CS155 Project 3

Spring 2015

CS 155 Staff

May 21, 2015

Due date

- This project is due on **Wednesday, June 3, 2015** at **11:59 PM**.
- **You may not use more than one late day for this project.** Hence, the hard deadline for this project is **Thursday, June 4, 2015** at **11:59 PM**.

Submission instructions

- Make sure your solutions/part[**1,2,3**] directories contain the deliverables mentioned below. Each directory should contain the deliverables as mentioned in each part; misplaced deliverables may not receive credit.
- Run the following command from the project root directory to generate the submission tarball submission.tar.gz:
  make submission
- Upload the submission tarball to a Stanford cluster machine (i.e. myth or corn).
- To submit, run the following command from the directory containing the tarball:
  /usr/class/cs155/bin/submit proj3
- Follow the on-screen instructions.
- You can check whether we have received your submission by going to /usr/class/cs155/submissions/proj3.

Introduction

This project is about Android security and network attack detection. It consists of three parts. The first part is an easy warm up, the second digs into details of Android permissions, and the third deals with detection of network attacks.

Parts 1 and 2 are performed on the Android emulator. The provided phone image is locked, but you can get root access to it through `adb` (Android Debug Bridge). In part 1, you will learn about dangers of keeping sensitive data unencrypted. In part 2, you will learn the basics of Android app revers engineering to craft an attack against a vulnerable application. In part 3, you will become familiar with packet dump formats, as well as the basics of IDS (intrusion detection systems).

At the end of this document there is a section called "Getting started with Android development" which you should read before beginning. Before beginning Part 1, you should be able to launch the provided Android Virtual Device image in the Android emulator and access the shell using `adb`.

Part 1

Warmup : A forensics treasure hunt

We have provided you an Android Virtual Device image (called `Bradley.avd`). Boot it up using instructions at the end of the document. On the default emulator, this will take about 3 minutes. Since the device is locked, you should look at the documentation for `adb`. (To get started, run `adb shell`.)

Retrieve at least the following information from the device:

1. The list of contacts.
2. Recent SMS messages.
3. IM credentials from the Xabber app.
4. E-mail credentials.
5. Bonus browser bookmarks, call logs, or anything else that is interesting.

Hint: Look around for .db files. Also, the user doesn't use the native email app, so look for alternative e-mail clients.

Part 1 Deliverables

1. **contacts.tsv** a tab-separated text file with one contact per line, listing each contact's name, email address and/or phone number (simply omit columns that do not exist).
2. **sms.txt** a file containing the body of each SMS on a separate line.
3. **im.txt** a file containing the user's IM *username@host* on one line and their password on a second.
4. **email.txt** a file containing the user's email address *username@host* on one line and their password on a second.
5. **extra.txt** anything else you find, in whatever format you choose.

Part 2

Android app security: Reverse-engineering and privilege escalation

The virtual device's user has installed an application called TrustedApp, which is supposed to send fun SMS messages to the user's friends. TrustedApp was installed with the READ_CONTACTS and SEND_SMS permissions.

However, the developers of TrustedApp did not take CS 155, so they accidentally introduced a vulnerability that allows a malicious third-party app to read the list of contact names on the device. The vulnerability is related to the inter-component communication features of Android, which determine when one application component can send a request to another. Your task is to discover the details and exploit this vulnerability.

You can assume that the user will install a malicious app written by you, called EvilApp. EvilApp has the INTERNET permission, but no other permissions. Design EvilApp to trick TrustedApp into extracting the contact list.

To proceed, you will need to use the Android SDK tools to reverse-engineer TrustedApp. Make sure you read through the "Getting started with Android development" section of this document and set up the SDK tools and emulator first.

1. **Setup: Find and disassemble the TrustedApp.apk.** Android applications are distributed in Android .apk files. These files are very similar to Java JAR files: zipped archives containing everything needed to run the app. The steps suggested for reverse-engineering TrustedApp are as follows:
   - Install the Android SDK tools, and start the provided Android image in the emulator.

- Find and copy the TrustedApp apk off the device using `adb`. (Hint: `adb push/pull`)
- Now that you have the .apk file, unzip it. The AndroidManifest.xml file, which describes the components of the application, will appear in an encoded format. A Google search will reveal various ways to decode it. You can also obtain this information from the manifest file by using `aapt` on the original .apk file using the command below.

```
aapt l -a [filename.apk]
```

- The .apk also holds a file called "classes.dex." This file contains the compiled Android application code. You can use the "dex2jar" tool to convert to a JAR file, and then extract the JAR file to find the .class files. This tool can be found here: `http://code.google.com/p/dex2jar/`.
- Once you have the class files, a Java decompiler can be used to view the (approximate) original Java code. A good tool to use is JD-GUI, found here: `http://jd.benow.ca/`.

Now use the extracted code and your understanding of how Android applications and services work to craft your attack!

2. **Programming task 1:** You will find that TrustedApp exposes a certain component that can be accessed by EvilApp. The developers of TrustedApp tried to include *some* security: you'll find that a secret key is required. **Find the key in the decompiled code and write the attack code in the provided EvilApp Eclipse starter project.** You will need to infer the interface in order to successfully send a request to TrustedApp. Make sure you understand how an AIDL (Android Interface Definition Language) file works: `http://developer.android.com/guide/components/aidl.html`. In order to test your code, you should follow these steps:
   (a) Create a fresh Android emulator image in the AVD Manager and start it. (Use the latest "Target", i.e. Android 4.4.2 - API Level 19.)
   (b) Import the EvilApp starter code into Eclipse.
   (c) Install the TrustedApp .apk file on the new device using "adb install." (It's easiest to close the emulator from Part 1 before this, so adb sees there is a single running emulator only.) Verify that you can open TrustedApp on the phone and use it to view the contacts.
   (d) **Important:** On the emulator, create some contacts using the phone's normal interface. These are the contacts you are going to extract.
   (e) Run your app using the Eclipse "Run" or "Debug" commands, and it should automatically push your compiled EvilApp onto the device and open it up.
3. **Programming task 2:** It turns out the secret key varies from version to version of TrustedApp, and you want to make an attack that works on all versions. **Find a way to retrieve the contacts from TrustedApp using EvilApp *without* your knowledge of the secret key.**
   Hint: Look for more mistakes by the TrustedApp developers in the source code.
4. **Q&A time:** The moral of the story for an app developer is that you should never roll your own security - take the time to use the established methods of the system you're building on.
   (a) What should the developers of TrustedApp have done to make their app secure against the attack performed by EvilApp?
   (b) What are the real-world defenses against reverse-engineering an Android app?

Part 2 Deliverables

1. A copy of one of the .java files from the decompiled TrustedApp.
2. **MainActivity.java**, containing your attack code using the secret key.
3. **MainActivity_no_key.java**, a separate file you create containing the same attack without using the secret key you found.
4. **Answers.txt**, a text file containing your answers to the two questions in item 4.

Part 3

Packet sniffing

You have been given a dump of packet header information for the network that your mobile device is on (`trace.txt`). Your device's IP address is 10.30.22.101. Lately, there have been a number of sketchy occurrences on this network. In order to learn more about network security, your job is to sift through the packet header information and detect some anomalous behavior on the network. Since the volume of packets is large, it is expected that you will write scripts in the language(s) of your choice to do the following:

1. Your mobile device has accessed a number of websites. **List 5 IP addresses of websites that it have been accessed by your mobile device.** (Hint: What port is typically used for webservers? Your mobile device's IP address is 10.30.22.101. There may be more than 5 valid answers to this question.)

2. Someone sketchy has been looking for attack vectors into hosts on this network by scanning for open ports on a machine. (Hint: What kind of pattern would you expect from a host that is port scanning other hosts? How might you isolate this pattern in the packet dump?)

   (a) What is the IP address of the origin of the port scan?

   (b) What is the IP address of the host that was scanned?

   (c) What range of ports was scanned?

3. There has been an unusually high volume of syn packets going from one host on this network to another host on this network. This can be indicative of a type of Denial of Service (Dos) attack called a syn flood.

   (a) What is the IP address of the syn flooding sender?

   (b) What is the IP address of the syn flooded receiver?

   (c) How many syn packets were sent from this sender to this receiver?

4. Some malware on your phone has caused it to behave improperly and inject a malicious packet in the network. More specifically, your phone has sent a packet during a TCP connection that it should not have sent. Doing so can sometimes cause problems on the network and crash hosts that are not implemented correctly. **What is the checksum value of the injected packet?** (Hint: Under what conditions will a client send a packet with a sequence number that has already been sent and acknowledged? Should these packets always have identical content? What does the checksum of a packet indicate?)

Part 3 Deliverables

1. **PacketSniffingAnswers.txt** A text file containing your answers to questions 1-4 above.

2. Any scripts you wrote to answer the above questions.

Getting started with Android development

1. Get the Android SDK tools for your platform at `http://developer.android.com/sdk/index.html`. Follow the instructions for your platform at `http://developer.android.com/sdk/installing/index.html`.

   (a) It might be useful to add adb and aapt to your system's path. To be safe, add the following folders:

      i. sdk/tools/

      ii. sdk/platform-tools/

      iii. sdk/build-tools/android-4.4.2/

2. Look over the first page of the Android docs: `http://developer.android.com/guide/components/fundamentals.html`. The opening section contains essential information on how Android works. You can skip the last two sections, "Declaring app requirements" and "App Resources."

3. Start Android Studio/Eclipse

4. Import the desired Android project (e.g. EvilApp) into your workspace via File >New >Import...
5. Start the Android image in the emulator.
   (a) In the toolbar, click “Android Device Monitor”
   (b) In the new window’s toolbar click “Android Virtual Device Manager”
   (c) The top of the AVD Manager window will contain a line “List of existing Android Virtual Devices located at [PATH]” Copy **Bradley.avd** and **Bradley.ini** into PATH.
   (d) Edit **Bradley.ini** such that the “path=” field contains the path to **Bradley.avd**, i.e. [folder name]/Bradley.avd.
   (e) Hit “Refresh” in the virtual device manager. You should see the device image appear.
   (f) Based on the contents of **Bradley.ini**, figure out if you need an older SDK platform, and install it via the “Android SDK Manager”. You can open the SDK Manager using the main toolbar (You should restart Eclipse after this step.)
   (g) “Repair” the image in the device manager, if necessary, and click “Start” to run the emulator.