CO447
Coursework 2: Mobile Advertising - AdWare or Malware?

Release date: Tuesday, 11th of November, 8:00am
Due date: Tuesday, 25th of November, 11:59pm

Course Website: `https://co447.doc.ic.ac.uk/`
Submission: CATE (`https://cate.doc.ic.ac.uk`)

Overview

Mobile advertising has become a major source of income for mobile application developers. Rovio, the company who developed Angry Birds, reported a 1 million dollars per month profit from its Android app back in 2010 when it was boasting 8 million installations. Now Angry Birds has at least 100 million installations on Android devices. Nevertheless, more often than not, mobile advertising comes at the expense of user privacy. For the advertising companies to be able to offer targeted, and thus more meaningful advertising, they benefit from profiling users in various ways. A vast array of data brokers such as Google—through AdMob—take on the task to build detailed user profiles. This way they can offer advertisers the capability to target specific groups of users according to a multitude of demographic information. However, with the current model of mobile advertising, there is nothing stopping a data broker or an advertiser, from aggressively collecting more and more user information.

In this assignment you will explore what a malicious advertising network can do to exploit the current model and learn as much as possible for a device user.

Please read the assignment carefully before starting the implementation.

Assignment Parts

This assignment is composed of two parts. In Part 1, you will learn how a library can be embedded in a host Android app and take advantage of the host app's privileges to infringe on a user's privacy. In Part 2, you will explore a real world app with embedded advertising and learn how to inject code in it and re-package it.

Part 1

Part A of this section, will guide you through running an Android app with an embedded library that captures some of the device user's information. In part B you will enhance your malicious library to capture more sensitive information.

Part 1: A

In this part you will be given a wrapper for a dummy Android app (under the dir **Part1/Code**) that requests the following permissions: `INTERNET`; `ACCESS_NETWORK_STATE`; `ACCESS_FINE_LOCATION`; `READ_PHONE_STATE` and `WRITE_EXTERNAL_STORAGE`. You will also be given a wrapper for an Android library (under the dir **myadlibrary**). The latter has an unimplemented method called **onLoad()** which the Android app is calling. Your job is to link that library with the Android

app and run it on an Android emulator.[1] You are not allowed to change the code of the Android app. Assume you can only change the library code. Try to read the user's location, the device's IMEI number, the user's contacts and the advertising ID. To avoid having a back-end implementation and having you send the acquired information there through the Internet, you should save the acquired information in a file accessible from the terminal.

The implementation blueprint below provides detailed information on the requirements of Part 1.

Implementation Blueprint

1. **Set up the environment:** Download the Android Studio or Android SDK. You can follow the instructions on: `http://developer.android.com/sdk/index.html`. If you are unfamiliar with Android development you should download the Android Studio as it contains everything you need to get started. From this point forward we will assume use of Android Studio although the process would be very similar with a different development environment. Install and launch Android Studio. Then open the `Android SDK Manager` and install everything under `Android 5.0` and `Android Support Library`, `Google Play Services` and `Google Repository` located under Extras. This might take a while so you might want to grab a cup of coffee or tea.

2. **Set-up an emulator and run it:** Create an emulator with an AVD (Android Virtual Device) that runs the Google APIs platform based on Android 5.0 API 21. Android Studio comes with built-in emulated devices. Launching the emulator might take time. Hence you should consider not closing it every time. Note that you will need a 64-bit OS to run the emulator "out-of-the-box". If you are working on a 32-bit OS you can launch your desired emulated device form the command line: `/path/to/Android/Sdk/tools/emulator -netdelay none -netspeed full -avd myNexus5 -force-32bit`. Once you have the emulator up and running, try using **adb** (Android Debug Bridge). More here: `http://developer.android.com/tools/help/adb.html`. Try using the following command from a terminal:

   - *adb devices*

   What does it do?

3. **Run the given app and library:** At this point you should have the emulator up and running. Import the given app and library into Android Studio. Android Studio projects are linked to the Android SDK. The Android Studio should ask you to re-link the project to your sdk. You can always change this through Gradle Scripts → local.properties. Next, link the given library to the given app at your IDE (if you are using the given Android Studio project as it is, the library would be already part of the project). Launch the app on the emulator. You should see a message (*Hello Ad!*) as soon as the host app calls the library's `onLoad()` method. Try changing that message. (*Here we have simulated the ad library functionality. You can read more on how you can integrate real ads in an app here:* `https://developers.google.com/mobile-ads-sdk/docs/admob/android/quick-start`).

4. **Get the device's location:** If you are using the emulator, you will have to set up a simulated location. You can do it through the GUI of the emulator or through the command line. Here's how to do it programmatically:

   - **Grab the emulator serial number:** Issue the adb command and grab the integer number associated with the emulator from the result (e.g. 5554).

[1] If you own an Android phone you can use that if you wish.

- **Telnet to the emulator:**
  telnet localhost "serial_number"
  You should be connected to the Android console now. Sometimes you might need to authenticate to your device before issuing the telnet command. Your auth token should be on a file ∼/.emulator_console_auth_token. Use that to *auth "auth_token"*.
- **Feed the emulator a simulated location:**
  geo fix "longitude" "latitude"

Enhance your library to grab the GPS location.

5. **Grab the IMEI of the device:** The IMEI is a unique identifier for the device. Advertisers grab this to identify adRequests and information gathered from the same device. Enhance your library to grab this number as well.

6. **Grab the Advertising ID:** Developers are now instructed to use the Advertising ID instead of the IMEI as a device identifier when that is being used for advertising:

   > "As a reminder, please note that starting 1 August 2014, new apps and app updates distributed through Google Play must use the advertising ID in lieu of any other persistent identifiers for any advertising purposes, on devices that support the advertising ID."

   Grab the Advertising ID too. More here: `https://developer.android.com/google/play-services/id.html#get_started`. Why did Google introduce the Advertising ID?

7. **Try to read the user's contacts:** Without changing the host app or its manifest, try to read the user's contacts. What happens?

8. **Store the acquired information:** An ad network would have sent the gathered information to its backend for enhancing the user's profile built. Since you don't have access to such a network's backend, and to avoid creating one, you can store the device's location, IMEI and advertising ID to a file. Create a file called *Part1_malad.txt*. Append one line per piece of information gathered as below:

   - *timestamp;longitude:-88.207270;latitude:40.110588*
   - *timestamp;IMEI:xxxx*
   - *timestamp;advertising_id:38400000-8cf0-11bd-b23e-10b96e40000d*

   Store the file on the external storage. On your laptop/PC issue the **adb pull** command to get that file from the emulator. Attach this to your deliverable.

Report Questions - Part 1 A

1. What does **adb devices** do?
2. Why did Google introduce the Advertising ID? (What's wrong with the IMEI?)
3. What happens when your library attempts to read the user's contacts? How can you handle this? (Assume you don't have access to the host app's code when developing the library)

Part 1: B

For this part of the assignment, try to get whatever you can from the device. You should get at least one piece of sensitive information that is different from the Part 1 A requirements. You can change the host app to request more permissions if necessary, but only if you find a real-world app that requests the same set of permissions.

Report Questions - Part 1 B

1. What changes did you make to the host app? If you have requested more permissions, cite a real-world app that requests those permissions.
2. What kind of data are you collecting? Why is this sensitive data? Should advertising libraries be allowed access to it or not? Do you think this type of information could be useful to advertising networks?
3. Explain your data collection algorithm/strategy and provide any instructions/requirements to run your code.

Part 2

In this section, you will act as a real-world adversary. A lot of malware, are actually re-packaged apps: popular apps are being de-compiled, injected malicious code, re-compiled and then submitted in third-party application stores. Users download such apps from untrustworthy stores usually because the official store is either unavailable in their countries, or because the re-packaged app unlocks some premium features. On part A you will go step-by-step through such a process, excluding the publication of the app for obvious reasons. In particular you will spot the advertising library and enhance its code to print a message. On part B you will embed actual malicious code in the advertising library of the victim app. This is some sort of mobile `malvertising` where the malicious payload is distributed through valid advertising networks.

Part 2: A

In this part, you will embed your code on a real-world ad library. You are being provided with a real app apk (`base.apk`) from GooglePlay, which uses Google's AdMob library to display mobile advertisements. Your goal is to **decompile** the app, and change the advertising library's method that the host app is calling to display ads. Once you embed your code you can **re-compile** the app and run it on the emulator. In this part you will just print a message from the real library and store it in a file (*Part2_malad.txt*).

You are free to accomplish the above in any way that you want. You can use the guideline below if you wish, but you don't have to. In fact you can attack your own real world app if you wish as long as it uses mobile advertising and your code runs from within the advertising library.

Implementation blueprint

1. **Test the given apk (or your own app apk)**. Install and run the apk on the emulator.
2. **Decompile the given apk (or your own app apk):** You can use the **apktool** to decompile an Android application package (see: `http://ibotpeaches.github.io/Apktool/`). You will end up with the smali code of the app. (`http://code.google.com/p/smali/`)
3. **Embed your code in the advertising library:** Locate the relevant smali file of the advertising library that is being called whenever the host app requests an app to be loaded. How did you locate the file? Enhance that code to print a message in the logs (see `http://developer.android.com/reference/android/util/Log.html`). The message could be: **Hello Malvertising!**
4. **Re-package the app:** Use apktool to compile the app with your embedded code, and run it on the emulator. Note that you will need to sign the app. You can use **adb install** to push a compiled and signed app on a device. You can also use **adb logcat** to view the logs on a terminal. Copy the log dumps relevant to your malicious code and paste it to a file named *Part2_malad.txt*. Explain the format of the file in your report and include the file to your deliverable.

Report Questions - Part 2 A

Describe your approach in embedding your code to the advertising library.

1. How did you locate the advertising library's smali file and method being called whenever an ad needs to be loaded?

2. What's your approach in embedding the malicious code in the advertising library?

Part 2: B

For this part of the assignment, try to enhance your re-packaged app to steal sensitive user information. You can get whatever information you want, as long as you can argue that this is private information. You can again use the logs to dump the extracted information. You can then copy-paste from the logs to *Part2_malad.txt*. Explain the format of the file in your report.

Report Questions - Part 2 B

Describe your approach in embedding your code to the advertising library. Moreover please describe what data you are collecting and how you are collecting them.

1. What's your approach in embedding the malicious code in the advertising library? (Could be the same as Part 2 A.)
2. What kind of data are you collecting? Why is this sensitive data? Should advertising libraries be allowed access to it or not? Do you think this type of information could be useful to advertising networks? Describe the format of *Part2_malad.txt*
3. What happens if you try to install the repackaged app while the original app is already installed on the emulator/device? Why is this happening?

Part 2: C

Can you amend your repackaged version so that it can be installed on a device that already has the original (benign) version installed? Try to write a script to automate the repackaging process. Include your script in the Part2_source.zip deliverable.

1. Describe your repackaging script. Which parts of the code you had to change to be able to co-install the repackaged version on the same device as the original apk?.)

Evaluation

The breakdown of the evaluation is as follows:

- Part 1 – 50%
    - Part 1 A – 25%
        - Correct code – 10%
        - Report question 1 – 5%
        - Report question 2 – 5%
        - Report question 3 – 5%
    - Part 1 B – 25%
        - Correct code – 10%
        - Report question 1 – 5%
        - Report question 2 – 5%
        - Report question 3 – 5%
- Part 2 – 45%

Part 2 A – 10%

Report question 1 – 5%

Report question 2 – 5%

Part 2 B – 15%

Report question 1 – 5%

Report question 2 – 5%

Report question 3 – 5%

Working Part2_repackaged.apk – 10%

Part 2 C – 10%

Report question 1 – 5%

Repackaging script – 5%

- Submission of all deliverables as requested – 5%

Report

As part of this assignment you must write a report in 11pt font. The report should be at most 2 pages. The report must explicitly answer the report questions listed in Part 1 A, Part 1 B, Part 2 A and Part 2 B. Name the report file "Report.pdf", and put your team members' names and logins at the top of the document. See **Submission** for details on where to place your report.

Contest

On Part 1 B and Part 2 B, we asked you to think as an adversary and try to steal private information in creative ways. The most creative hackers (i.e., the algorithms and techniques which employ the most interesting attack strategy) will not only earn lifelong "bragging rights" but also an honourable mention on the module's website! You may opt-out of this contest by stating it clearly at the top of your report. No extra credit or bonus points can be obtained by participating.

Submission

You must submit 8 deliverables on CATE by the submission deadline as shown below:

- **Report.pdf**: Your report.
- **Part1.apk**: The compiled executable of the final app with your modifications for Part 1. It should run on an Android emulator running Android 5.0 API 21.
- **Part1_malad.txt**: The output file of your Part 1 malware.
- **Part1_source.zip**: The source code of Part 1. This should be the given Part 1 source code including your code additions.
- **Part2_original.apk**: The apk of the real-world app you've chosen to attack for Part 2.
- **Part2_repackaged.apk**: The apk of the repackaged real-world app that you have used. It should be able to be installed and execute on an Android emulator running Android 5.0 API 21.
- **Part2_malad.txt**: The output file displaying the grabbed information by your Part 2 malware.

- **Part2_source.zip**: The source code for Part 2. This should contain only the files which you've modified to repackaged the original base.apk and the files which you've added or modified.

We will obtain your submission through CATE at 11:59pm the day of the deadline.

Late Submission. No submission will be accepted after the designated deadline.

Good Luck!