Android Permissions
C. Capps
November 28, 2012
Contents

1 Overview of Permission System

2 User Understanding of Permissions

3 Developer overprivilege of apps
   - Android Permissions Demystified

4 How to detect malicious apps using privileges
   - Mining Permission Request Patterns
   - Various Approaches in Analyzing Android Applications
   - Android Permissions: A Perspective Combining Risks and Benefits
What are Android Permissions?

- Apps statically request permissions in the AndroidManifest.xml file
- No support for dynamically granting apps permissions at run-time.
- The user sees a dialog at install time, and can choose to cancel installing the app based on the requested permissions
- Relies on the user’s understanding of the various permissions
Permission Categories

Permissions are organized into 3 categories:

- **Normal** - API calls that could annoy but not harm the user, e.g. `SET_WALLPAPER`
- **Dangerous** - API calls that could be used to charge the user money or leak private information such as `READ_CONTACTS`
- **Signature / System** - ability to delete application packages, control backup. Only allowed by apps signed by the manufacturer.
According to Felt, et al. [4] the most commonly checked permissions by the Android API are the following (number of methods that check these permissions):

<table>
<thead>
<tr>
<th>Permission</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLUETOOTH</td>
<td>85</td>
</tr>
<tr>
<td>BLUETOOTH_ADMIN</td>
<td>45</td>
</tr>
<tr>
<td>READ_CONTACTS</td>
<td>38</td>
</tr>
<tr>
<td>ACCESS_NETWORK_STATE</td>
<td>24</td>
</tr>
<tr>
<td>WAKE_LOCK</td>
<td>24</td>
</tr>
<tr>
<td>ACCESS_FINE_LOCATION</td>
<td>22</td>
</tr>
<tr>
<td>WRITE_SETTINGS</td>
<td>21</td>
</tr>
<tr>
<td>MODIFY_AUDIO_SETTINGS</td>
<td>21</td>
</tr>
<tr>
<td>ACCESS_COARSE_LOCATION</td>
<td>18</td>
</tr>
<tr>
<td>CHANGE_WIFI_STATE</td>
<td>16</td>
</tr>
</tbody>
</table>
Permission System

An app makes calls to the public API (and possibly hidden classes by using reflection.) This then communicates with a system process running in a Dalvik Virtual Machine. Apps can include native C code, but the native code can’t directly make API calls (need a Java wrapper.)

Diagram from Felt et al. [4].
Permission System

- Since permissions are checked in the system process, behavior is undefined if an app attempts to use an unauthorized permission
- Might throw a SecurityException
- Might crash the app
- Prevent a broadcast from being sent or received
- Users can create custom permissions
Contents

1. Overview of Permission System

2. User Understanding of Permissions

3. Developer overprivilege of apps
   - Android Permissions Demystified

4. How to detect malicious apps using privileges
   - Mining Permission Request Patterns
   - Various Approaches in Analyzing Android Applications
   - Android Permissions: A Perspective Combining Risks and Benefits
Study by Felt, et al.

- Not too surprisingly, users generally click past permissions warning without understanding them
- Study done by Felt et al. [5] from U.C. Berkeley, *Android Permissions: User Attention, Comprehension, and Behavior*
- Surveyed 308 Android users, and asked questions of 25 in a lab environment.
- 17% paid attentions to permissions at install-time
- 42% were completely unaware of permissions
Effective warnings

- In a paper by Baskar Sarma, et al. some guidelines for a good warning system are proposed
  1. Simple semantic meaning for users and developers
  2. Triggered by a small percentage of apps
  3. Triggered by many malicious apps

- Current system triggers too many warnings (93% of free apps have “dangerous” permissions)
Contents

1. Overview of Permission System
2. User Understanding of Permissions
3. Developer overprivilege of apps
   - Android Permissions Demystified
4. How to detect malicious apps using privileges
   - Mining Permission Request Patterns
   - Various Approaches in Analyzing Android Applications
   - Android Permissions: A Perspective Combining Risks and Benefits
Consequences of Overprivilege

- Overprivilege conditions users to accept unnecessary privileges
- Violates principle of least privilege
- Make applications more vulnerable
- More difficult to detect malicious apps with unusual permission patterns
Android Permissions Demystified

- Android Permissions Demystified[4]
- Experimentally determine which API calls require what permissions
  - Include private classes that developers can call using reflection
- Statically analyze Android APK files to detect overprivileged apps
First, they used Randoop to try calling all possible methods from a list of classes.

- Modified Android kernel to log all permission checks.
- Pool of input sequences, initially just primitive values.
- Difficulty: generate correct input so that an exception is not thrown.
  - Exception may prevent permission checks from being performed.
- Difficult to get instance of every input type, seed pool of inputs with common values obtained from API.
  - e.g. `android.content.Context.getSystemService("wifi")`
Results

- 85% coverage of API methods
- 1,259 API calls check permissions
- The API documentation only lists 78 (more at the top of classes, but very unclear)
- 6 methods are documented incorrectly
Stowaway

- Statically analyze an app, determine set of required permissions
- Examine methods that are invoked, directly or through reflection
- Many challenges, e.g. using a WebView requires the INTERNET permissions
- android-permissions.org
Results

- 35.8% of applications are overprivileged
- 56% of overprivileged applications use 1 extra permissions
- 94% use 4 or fewer extra permissions
- Most common unnecessary privileges:

<table>
<thead>
<tr>
<th>Permission</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS_NETWORK_STATE</td>
<td>16%</td>
</tr>
<tr>
<td>READ_PHONE_STATE</td>
<td>13%</td>
</tr>
<tr>
<td>ACCESS_WIFI_STATE</td>
<td>8%</td>
</tr>
<tr>
<td>WRITE_EXTERNAL_STORAGE</td>
<td>7%</td>
</tr>
<tr>
<td>CALL_PHONE</td>
<td>6%</td>
</tr>
<tr>
<td>ACCESS_COARSE_LOCATION</td>
<td>6%</td>
</tr>
<tr>
<td>CAMERA</td>
<td>6%</td>
</tr>
<tr>
<td>WRITE_SETTINGS</td>
<td>5%</td>
</tr>
<tr>
<td>ACCESS_MOCK_LOCATION</td>
<td>5%</td>
</tr>
<tr>
<td>GET_TASKS</td>
<td>5%</td>
</tr>
</tbody>
</table>
Contents

1. Overview of Permission System
2. User Understanding of Permissions
3. Developer overprivilege of apps
   - Android Permissions Demystified
4. How to detect malicious apps using privileges
   - Mining Permission Request Patterns
   - Various Approaches in Analyzing Android Applications
   - Android Permissions: A Perspective Combining Risks and Benefits
Permission Patterns

- *Mining Permission Request Patterns from Android and Facebook Applications*
- Paper by Mario Frank, et al. at U.C. Berkeley, rigorous statistical analysis of permission patterns [1]
- Android and Facebook permission patterns
- Determine riskiness of an app based solely on permissions used
Permission Patterns

- Statistically find permission patterns used by high reputation apps
- Whitelist apps with ordinary patterns, warn users about unusual patterns
- Used 188,389 apps for analysis
- Web-crawled the web version of the Android market, parsed HTML to get permissions used, number of ratings, average rating, cost, etc.
### Most commonly requested permissions

15 most requested Android permissions (Mario Frank, et al.) [1]

<table>
<thead>
<tr>
<th>requested</th>
<th>permission name</th>
</tr>
</thead>
<tbody>
<tr>
<td>69.76%</td>
<td>Network communication: full Internet access</td>
</tr>
<tr>
<td>43.24%</td>
<td>Network communication: view network state</td>
</tr>
<tr>
<td>30.26%</td>
<td>Storage: modify/delete USB storage &amp; SD card contents</td>
</tr>
<tr>
<td>26.47%</td>
<td>Phone calls: read phone state and identity</td>
</tr>
<tr>
<td>18.34%</td>
<td>Your location: fine (GPS) location</td>
</tr>
<tr>
<td>16.89%</td>
<td>Your location: coarse (network-based) location</td>
</tr>
<tr>
<td>16.16%</td>
<td>Hardware controls: control vibrator</td>
</tr>
<tr>
<td>15.01%</td>
<td>System tools: prevent device from sleeping</td>
</tr>
<tr>
<td>8.22%</td>
<td>Network communication: view Wi-Fi state</td>
</tr>
<tr>
<td>8.11%</td>
<td>System tools: automatically start at boot</td>
</tr>
<tr>
<td>6.71%</td>
<td>Services that cost money: directly call phone numbers</td>
</tr>
<tr>
<td>6.27%</td>
<td>Your personal information: read contact data</td>
</tr>
<tr>
<td>5.59%</td>
<td>Hardware controls: take pictures and videos</td>
</tr>
<tr>
<td>4.61%</td>
<td>System tools: set wallpaper</td>
</tr>
<tr>
<td>3.9%</td>
<td>System tools: retrieve running applications</td>
</tr>
</tbody>
</table>
Mining Permission Request Patterns

Boolean Matrix Factorization

- Goal: find statistically significant permission request patterns
- Input: binary matrix $\mathbf{x}$ where $x_{id} = 1$ means app $i$ requests permission $d$.
- Output: number of statistically significant patterns, $K$
- Matrix $\mathbf{z}$ — the permission patterns in each app
- Matrix $\mathbf{u}$ — the statistically significant permission request patterns
Boolean Matrix Factorization

- Define boolean product $c = a \otimes b$ of 2 matrices by:
  \[ c_{id} = \bigvee_{k=1}^{K} (a_{ik} \land b_{kd}) \]
- Want to find $z, u$ such that $x \approx z \otimes u$
- If app $i$ has pattern $k$ and pattern $k$ has permission $d$, then app $i$ has permission $d$
They trained this model on high reputation apps (average rating of 4 or higher, at least 100 user ratings)

$K = 30$ significant permission patterns

Note that Permission Request Patterns are not disjoint: apps can request multiple patterns (subsets of its permissions.)

A PRP with 1 permission indicates that a permission is requested a lot, but not always together with the same permissions.
More results

Most common permission request patterns:

- Storage: modify/delete USB storage and SD card contents
- Network communication: full Internet access
- Network communication: view network state
- System tools: prevent device from sleeping
- Phone calls: read phone state and identity
- Hardware controls: control vibrator
- System tools: automatically start at boot
- Network communication: view Wi-Fi state
- Your location: fine (GPS) location
- Your location: coarse (network-based) location
- System tools: retrieve running applications
- Your personal information: read contact data
- Your messages: read SMS or MMS
- Your messages: receive SMS
- Hardware controls: take pictures and videos
- Hardware controls: record audio
- System tools: modify global system settings

![Permission Request Pattern Graph](image)
Mining Permission Request Patterns

More results

- If an app has a permission request pattern that is not among these whitelisted patterns, then it is risky
- Can be used to predict likely reputation of new apps
- Good for detecting risky or buggy apps, but not a malware detector
- Did not analyze categories of apps in Google Play store
Various Approaches in Analyzing Android Applications

Network Visualization

- Various Approaches in Analyzing Android Applications with its Permission-Based Security Models
- Paper by Ittipon Rassameeroj and Yuzuru Tanahashi (U.C. Davis) [2]
- Visualizing related permissions per-category
- Create a network visualization based on permission data
Various Approaches in Analyzing Android Applications

Network Visualization

- Dataset 1: Adjacency matrix of permission concurrence
  - $M_{ij} =$ no. of apps where permission $i$ and permission $j$ are both requested
- Dataset 2: Adjacency matrix of distance between apps
  - Represent permissions of an app as a bit-vector
  - Distance between 2 apps is the Euclidean distance
  - Adjacency matrix of the resulting graph
Various Approaches in Analyzing Android Applications

Concurrent Permissions over All Apps

- Roughly divides permissions into large functional categories

![Permission network of all APKs. Operationals that are granted in the permissions of each cluster represents a unique aspect of the device. The purple cluster contains many operations that a phone would perform. The orange cluster contains many operations that a web client and GPS would perform. The pink cluster contains many operations that ubiquitous devices such as a smartphone would perform.](image)
APK Network

- Network of similar apps in Travel category

Fig. 2. APK network in the Travel category.
Results

- Suggests a method for manually finding suspicious apps
- For example, a tipping program that appears in the cluster for apps related to checking exchange rates
- Likely overprivileged or malicious
- Rank clusters by dangerous combinations of permissions
Overview of Permission System User Understanding of Permissions

Developer overprivilege of apps

How to detect malicious apps

Android Permissions: A Perspective Combining Risks and Benefits

Risk Signals

Android Permissions: A Perspective Combining Risks and Benefits
Explore various techniques for giving a warning to the user
Minimize warnings while maximizing detection of malware
By category and sub-category
Risk Signals

- Choose 26 critical permissions, a subset of the “dangerous” permissions
- Category-based rare critical permission signal (CRCP)
- $\text{CRCP}(\theta)$ means an app uses a permission that is used by less than $\theta$ percent of the apps in the same category (theta can be an arbitrary threshold, not just percentage of apps)
- Allow user to select category for app other than its assigned category for purpose of checking if signal is raised
- Tell user what percent of apps in the category trigger signal for any permission
Rare Pairs of Critical Permissions

- A pair of permissions triggers $RPCP(y)$ if:
  - The individual permission’s frequency is greater than $y$, but the frequency of the 2 permissions together is below $y$
  - i.e. the permissions are relatively common, but they are not seen together frequently
  - Trigger warning if $RPCP(y) \geq \theta$
Results

![Graph showing the relationship between Benign % and Malware % for different Android permissions scenarios.](image)

- #RCP(0) >= 1
- #RCP(2) >= 0
- #RCP(2) + #RPCP(1) >= 0
- #RCP(2) + 0.1*#RPCP(1) >= 0
- #RCP(5) >= 0
- #RCP(5) + #RPCP(5) >= 0
- #RCP(5) + 0.1*#RPCP(5) >= 0
- Weighted SVM + RBF Kernel
The SVM performed the best
However, trained only on specific set of apps
Linear combination of RPCP (pair-wise) and RCP (all apps) performed second-best
CRCP (by category) performed better than RCP
Android Permissions: A Perspective Combining Risks and Benefits

Frank, M. and Dong, B. et al. 2012
*Mining Permission Request Patterns from Android and Facebook Applications.* arxiv.org, submitted 8 October 2012.

Rassameeroj, I. and Tanahashi, Y. 2011

Sarma, B. and Li, N. et al. 2012

Felt, A. and Chin, E. et al. 2011
Android Permissions Demystified. CCS ’11: Proceedings of the 18th ACM conference on Computer and communications security


Grace, M. and Zhao, Y. et al. 2012


Pieterse, Heloise and Olivier, Martin. 2012.

Wikipedia Article. Google Play