DREU 2010: Attacking Android

Kathryn Lingel

Summer 2010

Abstract

Applications that are run on the Google Android operating system require users to grant specific security "permissions" in order to perform certain restricted functions. This summer, I worked together with Royce Cheng-Yue (UC Berkeley '12) to create a semi-automatic system to test the actual functionality of an Android phone against the outlined specifications to discover if the operating system is implemented as securely as it ought to be. We did this by creating a map of exactly what it is possible to do without any permissions and with different combinations of permissions. We chose four Android classes to test and wrote an Android application to run every documented function in those classes without any permission. Then, we developed a script and a number of computer programs to automatically discover all of the permissions a class might want, re-install and run the application with every possible combination of those permissions, and save all of the data to logfiles in XML format for easy parsing. The system is extensible, so more unit tests can be added with relative ease.

1 Introduction

As smart mobile phones become more popular and more capable, it becomes more and more important that they stay secure. Modern smartphone users do everything from work to banking on their phones, using applications developed by third parties and distributed by trusted sources. With thousands of applications being developed and used, untrustworthy applications have a good chance of slipping in unnoticed. If a user downloads a game, he or she doesn't expect it to be able to make phone calls; if he or she downloads a notepad, it probably shouldn't be able to access personal information and send it in a text message. Without any protection, a malicious application could get a user into all sorts of trouble, so it's important to make sure that a user knows what his or her applications are capable of. Android is one of the most popular smartphone OSes on the market [1], but it's also less than two years old and relatively untested. This project aims to explore the security of the Android operating system by investigating what applications are and are not allowed to do.

2 Background

Some knowledge of the Android operating system is necessary in order to understand the significance of our project. In particular, knowing the basic structure of applications is essential, as is understanding what permissions are and how they are intended to make applications secure.

2.1 Android Applications

Android applications (apps) are developed in Java, then compiled into `.apk` files, which are installed on an Android phone. Apps usually consist of one main class (which takes over when the app is launched) and any number of additional classes. Every app must have a manifest file in its root directory called `AndroidManifest.xml`. This manifest file identifies the app to the Android operating system, contains a list of all the classes associated with the app, and is where the application stores a list of all of the permissions that it uses.

2.2 Permissions

The system the Android operating system implements to protect its users involves objects called "Permissions", which grant applications the ability to do anything from sending a text message (requires `android.permission.SEND_SMS`) to switching off all functionality of the phone (`android.permission.BRICK`). If an application tries to use any functions that are locked by the OS without requesting the correct permission, then Android will not allow the function to execute; instead, it will throw a `SecurityException`. In order for an application to be able to use the function, then it has to declare its intent to use the relevant permission in its manifest file. Upon downloading and installing the application, the user is presented with a list of the permissions the application requires. The user then must choose either to grant the application all of the requested permissions and continue installation, or to cancel the installation. In theory, then, by explicitly granting the app permissions, the user ought to know exactly what the app can and cannot do.

2.3 Prior Work

In Spring '10, a team of three graduate students at UC Berkeley - Steve Hanna, Erika Chin, and Adrienne Felt - explored the possibility of a confused deputy attack in Android applications. They built a software tool and used it to investigate code paths between existing Android functions, with the goal of determining whether or not it was possible for an application without a certain permission to trick an application with that permission to performing a task for it. They found a large number of potentially harmful code paths, which inspired them to continue investigating in the area of Android security, specifically including the permissions system. [2]

3 Details

At the core of our system is an Android application, developed for Android release 2.1 or above and tested on a Nexus One running Android 2.2. This application consists of a "main" class and a number of "test" classes. Each "test" class corresponds to a class in the Android operating system, such as `android.bluetooth.BluetoothAdapter` or `android.net.wifi.WifiManager`. From the main class, one can choose to launch any of the test classes. When a test class is run, the application runs a number of test functions in series, writing the results to a logfile.

However, just running the tests once, without any permission, isn't thorough enough to map out which functions need exactly which permissions; maybe a function requires two permissions to run, but only throws one exception at a time. Additionally, it doesn't tell us enough about potential bugs in the permission system; for example, a programmer could hypothetically have had an off day and accidentally switched the code for `android.permission.FLASHLIGHT` with `android.permission.CAMERA`, which could allow someone to create a misleading application.

In order to fully explore the permission space of Android classes, we plugged the phone into a laptop running Ubuntu 10.04 and developed an automatic system to run the app multiple times. We developed a number of helper programs and tied our programs together with android commands with a bash script. When run, this script determines all of the permissions the class needs in order for all of the unit tests to pass, then runs the tests again multiple times, each time with a different combination of the relevant permissions. The results are saved to logfiles that are both human-readable and computer-parsable, for later analysis.

3.1 Unit Tests

Each of the "test" classes consists of a battery of unit tests corresponding to a specific Android class. Each unit test is a Java method which calls one function of the Android class being tested, encased in a `try/catch` block. Every function listed in the online Android documentation [3] gets its own test. If the function needs permission to run, then Android throws a `SecurityException`, which our program catches and notes in the logfile; if not, then our program also makes a note in the logfile. The `main` function of each class runs each unit test once in succession.

3.2 Detecting Permissions

Helpfully, whenever the Android operating system throws a `SecurityException` because of insufficient permission, the name of the required permission is contained inside the exception object's error message field. Every time our application caught a `SecurityException`, the error message in the exception was written to the logfile so that the name of the permission could later be parsed out.

3.3 Automation

When invoked with the name of a class to test, the bash script automatically builds, installs, and runs the application on the phone, then pulls the logfile from the phone, parses out the permissions needed by the phone, adds those permissions to the manifest file, and repeats until a full list of all the permissions needed to run the application is gathered. With that information, a series of programs on the computer create a list of all combinations of the necessary permissions, then generate new manifest files based on those combinations. For example, if an application required the three permissions `PERM_A`, `PERM_B`, and `PERM_C`, then eight unique manifest files would be generated: One requesting no permissions, one with all three, three files with one permission each (`PERM_A`, `PERM_B`, `PERM_C`), and three files with two permissions each (`PERM_A` and `PERM_B`, `PERM_B` and `PERM_C`, `PERM_A` and `PERM_C`). The script then compiles, loads, and runs the application with each manifest file in turn. Each run generates an XML logfile containing the name of each function tested and the name of the additional permission it required (if any), which are named by class, date, and time and saved to a folder on the computer.

4 Conclusions

Unfortunately, my time at UC Berkeley ended before we were able to analyze the results of logfiles from the four classes. While the system was in development, we did notice that documentation on the Android website is incomplete. Several functions, notably some in the `Telephony` class, require permission to run, even though the documentation makes no note of it.

In the near future, our system will be used to generate a map of which permissions are required by which Android functions, and whether or not there are any inconsistencies. Additionally, our results will be used as a basis for comparison against the results of a software tool that Erika and Adrienne are building, which will explore Android functions at a lower level, searching for the same permission vulnerabilities.

5 Acknowledgements

My partner for this project was Royce Cheng-Yue, an undergraduate student at UC Berkeley, who deserves great thanks for being wonderful to work with. Our supervising grad student, Steve Hanna, was always helpful, and was also great at encouraging us to come up with our own solutions, which was fantastic. I'm also grateful to the collective group of security grad students for being friendly and welcoming, and for allowing Royce and me to sit in on interesting research-related things like practice presentations and reading groups. Finally, thank you to my DREU mentor, Dawn Song, for being supportive and providing me with this opportunity.

References

[1] Nielsen Company, The, "Android Soars, but iPhone Still Most Desired", `http://blog.nielsen.com/nielsenwire/online_mobile/android-soars-but-iphone-still-most-desired-as-smartphones-grab-25-of-u-s-mobile-market/`, August 2, 2010.

[2] Hanna, Steve, Erika Chin and Adrienne Felt, *Disoriented 'Droids*, May 2010.

[3] Google Android documentation, "Security and Permissions", `http://developer.android.com/guide/topics/security/security.html`