CSE 484 / M 584: Lab #3

Mobile Security Lab

Due: **June 3, 2016 at 8pm**

Goal

The goal of this lab is to give you hands-on experience with Android security concepts.

Admin

- You may work individually, or in groups of up to 3 people.
- You should not need to write a large amount of code for any part of this lab.
- Please submit the required files for each part of the lab (described below) to the Catalyst dropbox. **Please also submit a txt file containing your group members and your group name (your group name will be used in parts 1 and 2).**

- Number of parts: 3 required (+ 1 extra credit)
- Total points: 80 (+ 20 extra credit)

Part 0: Setup

For this lab, you must set up an Android development environment. We recommend using Android Studio. (You may use something else if you prefer, though we have only tested the lab with Android Studio.)

Don't procrastinate on setup! Make sure that you can get a development environment setup ASAP.

Detailed setup instructions:

1. Download and install Android Studio. Just use the recommended setup option in the setup wizard.
   (If you already have Android Studio installed, or want to make sure your settings match ours, you can go to the Android Studio Preferences->System Settings->Android SDK and verify that Android Marshmallow (6.0), API Level 23 is installed.)

The rest of these setup instructions are best read in parallel with starting Part 1:

2. The apps to which we provide source code in this lab are set up as Android Studio projects. You should just be able to open them as existing Android Studio projects.

3. When you have a project open, click the green arrow to run the application.

4. The first time you try to run an application, Android Studio will tell you it can't find a device on which to run it. From the dialog that pops up when you try to run your first app, click "Create New Emulator", select "Galaxy Nexus", select "Lollipop (API Level 22)" and finish the wizard.
   (You're welcome to try using a real Android device, but the lab has been tested and these instructions assume use of an emulator.)

Android app development:

Not required, but if you have never written an Android application before, you may wish to check out this tutorial: Building Your First Android App.

Part 1: Exfiltrating Information

In this part of the lab, you will modify an existing application to secretly exfiltrate data.

Goal 0: *Create a simple web application that can store information sent to it.*

Create a web application that will store to a text file the values provided via URL parameters (hint: you did this for lab 2). Specifically, your application should take parameters “lat”, “lon”, and “groupname”. This will be useful in completing further tasks.

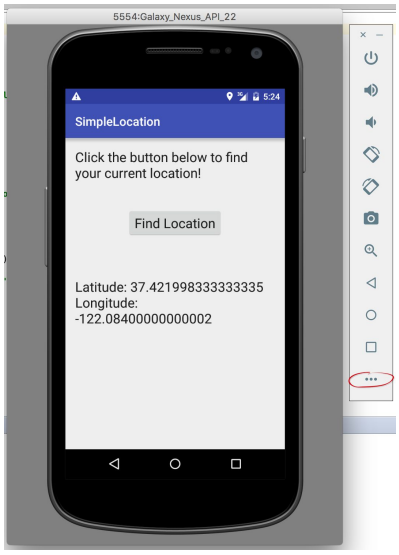
Goal 1 [20 points]: *Modify the SimpleLocation application to secretly access location information at a time not expected by the user, and exfiltrate that information to your server.*

1. Download and unzip the SimpleLocation application’s source code here: SimpleLocation.zip. This is a very simple Android application that prints out the current location on a button press.

2. Open the SimpleLocation application in Android Studio. Take a look at the source files by navigating through the project directory in Android Studio.

3. Without making any changes, run the project. You can do this by clicking the green arrow in Android Studio. **As described in “Part 0: Setup” above, the first time you do this, you will need to create a new virtual device (aka emulator).**

4. You’ll notice that the location is always null (unless you’re using a real device instead of an emulator. To give the emulator a fake location, click the “...” button circled in the screenshot below (bottom right):



5. **Your task [20 points]:** Modify this application to secretly access the device's location at unexpected times (i.e., when the user has *not* pressed the "Find Location" button), and exfiltrate that location to your web application (which you set up in part 0, above).

   When/How to Exfiltrate: You can be creative about how/when you send the location to your server, as long as it's NOT when the button is pressed. For example, you could do it when the app is first launched. (Don't wait too long, however -- when we grade your app, we won't run it forever :) Within 10 seconds of startup is a good cutoff.)

   Where to Exfiltrate (Important!): To exfiltrate location information, you should make a network request of the following form:

   `http://homes.cs.washington.edu/~username/loc.php?lat=47.6&lon=122.3&groupname=cookiemonsters`

   Notice that LocationActivity.java has a variable called `BASE_URL_TO_LEAK_LOCATION`. You should set this variable to the *base* URL to which you are leaking, i.e., http://homes.cs.washington.edu/~username/loc.php in the above example. When you test your app, you should make `BASE_URL_TO_LEAK_LOCATION` point to a server that you control. **However, when we grade your app, we will change it to a server we control. So make sure that your apps uses the `BASE_URL_TO_LEAK_LOCATION` variable when it constructs the path to which to exfiltrate the location information, and make sure that you use the parameter values specified above ("lat", "lon", and "groupname").**

   INTERNET Permission: To achieve this goal, you will need to add the INTERNET permission to the application's manifest -- otherwise you will see a runtime permission error. Don't worry, most users just click "okay" to the whole list of permissions!

6. **What To Turn In:** A zip file of your SimpleLocation source code (the whole Android Studio project).

Part 2: Permission Redelegation

In this part of the lab, you will modify your previous attack so that it doesn't require INTERNET permissions. You'll do this through a *permission redelegation* attack.

Goal [30 points]: *Create a version of your SimpleLocation application that performs the same attack as in Part 1 (exfiltrates information), but does so **without** the INTERNET permission.*

1. Download and unzip the SimpleLocation2 application's source code here: SimpleLocation2.zip. This application is exactly the same as SimpleLocation, but it's a fresh copy. (You will need to turn in separate solutions for Parts 1 and 2).

2. Download and unzip the SimpleGame source code here: SimpleGame.zip. (You might stop to laugh at how ridiculous SimpleGame is. Don't worry, its security is just as good!) Inspect this application's source code. This application has INTERNET permission, and is vulnerable to a permission redelegation attack.

3. **Your first task [10 points]:** Identify the permission redelegation vulnerability in SimpleGame. In a text file, identify the location(s) of the vulnerability, and briefly describe how another application can exploit it.

4. **Your second task [20 points]:** Modify SimpleLocation2 to secretly access the current location (you can reuse that part of your code from Part 1), and exfiltrate it to your server *without* using the INTERNET permission. Instead, find a way to leverage the SimpleGame to send information back to your web application (from Part 1, Goal 0).

5. **Important:**
   - **Do not modify SimpleGame.** We will be testing your SimpleLocation2 app with a clean copy of SimpleGame. We've only given you SimpleGame's source code so that you could find the vulnerability.

   - As in Part 1, we'll be grading your app by changing `BASE_URL_TO_LEAK_LOCATION` to point to a server that we control. **So make sure that your app uses the `BASE_URL_TO_LEAK_LOCATION` variable when it constructs the path to which to exfiltrate the location information, and make sure that you use the parameter values specified above ("lat", "lon", and "groupname").**

6. **What To Turn In:** A txt file with your answer to the question in bullet #3 above, and a zip file of your SimpleLocation2 source code (the whole Android Studio project).

Part 3: Encryption

In this part of the lab, you will explore a vulnerability in how Android applications might use encryption.

Goal [30 points]: *Extract the encryption key from an Android application (without access to source code).*

1. Download SimpleNotepad application here: SimpleNotepad.apk. This application lets users write notes, store them, and retrieve them. Because the developer knew that users might write private things in their notes, he/she decided to *encrypt* those notes before storing them on the device.

2. **Your first task [10 points]:** Find the encryption key used by SimpleNotepad, and briefly describe (one paragraph) how and where you found it.
   Hint: You don't have any source code! :(But check out APKTool, which lets you decompile Android applications.

3. **Your second task [10 points]:** Identify at least one other way in which the developer of SimpleNotepad is using encryption insecurely.

4. **Your third task [10 points]:** Briefly describe (one paragraph) why it's a problem that SimpleNotepad's encryption key is hard-coded into the app, and explain what the developer of SimpleNotepad should have done instead.

5. **What To Turn In:** A txt file containing your answers to bullets #2, #3, and #4 above.

Useful resources:

- APKTool: http://ibotpeaches.github.io/Apktool/
- http://tozny.com/blog/encrypting-strings-in-android-lets-make-better-mistakes/

Part 4: Forensics [Extra Credit]

In this part of the lab, you will analyze data dumped from two (unencrypted) Android phones.

Goal [20 points]: *Extract information from the data dumps from two Android phones.*

1. Two persons of interest were careless in keeping track of their phones. One of our spies has risked his life, but was able to dump data from their phones. These data dumps have been delivered to you for analysis. Get the the data dumps here: phones.tar.gz.

2. *Hint:* You will likely find a SQLite browser such as this one helpful for this part of the lab: http://sqlitebrowser.org/

3. **Your task [1 point each, up to 20 points total]:** Extract information from the phones:
    a. Phone number of the phone
    b. Name and Gmail account of user
    c. Phone information
        i. Model
        ii. Brand
        iii. Name
        iv. Device
        v. Board
        vi. CPU
        vii. Manufacturer
        viii. IMEI
        ix. Serial Number
    d. Operating System
        i. Type
        ii. Version
        iii. Build date
    e. Carrier
    f. List of applications installed
        i. Which came with the system?
        ii. Which did the user install?
    g. Phone numbers called from this phone
    h. Email addresses of people emailed from phone
        i. What are the contents of the email messages?
    i. Numbers to which SMSes have been sent
        i. What are the contents of the SMS messages?
    j. Recover the list of Wifi Networks to which the device has connected, if there were any passwords used, recover these

k. What method does the target use to unlock their phone (PIN, pattern, password, etc...)? Is there an alternate mechanism? What is it? If there is a password salt, what is it?
l. What was the voice volume level?
m. What was the alarm volume level?
n. What was the screen timeout set to?
o. Was the wifi on?
p. Was bluetooth on?
q. If the phone has a second factor application installed (like Google Authenticator), extract all of the key material.
r. Crack the screen unlock mechanism (i.e., crack the password/pin/pattern)
s. Location:
    i. What locations has the target been to?
    ii. What terms does the target search for with respect to locations?
t. Media
    i. Can you recover any images taken by the camera(s) or downloaded from the internet? (images that are part of the OS or downloaded as part of an application don't count).
u. Wifi MAC address
v. Bluetooth MAC address
w. Anything else you think might be useful

4. **What To Turn In:** A txt file containing all of your answers.