Permission Evolution in the Android Ecosystem

Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, Michalis Faloutsos
Department of Computer Science and Engineering
University of California, Riverside
{xwei, gomezl, neamtiu, michalis}@cs.ucr.edu

ABSTRACT

Android uses a system of permissions to control how apps access sensitive devices and data stores. Unfortunately, we have little understanding of the evolution of Android permissions since their inception (2008). *Is the permission model allowing the Android platform and apps to become more secure?* In this paper, we present arguably the first long-term study that is centered around both permission evolution and usage, of the entire Android ecosystem (platform, third-party apps, and pre-installed apps). First, we study the Android platform to see how the set of permissions has evolved; we find that this set tends to grow, and the growth is not aimed towards providing finer-grained permissions but rather towards offering access to new hardware features; a particular concern is that the set of `Dangerous` permissions is increasing. Second, we study Android third-party and pre-installed apps to examine whether they follow the *principle of least privilege*. We find that this is not the case, as an increasing percentage of the popular apps we study are overprivileged. In addition, the apps tend to use more permissions over time. Third, we highlight some concerns with pre-installed apps, e.g., apps that vendors distribute with the phone; these apps have access to, and use, a larger set of higher-privileged permissions which pose security and privacy risks. At the risk of oversimplification, we state that the Android ecosystem is not becoming more secure from the user's point of view. Our study derives four recommendations for improving the Android security and suggests the need to revisit the practices and policies of the ecosystem.

1. INTRODUCTION

The popularity of the Android platform is driven by feature-rich devices, as well as the myriad Android apps offered by a large community of developers. Furthermore, smartphones have become an integral part of daily lives, with users increasingly relying on smartphones to collect, store, and handle personal data. This data can be highly privacy-sensitive, hence there are increased concerns about the security of the Android ecosystem and safety of private user data [11].

ACSAC '12 Dec. 3-7, 2012, Orlando, Florida USA

To ensure security and privacy, Android uses a permission-based security model to mediate access to sensitive data, e.g., location, phone call logs, contacts, emails, or photos, and potentially dangerous device functionalities, e.g., Internet, GPS, and camera. The platform requires each app to explicitly request permissions up-front for accessing personal information and phone features. App developers must define the permissions their app will use in the `AndroidManifest.xml` file bundled with the app, and then, users have the chance to see and explicitly grant these permissions as a precondition to installing the app. At runtime, the Android OS allows or denies use of specific resources based on the granted permissions (Section 2). In practice, this security model could use several improvements, e.g., informing users of the security implications of running an app, revoking/granting app permissions without reinstalling the app, or moving towards finer-grained permissions.

In fact, the Android permission model attracts emerging malware that challenges the system to exploit vulnerabilities in order to perform privilege escalation attacks—permission re-delegation attacks [7], confused deputy attacks, and colluding attacks [15]. As a result, users can have sensitive data leaked or subscription fees charged without their consent (e.g., by sending SMS messages to premium numbers via the SMS related Android permissions, as the well-known Android malwares Zsone and Geinimi do [18]). While most of these attacks are first initiated when a user downloads a third-party app to the device, to make matters worse, even stock Android devices with pre-installed apps are prone to exposing personal privacy information due to their higher privilege levels (e.g., the notorious HTCLogger app [5]).

Previous research efforts focus either on single-release permission characterization and effectiveness [6,9,13] or on other permission-related security issues [7,8,15,17]. Unfortunately, there have been no studies on how the Android permission system has evolved over the years, which could uncover important security artifacts beneficial to improving the security of the ecosystem. We discuss previous work in Section 7.

In this paper, we study the evolution of the Android ecosystem to understand whether the permission model is allowing the platform and its apps to become more secure. Following a systematic approach, we use three different types of characterizations (third-party app permissions vs pre-installed app permissions, and two permission classifications from Google). We study multiple Android platform releases over three years, from *Cupcake* (April 2009) to *Ice Cream Sandwich* (December 2011). We use a stable dataset of 237 evolving third-party apps covering 1,703 versions (spanning

a minimum of three years). Finally, we investigate pre-installed apps from 69 firmwares, including 346 pre-installed apps covering 1,714 versions. To the best of our knowledge, this is the first longitudinal study on Android permissions and the first study that sheds light on the co-evolution of the whole Android ecosystem: platform, third-party apps, and pre-installed apps.

Our overall conclusion is that the security and privacy of the ecosystem (platform and apps) do not improve, at least from the user's point of view. For example, the evolution moves more and more toward violating the *principle of least privilege*, a fundamental security tenet. Specifically, our study of the permission evolution of the Android ecosystem leads to the following observations:

1. **The number of permissions defined in Android platform tends to increase, and the `Dangerous`-level set of permissions is the most frequent and continues to grow.** There were 103 Android permissions in the first widely-used release (API level 3); the number of permissions has grown to 165 in the most current release (API level 15). Furthermore, the `Dangerous`-level permissions is always the largest group across all API levels, e.g., 60 out of 165 permissions in API level 15, and is still growing.
2. **Added platform permissions cater to hardware manufacturers and their apps, rather than third-party developers.** Nearly half (49.1% in API level 15) of all permissions are not accessible to third-party developers. Furthermore, of all the *added* permissions between API levels 3 to 15, most (49 out of 62) are in privilege levels that are not available to third-party developers, e.g., `Signature` and `signatureOrSystem` levels.
3. **Android platform permissions are not becoming more fine-grained.** After carefully examining the Android permissions from API level 3 to 15, we observed that most permission changes are *not* geared towards fine-grained instances of previous permissions. In other words, the platform does not seem to be moving towards more fine-grained permissions, which would in general be a step towards increased privacy or security. Instead, the permission changes indicate clearly that the Android platform is striving to give more flexibility and control to smartphone vendors, e.g., HTC, Motorola, Samsung, by providing them with permissions of higher privilege.
4. **Permission additions dominate the evolution of third-party apps, of which `Dangerous` permissions tend to account for most of the changes.** From the analysis of third-party apps, we found that the number of occurrences of adding Android permissions is significantly higher than the number of deleted permissions. Surprisingly, permission changes are not due to changes in the platform. Interestingly, among those additions, newer versions of apps tend to favor adding `Dangerous` permissions most often (66.11% of permission increases in apps consisted of at least one more `Dangerous` permission).
5. **Macroscopic and microscopic patterns emerge when studying evolution of permission usage.** We found evidence that `Dangerous` permission usage sometimes oscillate as an application evolves, which might indicate that developers are unclear about certain permission definitions, and their correct usage.
6. **An increasing number of apps are violating the principle of least privilege.** The tendency of developers to request permissions that their apps do not need causes an app to become overprivileged (as is the case for 44.8% of apps).
7. **The power and privilege of pre-installed apps is growing.** Sixty-six percent of pre-installed apps are overprivileged. Furthermore, pre-installed apps have more power to control and customize Android devices through Android platform-defined and self-defined higher protection level permissions, e.g., `Signature`- and `SignatureOrSystem`-level permissions. Though granting vendors higher privilege is not surprising, end-users (the actual owners of the device) still have security concerns [5, 11]. We argue that since pre-installed apps have greater power over the device, the developers of pre-installed apps must understand and accept their responsibility to protect the end user.

Implications and Suggestions. Our work leads to the following recommendations for increasing the security and privacy of Android users.

1. **Securing the ecosystem must start at the Android platform.** The trends we reveal in the evolution of the Android platform conjure up many security and privacy concerns. The security of the Android ecosystem could improve dramatically by focusing on improving the security of the Android platform by: (a) cautiously increasing the set of `Dangerous`-level permissions, (b) balancing the security of users and convenience of vendors, and (c) offering fine-grained permissions to app developers.
2. **App certification should enforce checks against over-privileged requests.** The existence of over-privileged apps, which are increasing in number, is an indication of, at best, carelessness, and, at worst, greed or malice of the app developer. Checks should be incoporated to discourage permission over-privilege.
3. **App permission evolution and fluctuation indicate developer confusion in selecting legitimate permissions.** Indicated by not only the macro- and micro-evolution patterns of permissions, but also by the tendency of apps to become overprivileged, the the struggling battle of developers to select a set of legitimate permissions for their Android apps is clearly shown in our work.More emphasis should be put on correct permission usage to aid developers in selecting the approriate permissions to use.
4. **Pre-installed manufacturer apps need to be subject to far more scrutiny, as they could be the weakest link.** Pre-installed apps have significant power: (a) they do not require user approval for installation, as they come with the device, (b) they can usually not be removed, even if the user tries to, (c) they get access to higher-privileged permissions, and (d) they are often overprivileged. Pre-installed apps, with all their power, could cause significant damage to the device and user if compromised, thus pre-installed developers must be held to a higher security standard than normal developers.

2. THE ANDROID PLATFORM BASICS

We now proceed to present an overview of the Android platform, Android permission model and a set of definitions for the concepts we use throughout the paper.

2.1 Android Platform

Android was launched as an open-source mobile platform in 2008 and is widely used by smartphone manufacturers, e.g., HTC, Motorola, Samsung. The software stack consists of a custom Linux system, the Dalvik Virtual Machine (VM), and apps running on top of the VM. Each app runs in its own copy of the VM with a different user id, hence apps are protected from each other. A permission model, explained shortly, protects sensitive resources, e.g., the hardware and stored data. In this model, resources are protected by permissions, and only apps holding the permission (which is granted when the app is installed) are given access to the permission-protected resource.

API Levels. To facilitate app construction, the Android platform provides a rich framework to app developers. The framework consists of Android packages and classes, attributes for declaring and accessing resources, a set of Intents, and a set of permissions that applications can request. This framework is accessible to apps via the Android application programming interface (API). The Android platform has undergone many changes since its inception in 2008, and each major release forms a new *API level*. In this paper we studied all major API levels, from level 3 (April 2009) to level 15 (December 2011); levels 1 and 2 did not see wide adoption. With each API upgrade, the older replaced parts are deprecated instead of being removed, so that existing applications can still use them [4].

2.2 Android Apps

In addition to the platform, the Android ecosystem contains two main app categories: third-party and pre-installed.

Third-party apps are available for download from Google Play (previously known as Android Market [2]) and other app stores, such as Amazon [11]. These Android apps are developed by individual third-party developers, which can include software companies or individuals around the world. Malicious apps, designed for nefarious purposes, form a special class of third-party apps.

Pre-installed apps come along with the devices from the vendors. They are developed and loaded in the devices before the devices ever reach the user in the market. These apps can be designed and configured exclusively per device model depending on the needs of particular manufacturers and phone service carriers by the vendor developers.

We studied permission evolution and usage in all components of the ecosystem: platform, third-party apps and pre-installed apps.

2.3 Android Permissions

The set of all Android permissions is defined in the `AndroidManifest.xml` source file of the Android platfrom [10]. To access resources from Android devices, each Android app, third-party and pre-installed alike, requests permissions for resources by listing the permissions in the app's `AndroidManifest.xml` file. When the user wants to install an app, this list of permissions is presented and confirmation is requested; if the user confirms the access, the app will have the requested permissions at all times (until the app is uninstalled). The latest platform release, API Level 15, contains a list of 165 permissions; examples of permissions are `INTERNET` which allows the app to use the Internet, `ACCESS_FINE_LOCATION` which gives an app access to the GPS location, and `NFC` which lets the app use near-field communication. Android defines two categories of Android permissions: *Protection Level* and *Functionality Group*, described next.

API level	Android platform	SDK codename	Total permissions	Release (mm-dd-yy)
15	4.0.3	Ice Cream Sandwich MR1	165	12-16-11
14	4.0.2 4.0.1	Ice Cream Sandwich	162	11-28-11 10-19-11
10	2.3.4 2.3.3	Gingerbread MR1	137	04-28-11 02-09-11
9	2.3.2 2.3.1 2.3	Gingerbread	137	12-06-10
8	2.2.x	Froyo	134	05-20-10
7	2.1.x	Eclair MR1	122	01-12-10
6	2.0.1	Eclair 0 1	122	12-03-09
5	2.0	Eclair	122	10-26-09
4	1.6	Donut	106	09-15-09
3	1.5	Cupcake	103	04-30-09

Table 1: Official releases of the Android platform; base and tablet versions are excluded.

Protection Level. The levels refer to the intended use of a permission, as well as the consequences of using the permission.

1. `Normal` permissions present minimal risk to Android apps and will be granted automatically by the Android platform without user's explicit approval.

2. `Dangerous` permissions provide access to the user's personal sensitive data and various device features. Apps requesting dangerous permissions can only be installed if the user approves the permission request. These are the *only* permissions displayed to the user upon installation.

3. `Signature` permissions signify the highest privilege; they can only be obtained if the requesting app is signed with the device manufacturer's certificate.

4. `signatureOrSystem` permissions are only granted to apps that are in the Android system image or are signed with the same certificate in the system image. Permissions in this category are used for certain special situations where multiple vendors have applications built into a system image and need to share specific features explicitly because they are being built together.

Note that the definition of protection level clearly constrains the privilege for each Android permission: third-party apps can only use `Normal` and `Dangerous` permissions. However, pre-installed apps can use permissions in all four protection levels. When third-party apps request `Signature` or `SignatureOrSystem` permissions, the request is ignored by the platform.

Functionality categories. Android also defines a set of permission categories based on functionality; in total there are 11 categories, with self-explanatory names: `Cost Money`, `Message`, `Personal Info`, `Location`, `Network`, `Accounts`, `Hardware Controls`, `Phone Calls`, `Storage`, `System Tools` and `Development Tools`. There is also a `Default` category that is used when no category is specified in the definition of an Android permission [3].

3. DATASET DESCRIPTION

In this section, we describe the process we used to collect the permission datasets from the Android ecosystem.

3.1 Platform Permissions Dataset

Table 1 presents the evolution of the platform permissions: for each API level (column 1) we show the platform release number (column 2), the textual codename of the release (column 3), the number of permissions defined in that release (column 4), and the release date (last column). Note that we exclude API levels 1 and 2, as the platform only gained wide adoption starting with API level 3. Also, we exclude releases 3.x (named Honeycomb, API levels 11–13); Honeycomb can be regarded as a separate evolutionary branch as it was designed for tablets only, not for smartphones, its source code was not open-source at release, and it was eventually merged into platform version 4.0.

To obtain the permission definitions for each API level, we extracted the file `AndroidManifest.xml` from each release [10]. We then analyzed the changes in permissions between successive releases.

3.2 Apps Permissions Dataset

Third-party apps. We characterize permission usage evolution in third-party apps based on a *stable set* of 237 popular apps with 1,703 versions that span at least three years. We chose these apps because they are widely-used, have releases associated in each API level, and have more than one release per year; hence we could observe how apps evolve and how changes in the platform might lead to changes in apps.

Selecting this stable dataset was far from trivial, and was an involved process. First, we seeded the dataset with 1,100 apps (Top-50 free apps from each category) [16]. Then we crawled historic versions of apps from online repositories, and then retrieved their latest versions from Google Play [1,2]; in total, this initial set contained 1,420 apps with 4,857 versions. Next, we selected only those apps that had at least one version each year between 2009 and 2012. Finally, after eliminating those apps that did not match our requirements, we obtained the stable dataset of 237 apps with 1,703 versions, with each app's evolution spanning at least three years.

Pre-installed apps. Pre-installed apps are much more difficult to obtain because they are not distributed online by vendors—they come with the phone; moreover, the sets of pre-installed apps vary widely among phones and manufacturers. Therefore, to collect pre-installed apps, we used a different process compared to third-party apps. First, we gathered the firmwares of multiple phone vendors—HTC, Motorola, Samsung, and LG—from various online sources. Next, we unpacked the firmwares and extracted the pre-installed apps inside. In total, we collected 69 firmwares over the years which contained 346 pre-installed apps with 1,714 versions.

Permission collection. To obtain the permission list for each app, we use the tool `aapt` on each app version to extract the `AndroidManifest.xml` file, which contains the permissions requested by that version [10]. After obtaining the set of manifest files, we parse the manifest files to get the full list of the permissions used by each app version.

Our analysis is based on these datasets. The datasets contain applications from a large number of developers across a broad range of categories. Thus, we believe that our datasets reflect Android app permission variation and evolution in a meaningful way.

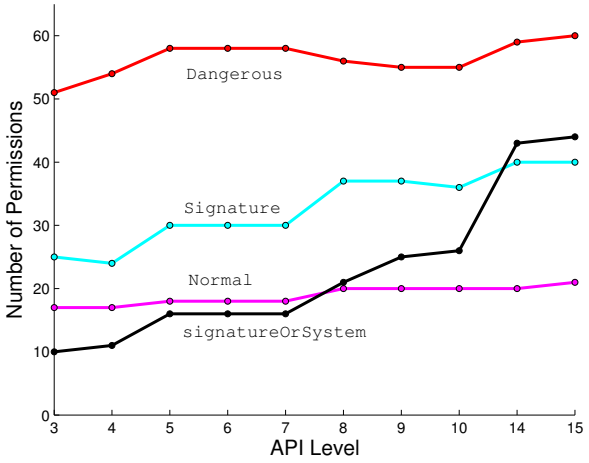


Figure 1: Protection Levels, e.g. Normal, Dangerous, Signature, signatureOrSystem, evolving over API levels.

4. PLATFORM PERMISSION EVOLUTION

We study the evolution of the Android platform permissions through a fine-grained, qualitative and quantitative analysis of permission changes between API levels. As we discussed in Section 2, the Android platform defines the list of all permissions in the framework's source code file `AndroidManifest.xml` for each API level. Since the API level directly reflects what permissions Android platform offers, we use the API level as the defining indicator to compare the Android permission changes.

4.1 The List of Permissions is Growing

As shown in Table 1, the number of Android permissions in each API level is significantly increasing. In early 2009, API level 3 had 103 Android permissions, while there are now 165 Android permissions in API level 15. The net gain of 62 permissions was the result of adding 68 new permissions and removing 6 existing ones. We present the permission evolution by protection level and functionality category.

In Figure 1, we show the permission evolution by protection levels (the levels were described in Section 2). We observe that the number of permissions in each protection level is increasing. In addition, we find that most of the increased permissions across different API levels belong to the protection levels `Signature` and `signatureOrSystem`, which indicates that most of the introduced Android permissions are only accessible to vendors, e.g., HTC, Motorola, Samsung, and LG. This raises significant security concerns for at least two reasons: (1) users have no control over the pre-installed apps, as the apps are already present when the phone is purchased, and (2) a flaw in a pre-installed app will affect all phones whose firmware contained that app. To illustrate the danger associated with pre-installed apps, consider the notorious `HTCLogger` pre-installed app, in which users of certain HTC phones were exposed to a significant security flaw. `HTCLogger` was designed to log device information for the development community in order to debug device-specific issues; as such, the app collects account names, call

API level	Dev tools	Sys tools	Accounts	Cost Money	Hardware Controls	Location	Messages	Network	Personal Info	Phone calls	Storage	Default
3	36	35	1	2	6	4	5	5	6	3		
4	-1	+2,-2				+1					+1	+2
5		+3	+4						+2			+7
6												
7												
8		+7										+6, -1
9					+1			+2	-2			+2
10												
14		+2		+1	+2,-1		+1	+1	+5	+1	+1	+12
15		+1						+1				+1
Overall	-1	+13	+4	+1	+2	+1	+1	+4	+5	+1	+2	+29

Table 2: Permission changes per API level and permission categories.

Dangerous permission	Category
READ_HISTORY_BOOKMARKS	Personal Info
WRITE_HISTORY_BOOKMARKS	Personal Info
READ_USER_DICTIONARY	Personal Info
READ_PROFILE	Personal Info
WRITE_PROFILE	Personal Info
READ_SOCIAL_STREAM	Personal Info
WRITE_SOCIAL_STREAM	Personal Info
WRITE_EXTERNAL_STORAGE	Storage
AUTHENTICATE_ACCOUNTS	Accounts
MANAGE_ACCOUNTS	Accounts
USE_CREDENTIALS	Accounts
NFC	Network
USE_SIP	Network
CHANGE_WIFI_MULTICAST_STATE	System Tools
CHANGE_WIMAX_STATE	System Tools

Table 3: Added `Dangerous` permissions and their categories.

and SMS data, GPS location, etc. Unfortunately, the app stored the collected information without encrypting it and made it available to any application that had the Internet permission [5].

In Table 2, we show the permission evolution by functionality categories: each column contains a category, each row corresponds to an API level, and cell data indicates the number of permissions added and deleted in that API level; note that, the first row shows the number of permissions in each category of API 3. We find that the number of permissions in nearly all the categories is increasing, with the exception of the `Personal Information` category, which yielded a decrease in the number of permissions from API 8 to 9, as shown in Table 2. After grouping the Android permissions into the 11 functionality categories, we find that the `Default`, `System_Tools` and `Development_Tools` categories contribute to most of the increases. Newly-added permissions in these categories allow developers and applications to take advantage of the evolving hardware capabilities and features of the device. We now proceed to providing observations on permission evolution at a finer-grained level.

4.2 Dangerous Group is Largest and Growing

From Figure 1, we can see that the `Dangerous` permission level (the levels were introduced in Section 2.3) vastly outnumbers all other permission types at all times. Note that the `Dangerous` permission set is still growing, even though it is already the largest. We further investigated the growth of permissions in the `Dangerous` protection level.

As shown in Table 3, `Dangerous` permissions are added in 5 out of 11 categories. Most of them are from personal data-related categories, e.g, `PERSONAL_INFO`, `STORAGE` and `ACCOUNTS`. We believe that this evolutionary trend shows that the Android platform provides more channels to harvest personal information from the device, which could increase the privacy breach risk if these permissions may be abused by Android apps.

4.3 Why are Permissions Added or Deleted?

To understand the rationale behind permission addition and deletion, we studied the commit history (log messages and source code diffs) of the Android developer code repository [10].

We found that, in most cases, permissions are added and deleted to offer access to more functionality offered by the device. Advances in the hardware strongly motivate such permission evolution. For instance, in API level 9, new hardware technology for near-field communication led to the introduction of a permission to access NFC. In API level 15, a permission to access WiMAX is introduced in order to access 4G networks.

Permissions can also be deleted to accommodate new smartphone features when they are removed and replaced by new permissions. For example, `READ_OWNER_DATA` was deleted after API level 8, but two new, related permissions, `READ_PROFILE` and `READ_SOCIAL_STREAM` were added in level 14.

Interestingly, some permissions were added in the earlier API levels while deleted later, as the associated functionalities are made available to public without manifest-declared permissions. For example, `BACKUP_DATA` was added in API level 5, but deleted in level 8, because the backup/restore function was made available to all apps by default.

Furthermore, most of the added permissions are permissions categorized as `Default`, `System_Tools` and `Development_Tools`, which are mostly used to access system level information to function and debug the Android apps. However, as we discussed before, most of those permissions are in the `Signature` and `signatureOrSystem` protection levels that are only available to vendor developers in pre-installed apps. This indicates that the added permissions facilitate the development of pre-installed apps by vendor developers, instead of third-party apps by third-party developers. The extended aid to vendors is somewhat adverse, since third-party developers are the dominant and active force in the Android ecosystem.

4.4 No Tendency Toward Finer-grained Permissions

Finer-grained permissions in Android, e.g., separating the advertisement code permissions from host app permissions [14], have been advocated by security groups from both academia

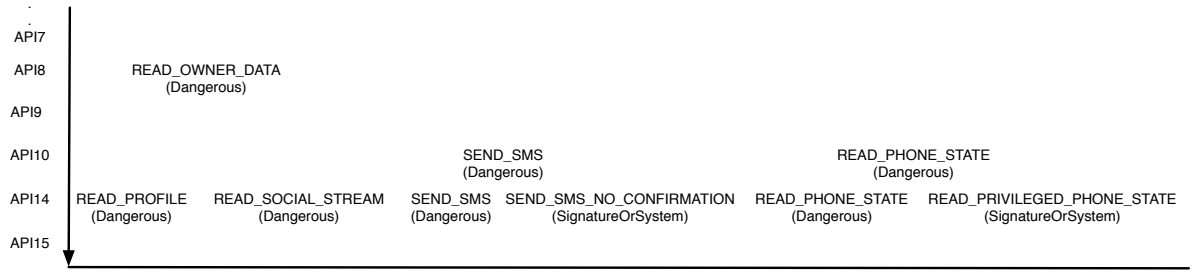


Figure 2: Functionally-similar permissions added and deleted between API levels.

and industry [9, 12, 16]. The basis for finer-grained permissions is the *principle of least privilege*, i.e., giving apps the minimum number of permissions necessary to provide a certain level of service.

We investigated whether Android permissions are becoming more fine-grained over time. After carefully examining the Android permissions from API level 3 to 15, we observe that the permission changes do *not* tend towards becoming more fine-grained. We found only one possible example of a permission splitting in `READ_OWNER_DATA`. However, there is no indication that the two new permissions were specifically designed to replace the previous one, as shown in the first example of Figure 2. Overwhelmingly, the permission changes indicate that the Android platform is giving more flexibility and control to the phone vendors. For example, as shown in Figure 2, `SEND_SMS` and `PHONE_STATE` permissions exist in both API level 10 and 14, but the newly added Android permissions `SEND_SMS_NO_CONFIRMATION` and `READ_PRIVILEGED_PHONE_STATE` gives the app a higher privileged access to the device. Further, those higher privileged permissions are `signatureOrSystem` permissions, which can only used by vendor developers. In summary, we do not observe the evolution of Android permissions that is trending to provide more fine-grained permissions.

5. THIRD-PARTY APPS

We now change our focus and investigate the variation and evolution of permissions from the perspective of the driving force of the Android ecosystem: the apps. We investigate two types of apps, *third-party apps* and *pre-installed apps*; we present and discuss the permission usage of Android apps across different versions and their evolution.

5.1 Permission Additions Dominate

We analyzed the permissions added and deleted in the 1,703 versions of the 237 third-party apps in our stable dataset. In Figure 3(a) we show the distribution of permission changes; on the x-axis we show the number of permission changes: permission additions are marked positive, permission deletions are marked negative. Note that the bulk of the changes are to the right of the origin (0 changes means no permission change), we can conclude that most apps add permissions over time, with some apps adding more than 15 permissions. Only a small number of apps, about 10, delete permissions, and the deletions are limited to at most 3 permissions.

We present the total numbers of permission addition and deletion events in the stable dataset in Table 4: column 2 illustrates that the addition of permissions occurs much more frequently than the deletion of permissions. To disambiguate between genuine permission additions and additions induced by changes in the platform (e.g., as a result of added functionality), we also computed the permission changes induced by changes in the Android platform, which we show in column 3 of Table 4). Surprisingly, these induced changes only account for a small number of the permission changes: less than 3% of either additions or deletions. In sum, we were able to conclude that permission changes, which consist mostly of additions, are not due to changes in the platform.

	Total changes	**Induced by platform changes**
Add	857	14 (1.63%)
Delete	183	5 (2.73%)
Total	*1040*	*19 (1.82%)*

Table 4: App permission changes in the stable dataset.

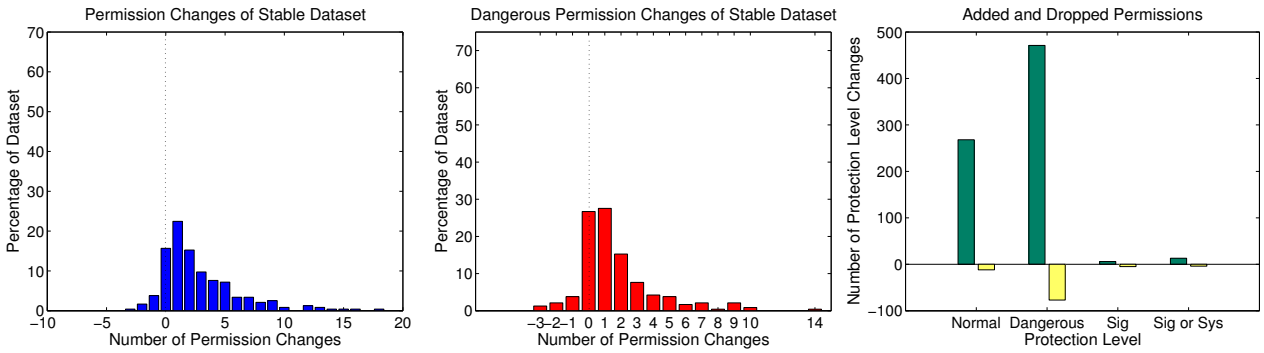
Android permission	**In Top 20?**
ACCESS_NETWORK_STATE	✓
WRITE_EXTERNAL_STORAGE	✓
WAKE_LOCK	✓
GET_ACCOUNTS	×
VIBRATE	✓

Table 5: Most frequently added permissions in the stable dataset.

We now set out to answer the question: *what is the primary cause for the permission additions*? We show the Top-5 most frequently added and dropped permission in the first column of Table 5 and Table 6; column 2 of these tables will be explained shortly. For the added permissions, we found that Android apps became more aggressive in asking for resources, by asking for new permissions. For instance, the Android apps adopt permissions such as `WAKE_LOCK`, `GET_ACCOUNTS`, and `VIBRATE`. `WAKE_LOCK` prevents the processor from sleeping or the screen from dimming, hence allowing the app to run constantly without bothering the user for wake-up actions. `VIBRATE` enables the phone to vibrate for notifying the user when the corresponding apps invokes some functionality. In order to meet the increasing requirement of storage, `WRITE_EXTERNAL_STORAGE` is added to enable writing data into the external storage of the device such as an SD card. We note that permissions that do not improve the user experience, e.g., `ACCESS_MOCK_LOCATION` and `INSTALL_PACKAGES`, the apps simply drop them.

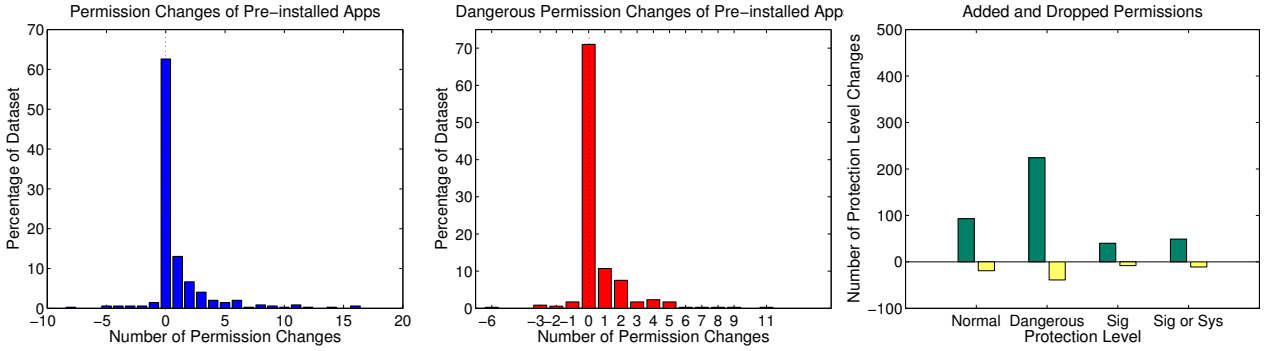
As Android Apps are increasingly adding new permissions, users are naturally have security and privacy concerns, e.g., *how can they be sure that apps do not abuse permissions?*

For comparison, in Table 7, we list the Top-20 permissions that Android malwares request (and abuse), as reported by Zhou and Xiang [18]. We now come back to column 2 in Tables 5 and 6; the columns show the result of comparing the added (and respectively, deleted) permissions in our stable dataset with the Top-20 malware permission list. A '✓' means the corresponding Android permission is in the Top-



(a) (b) (c)

Figure 3: Permission and protection level changes in the third-party apps.



(a) (b) (c)

Figure 4: Permission and protection level changes in the pre-installed apps.

Android Permission	In Top 20?
ACCESS_MOCK_LOCATION	×
READ_OWNER_DATA	×
INSTALL_PACKAGES	×
RECEIVE_MMS	×
MASTER_CLEAR	×

Table 6: Most frequently deleted permissions in the stable dataset.

20 malware permission list, while a '×' means the permission is not in the list. We found that most of the added permissions are on the malware list, while none of the dropped permissions are on the list. Though we certainly can not claim these third-party apps are malicious, the trend should concern users: as apps gain more powerful access, the overall system becomes less secure. For example, in the *confused deputy* attack, a malicious app could compromise and leverage a benign app to achieve its malevolent goals [15].

5.2 Apps Want More Dangerous Permissions

We now proceed to investigate the added permissions in the `Dangerous` protection level as they introduce more risks.

Figure 3(b) shows that 66.11% of permission increases in apps required at least one more `Dangerous` permission. In more detail, we list the frequently used `Dangerous` permissions in the first column of Table 8. We found that `WRITE_EXTERNAL_STORAGE` is the most requested `Dangerous` permission, in which sensitive personal or enterprise files can be written to external media. This permission is also a hotspot for most malicious activities. `INTERNET`, `READ_PHONE_STATE`, and `WAKE_LOCK` are also requested frequently by the new versions of the apps. The first two are needed to allow for embedded advertising libraries (ads), but these third-party ads are also raising privacy concerns of abusing the user's personal information. We then cross-checked this list with the Top-20 malware permissions [18], as shown in column 2 of Table 8. We observed that 9 of the 16 frequent permissions listed are also frequently used by malicious apps. This significant overlap intensifies our privacy and security concerns.

Permission	% of apps using it
INTERNET	97.8%
READ_PHONE_STATE	93.6%
ACCESS_NETWORK_STATE	81.2%
WRITE_EXTERNAL_STORAGE	67.2%
ACCESS_WIFI_STATE	63.8%
READ_SMS	62.7%
RECEIVE_BOOT_COMPLETED	54.6%
WRITE_SMS	52.2%
SEND_SMS	43.9%
VIBRATE	38.3%
ACCESS_COARSE_LOCATION	38.1%
READ_CONTACTS	36.3%
ACCESS_FINE_LOCATION	34.3%
WAKE_LOCK	33.7%
CALL_PHONE	33.7%
CHANGE_WIFI_STATE	31.6%
WRITE_CONTACTS	29.7%
WRITE_APN_SETTINGS	27.7%
RESTART_PACKAGES	26.4%

Table 7: Top-20 most frequent permissions requested by malware (from Zhou and Xiang [18]).

5.3 Macro and Micro Evolution Patterns

The characterization of permission changes we provided so far, in terms of absolute numbers (added/deleted), reveals

Dangerous permission	In Top 20?
WRITE_EXTERNAL_STORAGE	✓
WAKE_LOCK	✓
READ_PHONE_STATE	✓
ACCESS_COARSE_LOCATION	✓
CAMERA	×
INTERNET	✓
ACCESS_FINE_LOCATION	✓
READ_LOGS	×
READ_CONTACTS	✓
RECORD_AUDIO	×
BLUETOOTH	×
CALL_PHONE	✓
CHANGE_WIFI_STATE	✓
GET_TASKS	×
MODIFY_AUDIO_SETTINGS	×
MANAGE_ACCOUNTS	×

Table 8: Frequently used `Dangerous` Android permissions of stable dataset.

Macro pattern	Frequency
0→1	90.46%
1→0	8.59%
1→0→1	0.84%
1→0→1→0	0.11%

Table 9: Macro evolution patterns of permission usage in the stable dataset.

the general trend toward apps requiring more and more permissions. In addition, we also performed an in-depth study where we looked for a finer-grained characterization of permissions evolution in terms of "patterns", e.g., repeated occurrences of permission changes.

Macro patterns. To construct the macro patterns, we use 0→1 and 1→0 as the basic modes, where '0' represents the state that the corresponding app does not use a particular permission, '1' represents the state that the corresponding app uses a particular permission, and '→' represents a state transition. In Table 9, we tabulate the macro-patterns we observed in the stable dataset, along with their frequencies. We found that the permission additions dominate the permission changes (0→1 has a 90.46% frequency), as pointed out earlier in Section 5.1. We also found occurrences of other interesting patterns, e.g., permissions being deleted and then added back, though these instances are much less frequent.

Micro patterns. Some `Dangerous` permissions appear to be confusing developers. For example, the location permissions `ACCESS_COARSE_LOCATION` and `ACCESS_FINE_LOCATION`, provide different levels of location accuracy, on GSM/WiFi position and GPS location, respectively. Location tracking has been heavily debated because it could possibly be used to violate the user's privacy. We found that app developers handled the adding and deleting of these `Dangerous` location permission in an interesting way; to reveal the underlying evolution patterns of used by the `Dangerous` location permissions, we have done a case study of micro-patterns on two widely used location permissions, `ACCESS_COARSE_LOCATION` and `ACCESS_FINE_LOCATION`. We found that, although the most frequent macro evolution pattern of location permission is 0→1, the micro evolution patterns of the location permissions are quite diverse.

In Table 10, we tabulate the micro-patterns we observed for the location permission alone. For instance, 0→Both→Fine means both location permissions are used at first, then the `ACCESS_COARSE_LOCATION` permission is deleted in a later version of the app. 0→Fine→0→Fine shows the app added `ACCESS_FINE_LOCATION` at first, dropped it in a subsequent version, and finally, added back again. Though the table indicates several micro-patterns, note that using both location permissions dominates, with 50% of the total, which shows that more and more apps tend to include both location permissions for location tracking. We are able to make two observations. First, evolution patterns requesting `Dangerous` permissions clearly show the struggling balance between app usability and user privacy during the evolution of apps. Second, the patterns reveal that developers of third-party apps may be unclear with the correct usages of the `Dangerous` location permissions, which highlights the importance for the platform to be more clear on how to properly handle `Dangerous` permissions.

Micro pattern	Frequency
Both	6.67%
Fine→Both	10.00%
Fine→Coarse	3.33%
Coarse→Both	10.00%
0→Both	20.00%
0→Fine	10.00%
0→Coarse	26.70%
0→Fine→Both	3.33%
0→Both→Fine	3.33%
0→Both→Coarse	3.33%
0→Fine→0→Fine	3.31%

Table 10: Micro evolution patterns for the location permissions; Fine represents the `ACCESS_FINE_LOCATION` permission, Coarse represents the `ACCESS_COARSE_LOCATION` permission, and Both means both Fine and Coarse are used.

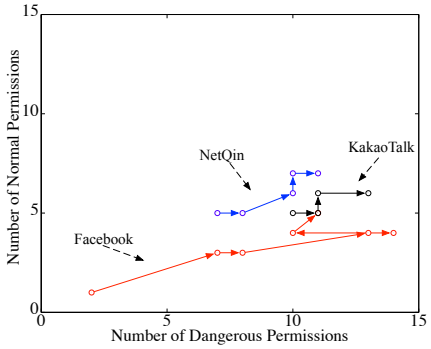


Figure 5: Permission trajectories for popular apps.

Permission trajectories. Due to the observed diverse permission evolution patterns, we plot the number of `Normal` against `Dangerous` permissions to visualize trajectories as apps evolve. We found many interesting trajectories, and highlight three, e.g., Facebook (red), KakaoTalk (black) and NetQin (blue), in Figure 5. Facebook added `Dangerous` permissions in great numbers early on, but recently they have removed many and instead added more slowly. Both NetQin and KakaoTalk continue to add permissions from either one permission level or both permission levels with each new version that is released. These diverse trajectories of popular apps again highlight the need for the the platform to provide

Micro pattern	Frequency
Legitimate →Over	58.57%
Over→Legitimate	32.14%
Over→Legitimate→Over	7.86%
Over→Legitimate→Over→Legitimate	0.71%
Over→Legitimate→Over→Legitimate→Over	0.71%

Table 11: Evolution patterns of the privilege levels of the stable dataset, where Legitimate **represents legitimate privilege and** Over **represents overprivilege.**

Permission	Protection level
GET_TASKS	Dangerous
MODIFY_AUDIO_SETTINGS	Dangerous
WAKE_LOCK	Dangerous
NFC	Dangerous
GET_ACCOUNTS	Normal

Table 12: Most added permissions from the Legitimate→Over (58.57%) subset of apps.

better references of Android permissions to developers.

5.4 Apps Are Becoming Overprivileged

Extra permission usage may lead to overprivilege, a situation in which an app requests the permission, but never uses the resource granted. This could increase vulnerabilities in the app and raise concern of security risks. In this section, we investigate the privilege patterns to determine whether Android apps became overprivileged during their evolution.

To detect overprivilege, we ran the Stowaway [8] tool on the stable dataset (1,703 app versions). As shown in Figure 7, we found that 19.6% of the newer versions of apps became overprivileged as they added permissions, and 25.2% of apps were initially overprivileged and stayed that way during their evolution. Although the overall tendency is towards overprivilege, we could not ignore the fact that 11.6% of apps decreased from overprivileged to legitimate privilege, a positive effort to balance usability and privacy concerns.

In addition, similar to the evolution patterns of permission usage, we also study the evolution patterns of overprivilege status for each app; we present the results in Table 11. We found that the patterns Legitimate→Over and Over→Legitimate dominate at 58.57% and 32.14%, respectively. However, like in the patterns of permission usage, we also found other diverse patterns during the evolution of apps, which again shows that there may be confusion for third-party developers when deciding on what permissions to use for their app.

In Table 12 and 13, we further refine the observations to show the kinds of permissions involved in the dominating patterns: we observe that `Dangerous` permissions are the major source that causes an app to be overprivileged, which again emphasizes that developers should exercise more care when requesting `Dangerous` permissions.

6. PRE-INSTALLED APPS

Pre-installed apps have access to a richer set of higher-privileged permissions, e.g., at the `Signature` and `signatureOrSystem` levels, compared to third-party apps, which gives pre-installed apps access to more personal information on the device [11]. Thus, we should investigate how Android permissions are used in pre-installed apps. We conducted a permission-change analysis for pre-installed apps

Permission	Protection level
READ_PHONE_STATE	Dangerous
ACCESS_COARSE_LOCATION	Dangerous
WRITE_EXTERNAL_STORAGE	Dangerous
ACCESS_MOCK_LOCATION	Dangerous
VIBRATE	Normal

Table 13: Most dropped permissions from the Over→Legitimate (32.14%) subset of apps.

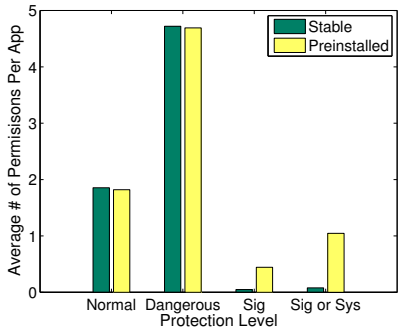


Figure 6: Average number of permissions per app, for each protection level, from stable and pre-installed datasets.

in a manner similar to the stable dataset. We present the results in Figure 4. Figures 4(a) and 4(b) indicate that permission usage is relatively constant, e.g., 62.61% of pre-installed apps do not change their permissions at all, which is significant when compared to our third-party apps with only 15.68%. Further, from Figure 4(c) and 6, pre-installed apps request many more `Signature` and `signatureOrSystem` level permissions than third-party apps, while at the same time requesting nearly just as many `Normal` and `Dangerous` level permissions. This shows that pre-installed apps have a much higher capability to penetrate the smartphone. Interestingly, the vendors also have the ability to define their own permissions inside the platform when they customize the Android platform for their devices. For example, HTC defines its own app update permission, `HTC_APP_UPDATE`.

The power of pre-installed apps requires great responsibility by vendors to ensure that this power is not abused. On one hand, vendors are able to customize pre-installed apps to take full advantage of all the hardware capabilities of the device, as well as create a brand-personalized look-and-feel to enhance user experience. On the other hand, users cannot opt out of pre-installed apps, and in most cases, cannot uninstall the pre-installed apps, which raises the question: *why should users be forced to trust pre-installed apps?* Hindering that trust is our finding that, despite being developed by vendors, 66.1% of pre-installed apps were overprivileged.

What if the power of pre-installed apps is used against the user with malicious intent? For example, the marred pre-installed app `HTCLogger` and other reported security compromised apps have already indicated such security risks do exist and can significantly damage the smartphone and/or the user data [5, 11]. The vendors' `Signature` and `signatureOrSystem` level permissions can be exploited by malicious apps to do an array of damaging actions, such as wiping out user data, sending out SMS messages to premium numbers, recording user conversations, or obtaining the device location data of the device [11].

As we analyzed the evolution of Android platform permis-

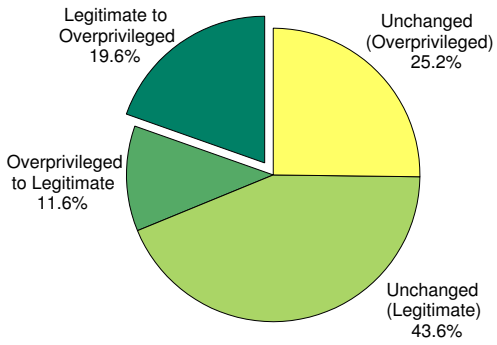


Figure 7: Overprivilege status and evolution in the stable dataset.

sions, it was interesting to see the evolution trends benefit vendors, rather than users. With the power vendors have in pre-installed apps, developers of pre-installed apps should be more careful in their development as they represent the trusted computing base (TCB) of the Android ecosystem. Up until now, there has not been any clear regulations or boundary definitions that protect the user from pre-installed apps. We argue that, since pre-installed apps have more power and privilege over Android devices, vendors need to realize their responsibility to protect the end-user.

7. RELATED WORK

None of the prior works on Android permissions has focused on understanding how Android permissions and their use evolve in the Android ecosystem.

Android permission characterization and effectiveness. Barrera et al. [9] introduced a self-organizing method to visualize permissions usage in different app categories. A comprehensive usability study of Android permissions was conducted through surveys in order to investigate Android permissions' effectiveness at warning users, which showed that current Android permission warnings do not help most users make correct security decisions [6]. Chia et al. [13] focused on the effectiveness of user-consent permission systems in Facebook, Chrome, and Android apps; they have shown that app ratings were not a reliable indicator of privacy risks.

Permission-related Android security. Enck et al. [17] presented a framework that read the declared permissions of an application at install time and compared it against a set of security rules to detect potentially malicious applications. Ongtang et al. [12] described a fine-grained Android permission model for protecting applications by expressing permission statements in more detail. Felt et al. [8] examined the mapping between Android API's and permissions and proposed Stowaway, a static analysis tool to detect over-privilege in Android apps. Permission re-delegation attacks were shown to perform privileged tasks with the help of an app with permissions [7]. Grace et al. [11] used Woodpecker to examined how the Android permission-based security model is enforced in pre-installed apps and stock smartphones. Capability leaks were found that could be exploited by malicious activities. DroidRanger was proposed to detect malicious apps in official and alternative markets [19]. Zhou et al. characterized a large set of Android malwares, e.g., accumulating fees on the devices by subscribing to premium services by abusing SMS_related Android permissions [18]. An effective framework was developed to defend against privilege-escalation attacks on devices [15].

8. CONCLUSION

We have investigated how Android permission and their use evolve in the Android ecosystem via a rigorous study on the evolution of the platform, third-party apps, and pre-installed apps. We found that the ecosystem is becoming less secure and offer our recommendations on how to remedy this situation. We believe that our study is beneficial to researchers, developers, and users, and that our results have the potential to improve the state of practice in Android security.

Acknowledgements

This work was supported in part by National Science Foundation award CNS-1064646, by a Google Research Award, by ARL CTA W911NF-09-2-0053, and by DARPA SMISC Program W911NF-12-C-0028.

9. REFERENCES

[1] Freewarelovers, May 2012. http://www.freewarelovers.com/android.
[2] Google Play. https://play.google.com/store, May 2012.
[3] Android. Android-defined Permission Category. http://developer.android.com/reference/android/Manifest.permission_group.html, May 2012.
[4] Android Developer. Android API. http://developer.android.com/guide/appendix/api-levels.html, May 2012.
[5] Android Police. Massive Security Vulnerability In HTC Android Devices. http://www.androidpolice.com/2011/10/01/massive-security-vulnerability-in-htc-android-devices, October 2011.
[6] A.P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin and D. Wagner. Android Permissions: User Attention, Comprehension, and Behavior . In *SOUPS*, 2012.
[7] A.P. Felt, H. Wang, A. Moshchuk, S. Hanna and E. Chin. Permission Re-Delegation: Attacks and Defenses. In *USENIX Security Symposium*, 2011.
[8] A.P.Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android Permissions Demystified. In *ACM CCS*, 2011.
[9] D. Barrera, H.G. Kayacik, P.C. van Oorschot and A. Somayaji. A Methodology for Empirical Analysis of Permission-based Security Models and its Application to Android. In *ACM CCS*, 2010.
[10] Google. Android Open Source Project, May 2012. http://source.android.com/.
[11] M. Grace, Y. Zhou, Z. Wang, and X. Jiang. Systematic Detection of Capability Leaks in Stock Android Smartphones . In *NDSS*, 2012.
[12] M. Ongtang, S. McLaughlin, W. Enck and P. McDaniel. Semantically Rich Application-Centric Security in Android. In *ACSAC*, 2009.
[13] P. H. Chia, Y. Yamamoto, and N. Asokan. Is this App Safe? A Large Scale Study on Application Permissions and Risk Signals . In *WWW*, 2012.
[14] P. Pearce, A.P. Felt, G. Nunez and D. Wagner. AdDroid: Privilege Separation for Applications and Advertisers in Android . In *ACM AsiaCCS*, 2012.
[15] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. Sadeghi, and B.Shastry. Towards Taming Privilege-Escalation Attacks on Android . In *NDSS*, 2012.
[16] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A Study of Android Application Security. In *USENIX Security Symposium*, 2011.
[17] W. Enck, M. Ongtang and P. McDaniel. On Lightweight Mobile Phone Application Certification. In *ACM CCS*, 2009.
[18] Y. Zhou and X. Jiang. Dissecting Android Malware: Characterization and Evolution. In *IEEE S &P*, 2012.
[19] Y. Zhou, Z. Wang, Wu Zhou and X. Jiang. Hey, You, Get off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets . In *NDSS*, 2012.