CS-558 Internet Systems & Technologies.
Presentation Report
(Android Permissions Demystified)
Due on May 11, 2015
Instructor: Evangelos Markatos

Michalis Diamantaris
Androis unrestricted application market and open source have made it a popular platform for third-party applications. Android supports third party development with an extensive API that provides applications with access to phone hardware (e.g., the camera), WiFi and cellular networks, user data, and phone settings. Access to privacy- and security-relevant parts of Androids rich API is controlled by an install-time application permission system. Each application must declare upfront what permissions it requires, and the user is notified during installation about what permissions it will receive. If a user does not want to grant a permission to an application, he or she can cancel the installation process. This paper studies Android applications to determine whether Android developers follow least privilege with their permission requests.

Android Permissions
In Android 2.2 there are 3 kinds of permissions

(a) **Normal permissions**: protect access to API calls that could annoy but not harm the user.

(b) **Dangerous permissions**: control access to API calls with potential harm to user.

(c) **Signature/System permissions**: regulate access to most dangerous privileges.

Permissions are required for invoking the Android API, for getting/setting user data in databases(e.g. content providers) and for sending/receiving intra and inter applications messages(e.g. intents).

Android Architecture
Android has a privileged separated architecture. Each application process is sandboxed and all the critical system libraries are in a separated JVM. In order for an application to access resources of another process, or get access to hardware components, a Remote Procedure Call has to be made, through the IPC/Binder. Then, if the application has the appropriate permissions, the access is granted.

The goal of the authors, is to discover for each API method, the permissions needed for a successful invocation. They constructed a permission map that identifies the permissions required for each method in the Android API. In order to do so, the modified Android 2.2’s permission verification mechanism to log permission checks as they occur. They generated API calls in three phases. First they used Randooop, an automatic unit test generator for Java. Randooop is a feedback directed testing tool, and can take the return value of a successful call and use it as an input in another. With Randooop they achieved a 60% coverage of the total API methods. Next, they created a custom tool, were generated tests are corrected by human supervisor. By doing so they added a 25% more coverage. Finally, the manually verified the correctness of the permission map. With this approach, they achieved 85% coverage and identified 1665 classes with 16,732 public and private methods in the Android API. They believe that the rest 15% probably does not need permission checks. The permission map characterizes different permissions based on, Signature/System, Unused, Hierarchical and Granularity permissions. In order to analyze an Android application and determine the maximum set of permissions it may require, they created Stowaway, a static analysis tool. Stowaway analyzes the applications use of API calls, Content Providers, and Intents and then uses the permission map to determine what permissions those operations require. They applied Stowaway to 940 Android app and identified 323 of 900 (35.8%) as having unnecessary privileges (over-privileged apps). They believe that overprivileged apps are due to common developer errors such as, confusing permission names, deputies, deprecated permissions, copy-paste code and testing artifacts. Finally, they believe that developers are trying to follow least privilege but sometimes fail due to insufficient API documentation.