Imperial College London

Department of Computing

COMP70004

Advanced Computer Security

Coursework 2: Mobile Advertising - AdWare or Malware?

Team Members:

Salim Al-Wahaibi (saa221)

Maxim Fishman (maf221)

Marcos-Antonios Charalambous (mc921)

29/11/2021

1 Part 1A

Q1: The *adb devices* command lists all the emulators/devices currently running that are attached to the host computer.

Q2: Google introduced the Advertising ID because it cannot be connected to Personally Identifiable Information (PII) and is resettable by the user. During the reset, the new Advertisement ID is completely disconnected from the old ID and its associated data. The user can also manually set the Advertisement ID to reflect their preferences.

IMEI should be kept confidential because it could be used to grant access to location and call logs. Therefore, it is a security risk to allow advertising libraries to use this sensitive information.

Q3: When trying to access the user's contacts, a *java.lang.SecurityException: Permission Denial* error occurs. This error occurs because the app does not have the *READ_CONTACTS* permission, which is required to read user contacts. To get around this barrier, the *READ_CONTACTS* permission can be added to the *AndroidManifest.xml* file in the Advertising Library.

2 Part 1B

Q1: The host app was changed by adding the camera permission *android.permission.CAMERA* in the host app's *AndroidManifest.xml* file. A real-world app that uses this permission is WhatsApp. WhatsApp uses the camera when scanning a QR code for adding contacts. This permission can be found here: WhatsApp

Q2: One of the goals of an advertising library is to showcase products that resemble users' interests. Therefore, we tried to collect sensitive data that would reveal users' personal lives and preferences. The data collected can be found below:

- All *processes* that are currently running on the device. This information could reveal the user's shopping preferences, especially if they use store-based apps, such as Nike. The user should be able to give consent for this data to be harvested by the advertisement network.
- *Network Adapter ID.* This is a unique hardware ID that cannot be changed by the user. Therefore, the user cannot disassociate from the data that was collected. The identifier should not be used by the ad library as the hardware ID is unique and the user can not reset it or set it to private.
- The *IP address and Network Routing Table*, which displays where the user is and all their connected interfaces in the local network. This information by can be used by advertising networks to promote region-specific adverts. This feature should not be available to the advertising networks as it can directly point to the user and get their location.
- *Network and SIM operator.* This data reveals the user's network and phone carrier. Thus, it can be used to generate more applicable network and phone plan advertisements. The library should only use this feature if consent is requested from the user.
- The user's *phone number* and any *ongoing call phone number.* The ad library can collect phone numbers and store them in a database, thus expanding the advertisement audience. This feature should only be used if the library informs the user and requests their consent.
- *Taking pictures without consent and without being noticed.* The library can use computer vision techniques to examine the user environment and promote more user-specific ads. The advertisement library should be strictly not allowed to use such a feature of the user as it violates the user's right to privacy.

This type of information is useful for advertising networks. As explained above, this data can help advertisement networks promote more targeted advertisements, and subsequently, boost their profits. However, adverts should adhere to the least-privilege principle and request only the necessary permissions from the user, regardless of what the host app allows. This principle will ensure that the user's privacy is not violated in the process of promoting curated ads.

Q3: Data collection is done by extracting data from readable files on the device and taking advantage of the permissions given to the host app. To retrieve process information, the *"top -n 1"* command is used, printing all of the running apps along with their CPU utilisation. To extract the network information, the world-readable file */proc/net/arp* and */proc/net/fib_trie* is used. For the caller number and status, the *CALL STATE* privilege was added to the host app. The camera privilege is used for taking pictures without notifying the user. The code for this part will run automatically once the host app is loaded.

3 Part 2A

Q1: We first viewed the code to get an understanding of the logic flow and interaction between the host app and the advertising library. To make this process easy, we used jadx, which decompiles Dex to Java. Although the code produced is obfuscated, it is still easier to read than the Smali code produced by the *apktool*. We then inspected *AndroidManifest.xml*. In this file, there was a *MainActivity* file that was set as the main starting point of the host app. We started searching the *MainActivity* file for keywords such as ad and loadAd, attempting to find a method in which the advertisements are loaded. We noticed that the starting point of the app is in the onCreate method, where an ad is built and loaded via the *loadAd* from Google's *AdView* class.

Q2: First, we used the *apktool* command to decompile the APK file. After detecting the method that loads the advertisement, we proceeded to inject the malicious code. This code needs to be written in the decompiled intermediate language, *Smali*. To print this information to the terminal, we needed to find Smali's equivalent to Android's *Log* functionality. After injecting the code and re-compiling the app back to an APK, the APK was aligned using *zipalign*.Then, we signed it with the *apksigner*. Finally, the *"Hello Malvertising!"* is printed in the logs when the app starts.

4 Part 2B

Q1: The malicious code was embedded into the advertising library by first de-compiling the APK package into the *Smali* intermediate language. Similarly to Part 2 A, jadx was used to examine the logic flow of the package in Java and find a suitable place to inject the code. From there, we managed to find the same *loadAd* method in the *Adview* class that is responsible for loading the advertisement, and serves as an injection point for malicious code and sensitive data collection.

Q2: The sensitive information gathered are:

- *Network Information (IP address, network adapter hardware identifier).* These values are sensitive since the user can be tracked and can be used to backtrack to the vendor of the specific hardware (i.e., Android, Apple).
- *Phone number* The phone number is sensitive because it directly identifies the user.
- *Location of the user in terms of longitude and latitude.* The information is sensitive because it tracks down the user location and builds a database of their movement.
- *Phone State (ringing, ongoing call, idle) and ongoing caller number.* This information is sensitive as it tracks the user callers and the time of the call. The ongoing caller and current phone number can be used to collect a database of phone numbers to advertise by SMS and attract more users.
- *IMEI hardware identifier.* The value is sensitive as the identifier is unique to each device, forbidding the user to control their ad-tracking preferences. The IMEI can also be cross-referenced in different apps that use the same ad library and able to locate the user.
- Read other *processes* running in the phone identifies other installed apps along with CPU utilisation for each process. From an advertisement library's perspective, this data provides different combinations of apps and products the user is interested in.

As explained in question 2 of Part 1B, advertisements should only request the minimum amount permissions to perform its functionality.

The *Part2_malad.txt* file consists of log dumps, split into 2 parts: part A and part B. In part A, the "Hello Malvertising!" string is preceded by a timestamp (date & time) of when this data was collected. In part B, all the sensitive information is logged, each preceded by a timestamp of collection.

Q3: When trying to install the repackaged app with the original app already on the device, an error is raised stating that the certificate for the repackaged app is invalid, since it has a different signature compared to the original app. This is happening because each Android app has a package name, which defines the Java namespace. Two packages of the same name cannot exist on the same device as overlapping namespaces will be created.

5 Part 2C

Q1: To overcome the above issue and have both versions of the app installed on the device, we changed the package name of the repackaged app in the first line of the Android Manifest file. The apps can now co-exist and are indistinguishable. The script first decompiles the given APK using the *apktool* command. Then, the script enters the decompiled directory and replaces the package name in the first line of the manifest file. Finally, the app gets recompiled and uses the given signature to sign the repackaged app. This script is written in Python 3 and is compatible with Windows, MacOS, and Linux.