```

1.2 Part 1: A.2

Before the introduction of `Advertising ID`, applications were capable of collecting data which is unique (personally-identifiable information) to a user's device (e.g. Android ID, IMEI), allowing them to track and profile users. By using the advertising ID, which is an anonymous, resettable string, applications can no longer do that, since once the advertising ID is reset, a new profile for the user has to be created and it can not be traced back to the actual user.

1.3 Part 1: A.3

Attempting to read the users' contacts without enabling the permission `READ_CONTACTS` throws a `java.lang.SecurityException`. To address this, the previously mentioned permission must be placed in the advertising library's manifest. In this way, the application developers are also informed of the data that the library is accessing.

1.4 Part 1: B.1

For the host application we granted more permissions to showcase how they can be used by a malicious ad library. The newly added permissions are: `READ_CALL_LOG`, `READ_CALENDAR`, `READ_SMS` and `READ_EXTERNAL_STORAGE`. All of these permissions are being requested by the Facebook application.

1.5 Part 1: B.2

We collect phone numbers, SMS messages, call duration and called number from the call log, and calendar names from the calendar. From the external storage we were able to access the gallery, to prove this we logged only the paths to the file, but a malicious library would be able to send the actual images through the network (since the ad library also needs network access permissions). All of this is sensitive private information and as such advertising networks should not have access to it without user consent. The collected data holds valuable information about the user's behaviour which could help ad providers target ads more efficiently, but this type of private data should not be made accessible for commercial use. Having access to this information poses a security and privacy risk since it shows the user's location, daily activities, information about their personal relationships, interests and more.

1.6 Part 1: B.3

When extracting information, we targeted data that could be used to build a user profile. The data collection algorithm is implemented in a second thread to allow us to get the location and advertising id. The thread is scheduled to run 2 seconds after the ad library is loaded to have enough time for the location listener to save the location. After that, the thread runs every 10 seconds to collect data periodically, even if the application is in the background. We iterate over all the contacts, call logs, SMS messages, calendars and photos from the device and log the collected data. Instead of sending the data over the network, we store it in the `Part1_malad.txt` file. If one of the data sources the ad library is querying for is missing from the device, a message will be printed in the logs to indicate that.

2 Part 2

2.1 Part 2: A.1

Since the ad provider's code is located at the following path `com/google/android/gms/ads`, we first used a `grep` command to search for the respective path in the smali files after decompiling the given APK using `apktool`. All the files follow the same pattern when interfacing with the ad library and we could pinpoint the call of the `loadAd` function which is necessary to load an ad in the application: `Lcom/google/android/gms/ads/AdView;->loadAd(Lcom/google/android/gms/ads/AdRequest;)V`. From the output of the `grep` command, we also got the location of the function definition which we later used to insert the malicious code, in the class: `smali/com/google/android/gms/ads/AdView.smali`.

2.2 Part 2: A.2 + B.1

The `loadAd` function is responsible for all the ads that are being loaded in the application, therefore we decided to insert the malicious code here. In order to print "Hello Malvertising!" with the `Log.d` function, we need to pass in two string constants, one with the tag and one with the actual message. For this we used `const-string` declarations and increased the local variables counter. After that we made the function call using `invoke-static` and moved the return value to one of the constants we previously declared. To find the IMEI number, we used an `invoke-virtual` call to `getSystemService` to get an object of type `TelephonyManager`. After that, we used it to call `getDeviceId` to get the IMEI number. We then made a `concat` function call to prepend "IMEI: " to the result and followed the same process as before to log the message.

2.3 Part 2: B.2

For this task, the IMEI number was collected from the device. This is sensitive information since it is personally identifiable and ad providers could persistently track the data coming from a specific device, tracing it back to the user. This can be useful to the advertising networks in building a more robust profile for each user and help them target ads better. Even so, this could pose a privacy risk and ad libraries should not have access to it. Moreover, a phone's IMEI number can be used to disable the phone's network access or to reveal information about the model and specifications of the phone. In the `Part2_malad.txt` file we store the log lines containing the "Hello Malvertising!" message and the IMEI number.

2.4 Part 2: B.3

If you try to install the repackaged application on a device that already has the original application installed, an error regarding "inconsistent certificates" will occur. The reason for that is that the original application is signed with a different private key than the one which was used to sign the malicious app, which we do not have access to.

2.5 Part 2: C

After the malicious code was inserted, the application was recompiled using `apktool` and signed using `uber-apk-signer`. To test the new application, we copied the APK to `platform-tools` in the `Android/Sdk` folder and used `adb` to install it on the emulator and get the logs. The only modification required to be able to install the two apps side-by-side is to change the `package` entry in `AndroidManifest.xml`. For this we replaced `package=com.skibapps.wiretapremoval` with `package="com.skibbapps.wiretapremoval"` using a `sed` command, changing the name of the package.

3 Appendix

3.1 Part 1 Extension

3.1.1 Strategy

In a broader sense, we sought to target permissions from which large volumes of data could be inferred. For example, an advertiser could construct a measure of how social a person is from number of SMS messages sent and average call duration, as well as number of calendar events. This would be a powerful metric for advertising.

3.1.2 Instructions

Placing a call on the emulated device is done by clicking "Phone" under the extended controls panel, selecting a number, and pressing the "Call" button. There is a similar process for sending an SMS message to the emulated device. For the calendar query, a sign-in into the Google Calendars app with a valid Google account is required. If no calendar exists, one must be created within the app.

3.1.3 Testing

In order to test the implementation, the following specifications were used for the emulator device: Pixel 2 API 21, Android 5.0 x86_64.

3.1.4 Collect User's Contacts

It would be possible to collect the contacts from the device using the ad library if *uses-permission android:name="android.permission.READ_CONTACTS"* was added in the manifest of the ad library even though it is not in the manifest of the main app.

3.2 Part 2 Extension

3.2.1 Strategy

In order to insert malicious code into the application we followed the following steps: decompile the APK, search for the function which has the information we want to extract, recompile the APK, sign the new version, install it in the emulator and test it. For example, to log the message "Hello Malvertising!", the following code was inserted in the `loadAd` function:

```
.locals 2
const-string v0,"log-tag:"
const-string v1,"Hello Malvertising!"
invoke-static v0, v1, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
```

3.2.2 Collecting More Data

Collecting the location and the advertising ID is a bit more complex; the former requires creating a `android.location.LocationListener` and the latter needs to be run in a separate thread, using `Executors`. Both can be easily coded in Java in a dummy project, compiled to smali and then copy-pasted to our own project (of course, minor modifications with respect to registers and such will be necessary).