Homework 10: Mobile security

In this exercise, you will learn how to reverse engineer an Android APK file using JADX and extract useful information from it.

The provided APK files have been obfuscated to a certain extent. They are designed to accept a string input, referred to as the "flag" or "pin code", and indicate whether it is correct or not. Your task is to reverse-engineer the source code of the APKs and determine the correct "flag" by analyzing its contents. By the end of this exercise, you should be able to identify the correct "flag" and verify its validity.

Setup

- You need to have Java installed, any version from the last 10 years should be fine.
- You need JADX. Download the latest version from this link.
- (Alternative) If you don't like Java or JADX, you can always use an online decompiler such as the one in this link. Disclaimer: we did not test the above online tool. We recommend running `jadx` yourself if possible.
- (Optional) Installing the JADX-gui can help but it's not necessary to solve this homework.
- (Optional) an Android device or Android emulator with side-loading enabled to test the apks yourself. It is *not* required to solve the homework.

Step 1: decompile the APK

Navigate to the folder where the APK file you wish to decompile is. Then, run: `jadx <apk>`. This command will create a new directory with the decompiled Java code. You can then navigate the decompiled code with the editor of your choice.

If you have the GUI version installed, you can also run `jadx-gui <apk>`. This will open an interactive code browser that lets you read the decompiled code.

Step 2: Get familiar with the apk format

As you saw in the lecture, an apk is not much more than an archive containing various types of files an android app might need. When analyzing an apk, `jadx` will create two folders:

- `sources`: This directory contains the decompiled Java, and this is where you'll want to spend most of your time in.
- `resources`: Everything else that composes an apk. In particular, it's divided in the following:
  - `AndroidManifest.xml`: This is the main configuration file for an Android app. It contains information about the app's package name, version, permissions, and other settings.
  - `classes.dex`: This is the compiled code of the app written in Java. It contains the bytecode that can be executed by the Android runtime.
  - `res/`: This directory contains the app's resources organized into subdirectories based on their type (e.g. drawable, layout, values).
  - `assets/`: This directory contains any additional files that the app needs, such as external libraries or custom fonts.
  - `META-INF/`: This directory contains metadata about the APK file, such as its signature and other information.
  - `lib/`: This directory contains any native libraries that the app uses, such as libraries written in C or C++.

Particular interest should also be given to the directory `resources/res/values`: It contains files that define various values used by the app. These files are typically written in XML format and can be accessed by the app's code.

Some common files that you may find in there are:

- `strings.xml`: This file contains the app's strings, such as the app's name, labels, and messages.
- `colors.xml`: This file defines the app's color values, and can be used to set the app's color scheme.
- `dimens.xml`: This file defines the app's dimension values, such as the size of text or margins.
- `styles.xml`: This file defines the app's styles, which are sets of values that can be applied to views in the app's layout.
- `attrs.xml`: This file defines the app's custom attributes, which are values that can be used in the app's layout files.
- `public.xml`: This file contains entries that define the names and IDs of the app's public resources.

Step 3: Start reversing!

All the apps included in this homework will ask the user for a *flag* or a *PIN*. Your job is to analyze the code of the app to understand what's the correct input the app expects.

However, the developer of those apps wanted to keep the correct input a secret, so the verification if your input is correct is not a simple string comparison.

Try to reverse engineer the code and understand what it is doing!

Step 4: Checking the flag

Once you have obtained the valid flag, you can either run the actual app on your phone/emulator and input it manually, and the app will tell you if your guess is correct.

If you do not have an Android device or emulator at hand, you can submit your flag to the following URL http://hexhive006.iccluster.epfl.ch:8001/<challenge>/<flag>

For example, if the correct flag for the *test* challenge was `MOBISEC{test_flag}`, you can verify it by visiting http://hexhive006.iccluster.epfl.ch:8001/test/MOBISEC%7Btest_flag%7D

Do not worry about URL-escaping the curly brackets, most browsers will do that for you.

Challenges

- *babyrev*: Starting challenge. Look for a function called `checkFlag`. You need to understand which string satisfies all checks to get the correct flag.
- *pincode*: Look for a function called `checkPin`. This time you need to provide a 6-digit PIN code. The check is a bit weird though, right? This challenge requires a bit more than just reverse engineering.
- [OPTIONAL] *gnirtz*: Here you have the `checkFlag` like in the first challenge. This time getting the flag will be considerably harder. Some light obfuscation is present. Only for motivated students.

HINTS, TIPS AND TRICKS:

- You may want to give a closer look at the above mentioned `public.xml` and `strings.xml`.

- Educated guesses are a key skill for reversing. It is very often not required to understand every small detail of how a certain piece of code works. Do not follow the rabbit too deep into the hole!
- Make sure you take notes of the things you are confident you understood.
- You do not need to understand how md5 works to solve *pinhole*.
- Are 6-digit PINs really safe?
- If you can't get your solution for *pincode* to work, you can always craft another apk or copy and modify the Java code somewhere. Make the check simpler and verify that your solution works.

Credit:

This homework was adapted from Prof. Fratantonio's MOBISEC course from EURECOM. We thank him greatly for sharing some of his introductory material with us. If you liked those challenges, go check it out at this link!