Sleeping Android: Exploit through Dormant Permission Requests

James Sellwood

Supervisor: Jason Crampton

Submitted as part of the requirements for the award of the MSc in Information Security at Royal Holloway, University of London.

I declare that this assignment is all my own work and that I have acknowledged all quotations from the published or unpublished works of other people. I declare that I have also read the statements on plagiarism in Section 1 of the Regulations Governing Examination and Assessment Offences and in accordance with it I submit this project report as my own work.

Signature:
Date: August 28, 2012

Contents

List of Figures

List of Tables

Executive Summary

This report begins by providing a very brief history of telecommunications, followed by background into the usage and capabilities of modern smartphone devices. As this report focuses on information security considerations, it highlights some of the security requirements associated with the significant usage of these ubiquitous devices. The aspect of mobile device information security that is considered within this report is the security architectures employed to achieve those security requirements. In particular, the major security relevant architectures within the Android platform are described: Sandboxing, Permissions and App Stores. Following the description of these significant security concepts, a number of detailed investigations into the Android permission architecture are performed and discussed. The investigations are structured around four distinct topics.

The Permission Categorisation Investigation looks at how Android permissions are organised both regarding severity, through the use of protection levels, and regarding logical subject area, through the use of permission groupings. The 130 currently documented Android permissions are investigated through the development and use of a test app, *Permission Test*, which allows them to be categorised. This investigation provides information not currently documented within the Android documentation and thereby not obvious to developers.

Following the categorisation of the 130 permissions on Android 4.0.4 (Ice Cream Sandwich), a Permission Evolution Investigation is performed tracking the categorisation and labelling of the permissions across six of Android's platform versions. The versions tested cover the Android API versions 8, 10, 13, 14, 15 and 16 — platforms Froyo through to the recent release, Jelly Bean (Android's platform versions are named alphabetically after desserts). This investigation highlights the changes that have occurred across platform releases and in particular identifies several potential discrepancies in the Android documentation.

Having built up considerable information and understanding through the first two investigations, the third investigation looks specifically at Third-party Permission Requests. Unlike the 130 permissions already discussed, third-party permission requests are defined by app developers themselves and are used to provide limited authorisation in relation to Inter-Process Communication (IPC). This investigation makes use of two more test apps, *Permission Test Creator* and *Permission Test Requestor*, and helps frame a hypothesis which forms the basis for the final investigation.

The most significant investigation of this report identifies the existence of a weakness in the Android permission architecture, exploitable through the use of Dormant Permission Requests. A fourth test app, *Permission Test Jelly Bean*, is developed and demonstrates exploiting this weakness. The repercussions and potential mitigation of the weakness are then discussed.

The conclusion highlights related work in this field, covering relevant research which has been of significant value. Whilst other work in this field has discussed several mechanisms through which malicious apps can exploit the Android permission architecture, to my knowledge the Dormant Permission Requests weakness I describe in this project has not previously been identified. I believe this contribution to therefore be unique within the existing body of knowledge. Finally, ongoing and future work is introduced with this report as the principle basis.

Chapter 1

Introduction

1.1 A Very Brief History of Telecommunications

The first electrical telegraph systems were developed in the late 1830s in the UK, by Charles Wheatstone and William Cooke, and in the US by Samuel Morse [105]. Initial significant installations were put in place, over distances of around ten miles, but relied on physical wires to transmit the electrical signal from one end to the other. Following on from the development of the electrical telegraph and in particular the use of electromagnets, Alexander Graham Bell communicated clearly using the first telephone in 1876 [105]. This was the birth of electrical telecommunications; the next fundamental step was taken as the 20th century began, when Guglielmo Marconi (having been working on wireless since he was 21 years old) transmitted the first message across the Atlantic — over a distance of 2100 miles [29].

Advances continued through the early 1900s with a significant drive coming from the invention of the transistor in 1947 by John Bardeen, Walter Brattain and William Shockley at Bell Labs [59]. Amplification was a necessity to break through the otherwise crippling limitation of the weakening of radio signals as they traveled further distances. Up until then, it had been provided by the larger, less reliable and more energy demanding vacuum tubes. Whilst devices used for computation were developed far before the transistor, the first electrical computer to make sole use of transistors (and no vacuum tubes) was the IBM 608 calculator announced in 1955 [33]. The introduction of transistor only computers allowed them to be significantly more compact and more reliable, a change which would continue through the years to today's lightweight portable computing devices.

The mobile telephone started its life as a car telephone in 1946 [19] with the first personal mobile telephone call made by Martin Cooper from AT&T in 1973 [2]. The existence of a cellular network allowed Martin Cooper to call his opposite number Joel Engel, also working on the technology, at Bell Labs. Following the initial use of analog networks, digital networks in the form of GSM started to be adopted in 1990 with the first call made in 1991 [20]. At this time communications were just voice, with SMS (Short Messaging Service) introduced in 1992. GSM mobile data communications were enhanced with the introduction of GPRS (General Packet Radio Service) in 2000.

The '90s saw a dramatic increase in the development of mobile phones as well as the companies which have since become household names, their logos branded on devices in many people's pockets. Ericsson began as a telegraph repair shop in 1876 with their first mobile system produced in 1981 [30]. Samsung, which had developed its first computer in 1983, developed its first mobile handset in 1991 [50]. Nokia in contrast, started out as a maker of paper in 1865 but went on to focus on electronics and to produce the first retail GSM (Global System for Mobile) mobile in 1992 [58]. HTC Corp, was founded in 1997 [3] and has since been at the leading edge of mobile devices, whilst the most recently launched, prominent company is RIM, who produced their first BlackBerry in 1999 [4]. Apple was incorporated in 1977 [14] producing computer parts and whilst it has had possibly the greatest impact so far on the mobile phone and its usage, it was the last of these big names to produce a mobile handset. The introduction of the iPhone in 2007 ultimately heralded a world of smartphones and rather rapidly changed the way we use mobile devices for ever [16]. With the release of the iPhone 3G, Apple also introduced the App Store [13] — bringing applications to consumers in a way that had never been done before. The App Store concept has since been duplicated my multiple mobile platform vendors to produce the likes of Google Play (until recently, known as Android Market), Microsoft Marketplace and Nokia Ovi Store.

All in all these developments took place over a period of some 180 years but the advancements of the last 30 years have seen a step change in technology, usability and the penetration of mobile devices that far exceeds anything that has gone before. The International Telecommunication Union estimated in November 2011 that there were a little under 6 billion mobile subscriptions in the world, over double the 2.7 billion which there had been in 2006 [40]. This number includes an estimated 119.5 subscriptions per 100 inhabitants within Europe — nearly 120% penetration [43].

1.2 Mobile Internet and Ubiquitous Connectivity

Whilst the development of telecommunications, computers and mobile phones were all significant and wide reaching advancements in terms of our technological capability, it was another development which started the revolution in the distribution and access to information. The ARPANET, the Advanced Research Projects Agency Network, was decommissioned in 1990 (having been started in 1969) following successful initial demonstrations and limited usage of the Internet [82] as a TCP/IP based packet switching network. Growth of the Internet provided connectivity like never before but it was the invention of the World Wide Web (WWW or Web) by Tim Berners-Lee that brought together the concepts of hypertext, the linking of documented content to get around the inherent linearity of paper, and the newly available connectivity to produce a global publishing technology. With the Web, publishers of content could link relevant information together in a manner such that the reader could follow the logical story path that had been laid out for them, stepping from one piece of information to the next.

As the Internet and the Web matured and as computers became more powerful, advanced graphical technologies became integrated with the initial HTML standard (Hypertext Markup Language) for consumption through the Web. Soon video and 3D rendering were possible and with ever increasing communications capabilities this kind of content became ever more popular. Today the web site YouTube, which hosts video uploaded and viewed by people from all over the world, receives 60 hours of video uploaded every minute and over 800 million unique visitors a month watching over 3 billion hours of video [61]. The site was originally started in December 2005 [62] following a beta launch in May. As well as being popular on home computers, YouTube's mobile site, launched in June 2007 [62] gets over 600 million views a day after traffic tripled in 2011 [61]. This leap in visitors is no doubt attributable to the growth in popularity and connectivity in modern smartphones.

Mobile devices are today commonly sold with a myriad of connectivity technologies. The new iPad, released in March 2012, is not atypical in its support for Bluetooth (version 4.0), mobile data in the form of GSM/EDGE (850, 900, 1800 and 1900MHz), UMTS/HSPA/HSPA+/DC-HSDPA (850, 900, 1900, 2100MHz) and 4G LTE (700 and 2100MHz) as well as Wi-Fi (802.11a, b, g and n) [15]. This is not withstanding the physical USB based connector and the 3.5mm audio port. Devices commonly also support GPS (Global Positioning System) for location based services and a myriad of sensors.

This wide selection of communications capabilities allows mobile devices of today to establish and maintain connections to a variety of devices, as well as the Internet, as their owners make their way through their daily lives.

1.3 Mobile Applications

According to Oxford Dictionaries, a smartphone is “a mobile phone that is able to perform many of the functions of a computer, typically having a relatively large screen and an operating system capable of running general-purpose applications” [54]. Other definitions are similar, with a recurring emphasis on the theme of a mobile phone being capable of running applications [53, 51, 52]. These definitions should be compared to that of a feature phone, defined by Oxford Dictionaries as “a mobile phone that incorporates features such as the ability to access the Internet and store and play music but lacks the advanced functionality of a smartphone” [25]. One could be forgiven for struggling to explicitly distinguish between the two, especially as modern feature phones are becoming more feature rich, making them more attractive and functional for those unable or unwilling to purchase a smartphone. Figure 1.1 highlights, for a number of countries, the month when smartphone devices started outselling feature phones [73].

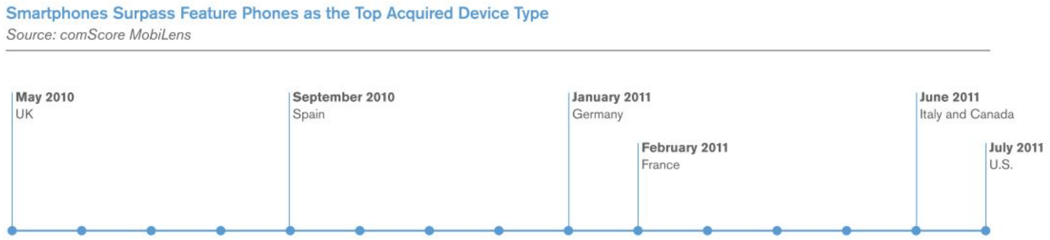


Figure 1.1: Smartphones Surpass Feature Phones as the Top Acquired Device Type

The defining capability that all smartphones do possess, as do some feature phones to a lesser extent, is the capacity to download and install any of a wide variety of applications to further enhance the device’s functions and usefulness. Prior to the introduction of Apple’s App Store, whilst this process was possible, the discovery of new applications was haphazard. Whilst individual developers would provide their own software for download, consumers had to search amongst all other Internet content for applications suitable for their device and their needs. The concept of a mobile application store brought together a wide variety of applications (as well as digital content such as books, music and films) into a single, searchable, categorised store such that consumers can easily locate and download anything on offer. These application stores are huge too. In March 2012 Apple’s App Store reached 25 billion downloads to 315 million devices worldwide in an app store which has over 550,000 applications to choose from [18]. The month before, Google announced that the Android Market (as it was still known at that time) had in store more than 450,000 applications available to more than 300 million devices [6] having previously announced in December 2011 that it had reached 10 billion downloads [1].

To make finding applications easier and to allow for browsing of the app stores, mobile app stores are categorised. Categories commonly exist for books, business, games, productivity and social applications amongst the many others. As many applications attempt to provide some form of personalised service to the user it is frequently necessary to enter at least some basic information into the application to configure or make use of it. This combined with the fact that “smartphones are often uses for privacy-sensitive tasks” [98] means that in addition to the already present telephone contact information it is common to find far more personal information such as calendar entries, e-mail messages, social networking status information and even usernames and passwords [106].

1.4 Device Usage

Whilst it was once said that “the killer app for phones is voice” [86], as the capability of devices continue to grow consumers are making use of their mobile phones for an ever increasing number of activities. These capabilities are backed by manufacturers producing ever more complex combinations of hardware with the mobile phone industry being one of the most competitive and rapidly developing electronics markets. At a recent mobile security conference, Mike Gibson (Director, Enterprise: UK & Ireland at RIM) emphasised this point with the statement "A year in smartphones is like six years in other industries" [91]. Mobile devices have become a staple of modern life to such an extent that it is not uncommon to see school children with their own personal mobile phones — something which would have been unheard of some 30 years ago or less. In a similar way consumers don't leave home without their smartphones — 78% in the UK and 80% in the US according to a recent Google study entitled "Our Mobile Planet" [83, 84]. It is worth considering for a moment just some of the many activities that today's smartphones are used to perform on an hourly basis. (Note: This list applies an approximate popularity order, where information available, based on some of the many studies on smartphone usage [31, 32, 56, 73].)

- Text messaging
- Taking photos
- Browsing the Internet
- E-mail
- Telling the time
- Downloading apps
- Alarm clock
- Playing games
- Social networking
- Navigating
- Listening to music
- Watching video
- Voice calling
- Recording video

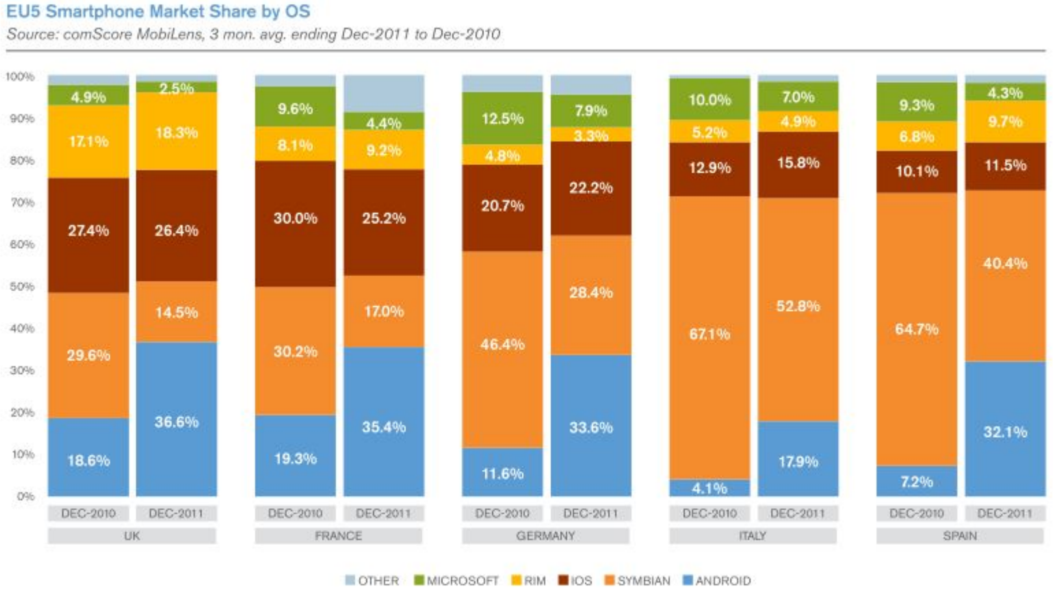
Many of the activities listed above involve the creation, transmission or storage of information. In some cases that information is likely to have little sensitivity as far as a specific individual is concerned — the data that makes up a music track or the content of a public web page for example. That said there is considerable information which is considered sensitive — for example, personal e-mail and text messages or Internet search history. In many cases this distinction may not be so definite or consistent, with the sensitivity changing over time as the specific information changes or as the relationships that that information relates to change. It is therefore often difficult to identify globally accepted rules, which can be successfully applied across wide cross-sections of consumers, identifying what types of information they will consider sensitive. Further to this, sensitivity perception is a spectrum commonly influenced by an individual's culture and upbringing. Thus two individuals can potentially have very different sensitivity viewpoints in identical scenarios. As well as the information associated with these common tasks, consumers may also be using their smartphones specifically to securely store sensitive information. There are numerous 'vault', 'wallet' or 'password keeper' apps which claim to provide secure storage for information such as account details and passwords. Access to such apps is usually secured with a 'master password' which enables the decryption of the information held within the vault.

Ignoring the specific complexity of classifying what information is sensitive to a particular individual, there is no doubt that sensitive personal information commonly exists on smartphones and the increasing usage being seen suggests the volume of this information is only likely to grow. This demand is being supported by the growing device and memory card capabilities resulting in high end smartphones being capable of directly or indirectly (through the use on-board or external memory card storage) providing tens of gigabytes of storage space. It is no wonder that it has been said that "smartphones in general are perhaps the one electronic device that knows the most about an individual" [87]. All this information must be secured to a level acceptable to consumers so as to maintain privacy and avoid the user being a victim of fraud, theft and other illegal activities.

1.5 The Operating System Market

There are a number of operating systems currently in use on smartphone devices with the exact balance of their distribution depending on the country. In European countries, as can be seen in Figure 1.2, the share held by each operating system varies, with the older Symbian OS being particularly popular in Italy and Spain whilst newer operating systems like iOS and Android compete more strongly in UK and France [73]. No matter the country, it is clear that Google's Android has seen significant growth in recent years with on average a tripling of market share seen between December 2010 and December 2011 over the European countries shown.

Note: comScore confusingly fails to be consistent in the labelling of both Figure 1.2 and 1.3. Android and iOS are the full names of the operating systems from Google and Apple respectively. Symbian and Palm are shortened versions of the operating system names Symbian OS and Palm OS whilst also being part of the company names Symbian Limited and Palm Incorporated. RIM's operating system is called BlackBerry OS whilst Microsoft have actually had several mobile platform operating systems including two during the time period concerned — Windows Mobile and Windows Phone. For consistency when referring to these figures I have used comScore's naming convention — even though this involves referring to company names as operating systems in some cases.



EU5 Smartphone Market Share by OS

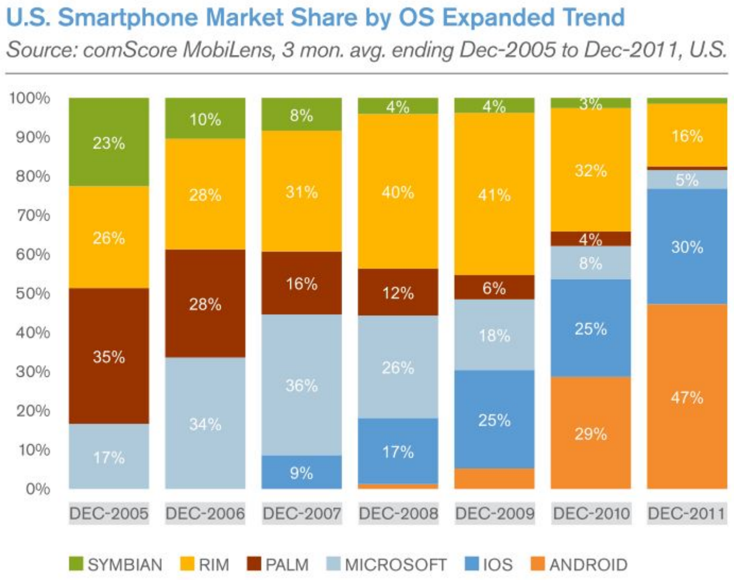
Source: comScore MobiLens, 3 mon. avg. ending Dec-2011 to Dec-2010

Figure 1.2: EU5 Smartphone Market Share by OS

Figure 1.3 highlights the trends in operating system share experienced over the last seven years in the US [73]. There have been several dramatic changes in that time with Palm, Microsoft and Symbian all losing significant amounts of their market share to the rising giants of iOS and Android.

Historically the operating systems seen in use in 2005 (from Figure 1.3) where far more limited from a user's point of view than those taking the lion's share in 2011. Whilst each of those operating systems (Symbian, RIM, Palm and Microsoft) allowed the installation of applications, the lack of centralised app stores, as introduced by Apple in 2008, meant that the process of discovery, purchase and installation was a disjointed one. I believe that the success of today's leading smartphone operating systems is in no small part driven by the mobile app culture that has developed from the creation of the centralised app store and the ease of digital consumption which flows from this.

Through the rest of this report I will focus on the Google Android platform whilst also providing considerable reference to Apple iOS. The reasons for focusing on Android are firstly, as attested by Figure 1.2 and 1.3, it is currently the most popular operating system in the UK and US and secondly as it is an open platform there is considerable information available and research targeting



U.S. Smartphone Market Share by OS Expanded Trend

Source: comScore MobiLens, 3 mon. avg. ending Dec-2005 to Dec-2011, U.S.

Figure 1.3: US Smartphone Market Share by OS (Expanded Trend)

it. There are many similarities, as well as some interesting disparities, with iOS which make this an interesting platform for comparison.

1.6 Security Requirements

As consumers have seen benefit from the use of smartphones in their personal lives these devices have become integrated into their daily activities. In the UK, 59% have used their smartphone every day in the past seven days whilst in the US the figure is slightly higher at 62% [83, 84]. This level of integration has resulted in an expectation within consumers that smartphone devices can bring benefit to working activities in the same way as they do to personal ones. A recent Cisco study entitled “The Future of Work” found that seven out of ten college students “believe that company-issued devices should be allowed for personal and business use because of the blending of work and personal communications in their daily lifestyles” [72]. The same study also found that 81% of college students “want to choose the device for their jobs — either budget to buy their own or use their own personal device”.

It is clear that to many people, smartphones are both popular and important, holding significant amounts of personal and corporate information in an extremely portable form factor. Users are reliant on the security of these devices to protect this information. In reality, it is not just the device, but the apps which they download which must provide the necessary security features to ensure user information is kept safe. Whilst this may be the need, a study by viaForensics identified “10% of apps stored passwords in plain text, perhaps the most direct threat to user security in this study” having been “able to recover 76 out of 100 Usernames for apps tested” [106]. Even when not in plain text such information may still be at risk. Researchers at Zvelo were able to identify the location of the hash of the user’s PIN for the Google Wallet app and subsequently discovered the PIN’s value through a quick exhaustive search of the possible 10,000 hashes [28].

Consumers rely on the manufacturers of smartphones and the developers of their apps to provide protection that might be deemed ‘normal’ or ‘suitable’ given the application. For example, a recent small study found that “most users think that software vendors should be most responsible for the security on mobile devices” [68]. In some sense, consumers’ expectations that such protection should be built-in are not unwarranted; after all we rely on similar expectations with regard to many other products and services. This ‘trust’ in the existence and proper functioning of a basic level of protection could be considered equivalent to what is expected of modern automobiles, building security systems or lift control systems. That said, smartphones are both multifunction devices (in fact they could be considered the quintessential multifunction device) and designed to support extensibility through both hardware and software additions. The need to support such a variety and flexibility of operations has dramatic affect on how intrinsically 'locked-down' or secure something can be. It is also important to remember that security features often impact the usability of a product and consumers rarely, if ever at all, buy products with a security over usability mindset.

The next chapter, Chapter 2, looks at the security architectures found in the Android operating system in order to highlight the mechanisms employed to protect user information. Subsequent to this, Chapter 3 looks in detail at the permissions architecture within Android and shows how the permissions have evolved through recent versions of the operating system. Chapter 4 then identifies a specific weakness in the permissions architecture and, through the use of a test app, demonstrates how this weakness can be exploited. The repercussions of this exploit and potential mitigation is also discussed. My conclusions, along with related and future work are covered in Chapter 5.

1.7 Motivation & Objective

I have been interested in information security for a number of years, having gravitated towards the subject as security concerns became ever more obvious and important to me through the many other computer-related subjects I have worked with. I have always had an aptitude towards technology and a deep need to understand the exact workings of those technologies I come across. This desire, combined with a highly procedural approach to my learning and research, has meant that I have never accepted the 'just because' answer and have chased down the details involved in technologies in order to understand and visualise their operation. This detailed understanding and inquisitive approach has often led to more questions and 'what-if' scenarios which align well to security research.

My objective through this report has been to direct my attention towards the inner workings of the Android permissions architecture. Mobile devices are becoming ever more prominent in daily life and their features and capabilities continue to grow. This increased reliance on them amongst the general population and my existing fascination with technology in general, and mobile in particular, identified it as an excellent target for investigation. The Android permissions architecture whilst easy to understand at a high-level, at least for technical individuals, is not fully documented and has evolved in recent years across versions of the Android platform. I therefore have chosen to investigate the detail of Android permissions and the evolution of the architecture, identifying differences between platform versions. Where specific differences are identified, and found to be worthy of further research, specific investigations will be carried out to explore deeper into those platform variations.

Chapter 2

Android Security Architectures

2.1 Sandboxing

One of Android's core security architecture components is the way that installed apps run. Apps are sandboxed from one another — partitioned so as to prevent unauthorised interference. This is done through the use of standard Linux capabilities associated with user and group memberships but it is also related to the Dalvik Virtual Machine (VM), in which apps run. That said, the Android developer documentation is very clear (in several places) that "the Dalvik VM is not a security boundary" [23, 45] although this is specifically referring to the VM boundary itself. The VM allows interaction from inside to outside, thus allowing an app to have a native code component running on the device. This facility precludes the Dalvik VM from being a security boundary itself, however, it is directly involved with the sandboxing process.

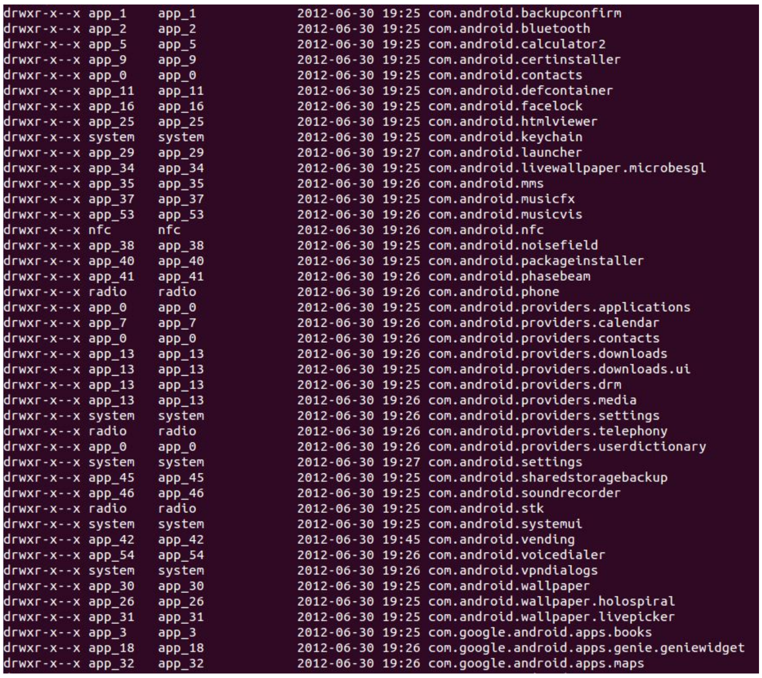
Firstly each "Android application runs in a separate Dalvik virtual machine in its own process context" [90]. These instances of Dalvik are spawned from a parent process called Zygote. This provides process separation of the apps, reducing the likelihood of issues such as "buffer overflows, remote code execution, and stack smashing" [101]. Secondly apps "are stored in directories that are assigned unique Linux UIDs during installation" [90]. In fact both a User ID and Group ID are created for each freshly installed app as seen in Figure 2.1. When the app is run in its Dalvik instance, that process is run using the app's specific User ID. This results in each app being limited to accessing the files in its own directory with this access control managed by the underlying Linux kernel.

Note that several directories in Figure 2.1 break the standard with regards to a unique app based ID. These directories relate to certain system installed apps which come pre-installed with the operating system and employ different rules because they must be accessible to other apps.

As further protection the production builds of Android that come on smartphone devices do not include a `'su'` (switch user) command binary, thereby preventing elevation of privileges by switching to a different (e.g. root) account once running under their specific user account. Part of the process of 'rooting' an Android device therefore involves loading an `'su'` binary onto the device for this very purpose.

Using these sandboxing mechanisms, Android segments each app, controls its access and manages its scope so as to protect both the operating system and other apps. That said, there is one way in which apps can interact with each other's files. "During file creation, the application may explicitly define that the created file should be readable or writable by other applications as well" [90]. This is done through the use of `'MODE_WORLD_READABLE'` or `'MODE_WORLD_WRITEABLE'` and is obviously a specific choice made by the developer (or the user if the developer defers the choice to them). Usually these modes are used in circumstances such as when an e-mail or web app downloads files which the user may want to access through other apps on the device.

Android also makes use of external storage which is designed for shared files, and thus can not be considered as belonging to one app or another. Anything saved here is automatically world



```
drwxr-x--x app_1    app_1              2012-06-30 19:25 com.android.backupconfirm
drwxr-x--x app_2    app_2              2012-06-30 19:25 com.android.bluetooth
drwxr-x--x app_5    app_5              2012-06-30 19:25 com.android.calculator2
drwxr-x--x app_9    app_9              2012-06-30 19:25 com.android.certinstaller
drwxr-x--x app_0    app_0              2012-06-30 19:25 com.android.contacts
drwxr-x--x app_11   app_11             2012-06-30 19:25 com.android.defcontainer
drwxr-x--x app_16   app_16             2012-06-30 19:25 com.android.facelock
drwxr-x--x app_25   app_25             2012-06-30 19:25 com.android.htmlviewer
drwxr-x--x system   system             2012-06-30 19:25 com.android.keychain
drwxr-x--x app_29   app_29             2012-06-30 19:27 com.android.launcher
drwxr-x--x app_34   app_34             2012-06-30 19:25 com.android.livewallpaper.microbesgl
drwxr-x--x app_35   app_35             2012-06-30 19:26 com.android.mms
drwxr-x--x app_37   app_37             2012-06-30 19:25 com.android.musicfx
drwxr-x--x app_53   app_53             2012-06-30 19:26 com.android.musicvis
drwxr-x--x nfc      nfc                2012-06-30 19:26 com.android.nfc
drwxr-x--x app_38   app_38             2012-06-30 19:25 com.android.noisefield
drwxr-x--x app_40   app_40             2012-06-30 19:25 com.android.packageinstaller
drwxr-x--x app_41   app_41             2012-06-30 19:26 com.android.phasebeam
drwxr-x--x radio    radio              2012-06-30 19:26 com.android.phone
drwxr-x--x app_0    app_0              2012-06-30 19:25 com.android.providers.applications
drwxr-x--x app_7    app_7              2012-06-30 19:26 com.android.providers.calendar
drwxr-x--x app_0    app_0              2012-06-30 19:26 com.android.providers.contacts
drwxr-x--x app_13   app_13             2012-06-30 19:26 com.android.providers.downloads
drwxr-x--x app_13   app_13             2012-06-30 19:25 com.android.providers.downloads.ui
drwxr-x--x app_13   app_13             2012-06-30 19:25 com.android.providers.drm
drwxr-x--x app_13   app_13             2012-06-30 19:26 com.android.providers.media
drwxr-x--x system   system             2012-06-30 19:26 com.android.providers.settings
drwxr-x--x radio    radio              2012-06-30 19:26 com.android.providers.telephony
drwxr-x--x app_0    app_0              2012-06-30 19:26 com.android.providers.userdictionary
drwxr-x--x system   system             2012-06-30 19:27 com.android.settings
drwxr-x--x app_45   app_45             2012-06-30 19:25 com.android.sharedstoragebackup
drwxr-x--x app_46   app_46             2012-06-30 19:25 com.android.soundrecorder
drwxr-x--x radio    radio              2012-06-30 19:25 com.android.stk
drwxr-x--x system   system             2012-06-30 19:25 com.android.systemui
drwxr-x--x app_42   app_42             2012-06-30 19:45 com.android.vending
drwxr-x--x app_54   app_54             2012-06-30 19:26 com.android.voicedialer
drwxr-x--x system   system             2012-06-30 19:26 com.android.vpndialogs
drwxr-x--x app_30   app_30             2012-06-30 19:25 com.android.wallpaper
drwxr-x--x app_26   app_26             2012-06-30 19:25 com.android.wallpaper.holospiral
drwxr-x--x app_31   app_31             2012-06-30 19:25 com.android.wallpaper.livepicker
drwxr-x--x app_3    app_3              2012-06-30 19:25 com.google.android.apps.books
drwxr-x--x app_18   app_18             2012-06-30 19:26 com.google.android.apps.genie.geniewidget
drwxr-x--x app_32   app_32             2012-06-30 19:26 com.google.android.apps.maps
```

Figure 2.1: Partial directory listing showing User and Group IDs (e.g. `app_1`)

readable so as to allow for the reading of files on other devices when the memory card is transferred. Technically some Android devices have built-in external storage which is not removable, but we will ignore that nuance for now. By default there are a set of directories within the external storage which give an idea to the kind of information that this facility may be used for:

- '`Music/`'
- '`Podcasts/`'
- '`Ringtones/`'
- '`Alarms/`'
- '`Notifications/`'
- '`Pictures/`'
- '`Movies/`'
- '`Download/`'

Whilst these protections are in place there are many reasons why apps need to be able to interact. Without any ability to cross-communicate how could a user, for example, trigger a web page to open via a URL in an e-mail? In order that such interactions can occur but are carefully marshalled, Android provides an Inter-Process Communication (IPC) mechanism passing 'Intents'. Whilst this IPC does support a filtering mechanism, this is not an access control or security feature but is instead designed as a means to determine a destination app's capabilities.

The sandboxing provided within the Android platform provides a considerable foundation for the security requirements discussed previously. In limiting an app's access to both the data and processes of other apps, the sandboxing greatly reduces the risk of unauthorised disclosure or modification of sensitive information. This protection does have boundaries though, as has been highlighted above, with the ability for files to be written as world readable/writable, either explicitly or through the use of external storage. This is one important case where the user is likely to be reliant on the practices of the app developer. If the developer has, for whatever reason, coded their app so that sensitive files are stored in this manner then the information will be exposed. Whilst the default directories, within the external storage, align with information of the types likely desired as accessible to multiple apps, there may be circumstances where such information is considered sensitive by a user. As discussed previously any sensitivity likely depends upon subject matter, context and the user's viewpoint.

iOS Comparison Apple iOS also employs sandboxing which it defines as "a system feature that provides fine-grained control of the ability of processes to gain access to system resources, therefore limiting the amount of damage that can be done by a malicious hacker who takes control of an app" [65]. In iOS, any third-party app is sandboxed by placing "each app (including its preferences and data)"[63] in a unique home directory "which is randomly assigned when the app is installed" [64]. System resources are protected due to the fact that the third-party apps run as a user 'mobile' which is 'non-privileged' as far as the operating system is concerned. These protections are very similar in concept to the sandboxing provided by Android although the fact that all third-party apps run as a single user has been considered by some as a "cause for concern" [93].

2.2 Permissions

We have seen how Android uses sandboxing to control which processes and files a particular app can interact with. A further sandboxing control applied to apps, comes from the fact that they must explicitly request permissions so as to successfully use the API calls which access certain system features. For example there are permissions for accessing location based services (`'ACCESS_COARSE_LOCATION'` and `'ACCESS_FINE_LOCATION'`) as well as for using the device's Internet connection (`'INTERNET'`) [42]. "Each application must declare upfront what permissions it requires" through the setting of the one or more `<uses-permission>` tags within its `'AndroidManifest.xml'` file [94]. Permissions "must be accepted by the user at install time" after the user has been "prompted to accept or deny the permissions requested by the application" within its manifest [67]. This process is atomic in that the "users may only grant all requested permissions or deny them all by not installing the application" [66].

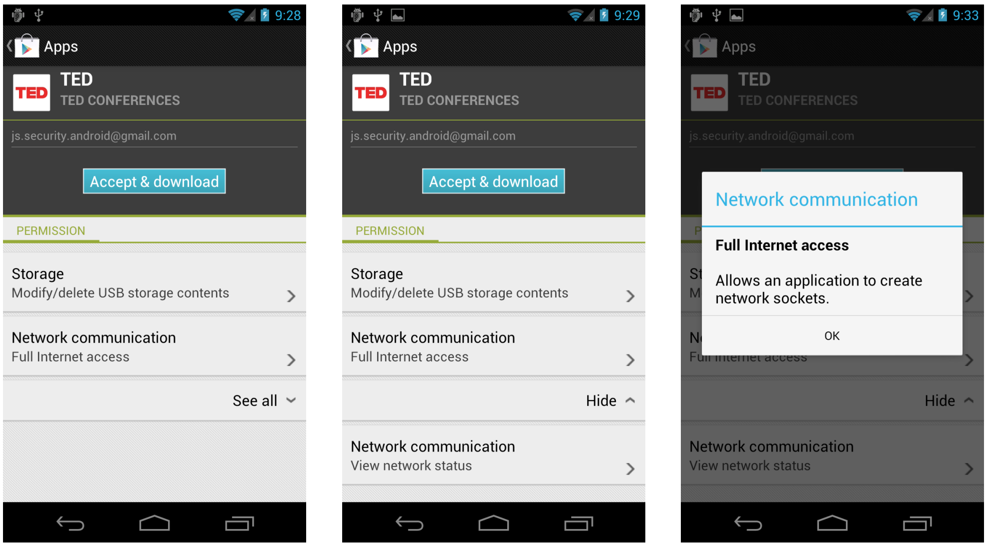
By default only `'dangerous'` permissions are displayed to the user during the pre-installation permission authorisation request (Figure 2.2a), with less worrying (`'normal'`) permissions initially being hidden in a collapsed display area (Figure 2.2b). Along with the permission label and grouping (figures 2.2a and 2.2b), a more user-friendly textual description is also provided (Figure 2.2c) in certain circumstances. When the permissions are accepted the "granted permissions are stored in an in-memory data structure and also serialised to disk" [69].

As of July 2012 there are some 130 permissions currently listed within the Android documentation [42] (see Chapter 3 for a detailed investigation into these permissions). These are manifest permissions, the majority of whose identifier string begin `'android.permission.'` (e.g. `'android.permission.INTERNET'`). Currently there are four manifest permissions which do not fall under the `'android.permission.'` hierarchy. These are:

- `'com.android.alarm.permission.SET_ALARM'`
- `'com.android.browser.permission.READ_HISTORY_BOOKMARKS'`
- `'com.android.browser.permission.WRITE_HISTORY_BOOKMARKS'`
- `'com.android.voicemail.permission.ADD_VOICEMAIL'`

As well as being able to request any combination of these predefined permissions, an app may define and/or request bespoke third-party permissions. Whilst these third-party defined permissions do not restrict access to any underlying Android system functionality, they are designed to be used as part of the marshalling of IPC between apps or app components.

Permissions are defined, by the system or a developer in the case of third-party permissions, under one of four protection levels. The default value is `'normal'` which indicates the lowest risk features. Permissions defined under this protection level are automatically approved without requiring the user's explicit consent. Next is `'dangerous'`, which represents higher risk features.



Network communication

Full Internet access

Allows an application to create network sockets.

(a) ‘dangerous’ only (b) ‘dangerous’ & ‘normal’ (c) user-friendly description

Figure 2.2: Pre-installation permission authorisation prompts

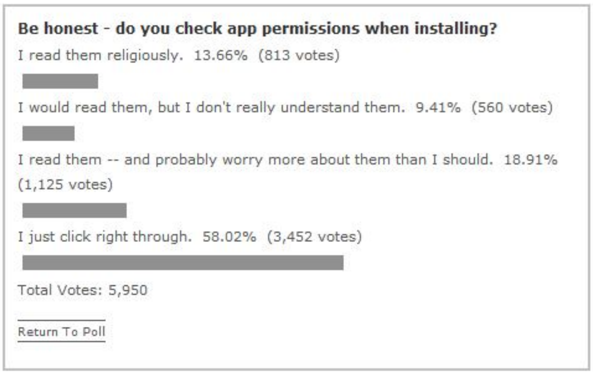
Any permissions marked with this protection level will require explicit approval from the user. The third protection level is ‘signature’. Permissions under this protection level will only be granted by the system to an app which is signed with the same certificate as the app that declared it. In the case of third-party permissions, this ‘signature’ protection level allows a developer to tie permissions to their own apps. The final protection level is a variant of the ‘signature’ level and is called ‘signatureOrSystem’ [44]. Permissions under this protection level are only granted to apps within the Android system image or apps signed with the certificate used to sign apps within the system image. Two further protection flags are mentioned in the documentation (‘system’ and ‘development’ [48]) and are used by certain manifest permissions in Android 4.1. These protection flags are discussed further within Chapter 3.

Whilst the authorisation to use the requested permissions is achieved through the user's acceptance to install an app, the enforcement of these permissions is performed by the Android operating system at runtime. There are, in fact, several instances when permissions are checked, with attempts to use functionality without the necessary permission resulting in a security exception. According to the Android documentation, the specific locations of these permissions checks are [45]:

- At the time of a call into the system, to prevent an application from executing certain functions
- When starting an activity, to prevent applications from launching activities of other applications
- Both sending and receiving broadcasts, to control who can receive your broadcast or who can send a broadcast to you
- When accessing and operating on a content provider
- Binding to or starting a service

The permission model within Android is possibly one of the most researched areas of the operating system with a variety of interesting findings having been made (see Section 5.1 for Related Work). One of the primary observations is that a developer requests permissions through the ‘AndroidManifest.xml’ file separate to making any permission-requiring API calls in their app's code. You can imagine there being two lists — one of required permissions and the other of requested permissions (in reality no list of required permissions is maintained and there is in fact a third list, that of the granted permissions which may be different again). The fact that these two lists are distinct means that actually the permissions requested (and thereby prompted for the user to check) may be different to those actually required by the app itself [66, 85].

Following on from that thought, there are a number of scenarios that can be imagined whereby the two lists (requested versus required) get out of sync. An obvious starting point is that either list could include permission entries not on the other list. If the additional entries are on the requested list, then the app is asking the user to authorise permissions that it will not require [66, 94]. If we were to assume that the user installs such an app, then it could be argued that no harm is done. After all, the user has approved those permissions for the app (thus allowing it to be installed), if it does not use them (or more specifically isn't coded to use them) what harm could that be. It has however been pointed out that "overdeclaring breaks the principle of least privilege" with "severe consequences should the overprivileged application have an exploitable vulnerability" [66]. The impact of the consequences is obviously highly dependent on both the type of vulnerability exposed and the capability associated with the unrequired yet requested permission. It is possible, that excessive permission requests could result in exploitable vulnerabilities where an app limited to the necessary permissions remains secure. The principle of least privilege is a long standing security concept for good reason [99] and whilst the likelihood of such a combination of vulnerability and permission may seem far fetched, experience would suggest that ignoring this principle is unwise. Whether or not such an event does occur, the developer is running the risk that a user will decide not to install the app because of a requested permission that is actually not required. That said, Shabtai et al feel that "in practice, the user is unlikely to deny installation of an application he or she wants based on such a list" [100]. This view is backed up by the results of an Android Central poll shown in Figure 2.3 where the majority (58.02%) of the 5,950 respondents admitted "I just click right through" [41] when prompted to authorise permissions during installation.



Be honest - do you check app permissions when installing?

I read them religiously. 13.66% (813 votes)

I would read them, but I don't really understand them. 9.41% (560 votes)

I read them -- and probably worry more about them than I should. 18.91% (1,125 votes)

I just click right through. 58.02% (3,452 votes)

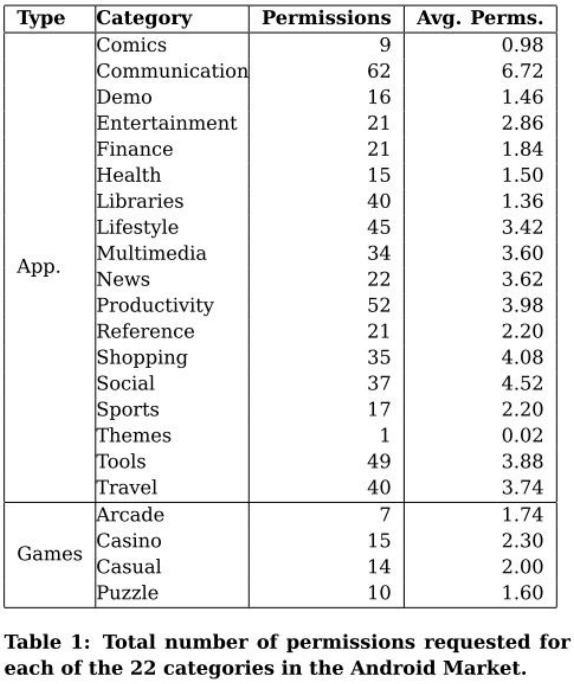
Total Votes: 5,950

Figure 2.3: Be honest - do you check app permissions when installing?

As an alternative the additional entry may be on the required list rather than the requested list. In this case, a security exception will be raised when the app comes to use the API call for which it has not requested the appropriate permission. One would hope that such an issue would be identified during the testing of the app however bugs are common within software development and there is no reason to think that such an issue will not happen here. In fact research done in this area has found that this scenario may actually be quite common. For example, Enck et al found that out of the 1,100 apps they investigated, "246 applications (22.4%) included code to obtain a phone identifier; however, only 210 of these applications have the `READ_PHONE_STATE` permission required to obtain access" and that whilst "505 applications (45.9%) attempt to access location, only 304 (27.6%) have the permission to do so" [78]. It is not clear from their paper as to whether the resultant security exceptions are handled within the app's code or not.

Aside from requesting permissions which are not required and failing to request permissions that are, several other permissions-related issues have been identified. Both duplicate requests, where a single permission is listed multiple times in the manifest, and incorrectly spelt permissions have been seen during permission reviews [67]. Duplicate permissions are simply ignored, whilst incorrectly spelt permissions may be assumed to be third-party permissions depending on the nature of the error. This in turn raises the same concerns around the principle of least privilege as with requesting manifest permissions which are not required, as a misspelt permission could be seen as both an additional unrequired permission and a missing required one.

So far we have considered the total number of permissions available to Android but it is worth understanding that whilst there are 130 permissions available such large numbers are not requested by commonly used apps. There have been several pieces of research which have looked at the average number of permissions in various circumstances. Zhou et al found that "the average number of permissions requested by malicious apps is 11 while the average number requested by benign apps is 4" [109]. Hui Chia et al found similar results within their research, they calculated the mean number of permissions for 650 randomly selected popular (top-selling-free and top-selling-paid) Android apps as 4.5 compared to 3.0 for 1210 new Android apps (first appearing mid June 2011) [70]. The results shown in Figure 2.4 provide a similar picture when considering 1,100 apps taken across each of the categories in the Android Market (now called Google Play) [67]. From all these numbers, an average of three or four permissions seems an acceptable approximation and also an amount which, assuming adequately explained, one would hope should be manageable for consumers to consider.



Type	Category	Permissions	Avg. Perms.
App.	Comics	9	0.98
	Communication	62	6.72
	Demo	16	1.46
	Entertainment	21	2.86
	Finance	21	1.84
	Health	15	1.50
	Libraries	40	1.36
	Lifestyle	45	3.42
	Multimedia	34	3.60
	News	22	3.62
	Productivity	52	3.98
	Reference	21	2.20
	Shopping	35	4.08
	Social	37	4.52
	Sports	17	2.20
	Themes	1	0.02
	Tools	49	3.88
	Travel	40	3.74
Games	Arcade	7	1.74
	Casino	15	2.30
	Casual	14	2.00
	Puzzle	10	1.60

Table 1: Total number of permissions requested for each of the 22 categories in the Android Market.

Figure 2.4: Total number of permissions requested for each of the 22 categories in the Android Market

Whilst the permissions architecture is a critical component of Android's sandboxing it is clear that there are issues related to how it is being utilised within developed apps. Whilst it would be easy to solely blame developers for these mistakes, several researchers have felt that the structure, number and documentation of Android's permissions contribute a significant amount to the problems seen. Wain Yee Au et al analysed the Android documentation associated with platform releases and identified that "there has been an Android release every three months on average with four permissions changing on average (either added, removed or deprecated) with each release" [66]. Porter Felt et al identified that the Android "documentation lists permission requirements for only

78 methods, whereas our testing reveals permission requirements for 1,259 methods (a sixteen-fold improvement over the documentation)" [94]. In a similar way Grace et al "found out that though Android's permission based security model might be comprehensive enough in specifying the permissions required to access sensitive data or features, the available API documentation is incomplete about which APIs a permission grants access to" [85]. It seems likely that part of the reason why developers are struggling to specify permissions correctly is that the documentation required for them to accurately and completely identify their requirements does not exist. This topic is investigated further within Chapter 3.

iOS Comparison There is not much to compare when you consider permissions and iOS. As has been highlighted in the past, "Android presents 124 permissions to users with install-time warnings, whereas iOS obtains user consent for two permissions with runtime confirmation dialogs" [95]. It is expected that the next version of the iOS platform (iOS 6) will require user consent when apps first attempt to access several more pieces of user information. Currently the prompts are related to location information however recent issues with malware have led Apple to look at introducing prompts in the areas of contacts, calendars, reminders and pictures [17].

Based on this comparison, some may consider iOS far behind Android from a permissions based, user choice perspective. In reality, in both Android and iOS the user only has two options — install the app, or do not install the app. Whilst it may be possible to make a more informed decision with regard to Android apps, if Figure 2.3 is anything to go by most consumers aren't taking advantage of that additional information (assuming they have the understanding to be able to).

Some have indicated that a more selection based authorisation process could be beneficial [100, 92, 69], whereby rather than not installing an app because of the permissions it has, a user instead only authorises those permissions they are willing to allow it. Such a situation could allow the user a level of 'testing' the app before becoming satisfied it is trustworthy and enabling those previously disabled permissions. Assuming the average number of permissions per Android app so far seen and the total number of iOS permissions such a proposition may seem eminently manageable from a user perspective. In reality it is not the number of permissions which is the current block — after all, the average of three or four quoted previously is from studies taken over the last three years. It is far more likely that consumers are simply failing to engage with the security warnings themselves either because they see them as a nuisance, slowing their access to new functionality or because the information they hold has little resonance with them [97, 89]. If selective permission authorisation were to be made available, based on current understanding of user habits it would likely benefit very few users — only those with an existing desire to engage in security decisions. For those without that desire, or limited understanding of the permissions and associated capabilities, selective permissions would probably be overwhelming. For this majority, I am not convinced that simply more choice is what the user requires. I believe, it is the issues of usability and risk communication which must be resolved in order that consumers will interact with security messages. This requires a focus on providing useful and engaging decision-making information and is a problem not limited to smartphones.

2.3 App Stores

There are currently two mechanisms by which a user can install a newly developed Android app. The first method involves using the Android app store, called Google Play (previously Android Market) whilst the second involves the direct installation from a third-party source. Google Play offers a number of benefits to both developer and user. For the developer, Play acts as an advertising, purchasing, fulfillment and reporting platform for their apps and allows configuration of the categorisation, branding and cost (unless free) prior to user discovery. For the user, Play provides a central location for media content such as apps, movies and music with the power of Google search to help identify and locate potential content of interest. Where necessary, Google act as the financial payment processor, allowing consumers to only provide their credit card details to Google, whilst in turn purchasing content from a variety of different developers and providers. Google has also established a community within Play, by enabling consumers to rate and comment on apps they have installed as well as by showing installation statistics. Google's intention is to strengthen this community by allowing developers to reply to comments in the future with those replies visible along with all the user comments.

As well as making the initial discovery and purchase of apps an easy process, Google Play provides an app management and update service. This tracks installed, previously installed and purchased apps in order to allow the user to easily uninstall or reinstall apps on the same or a future device. With regards to app update, Google Play identifies when an app update has been uploaded by the developer and notifies the user. It will even automatically download and install the update if the user has configured that behaviour for that specific app.

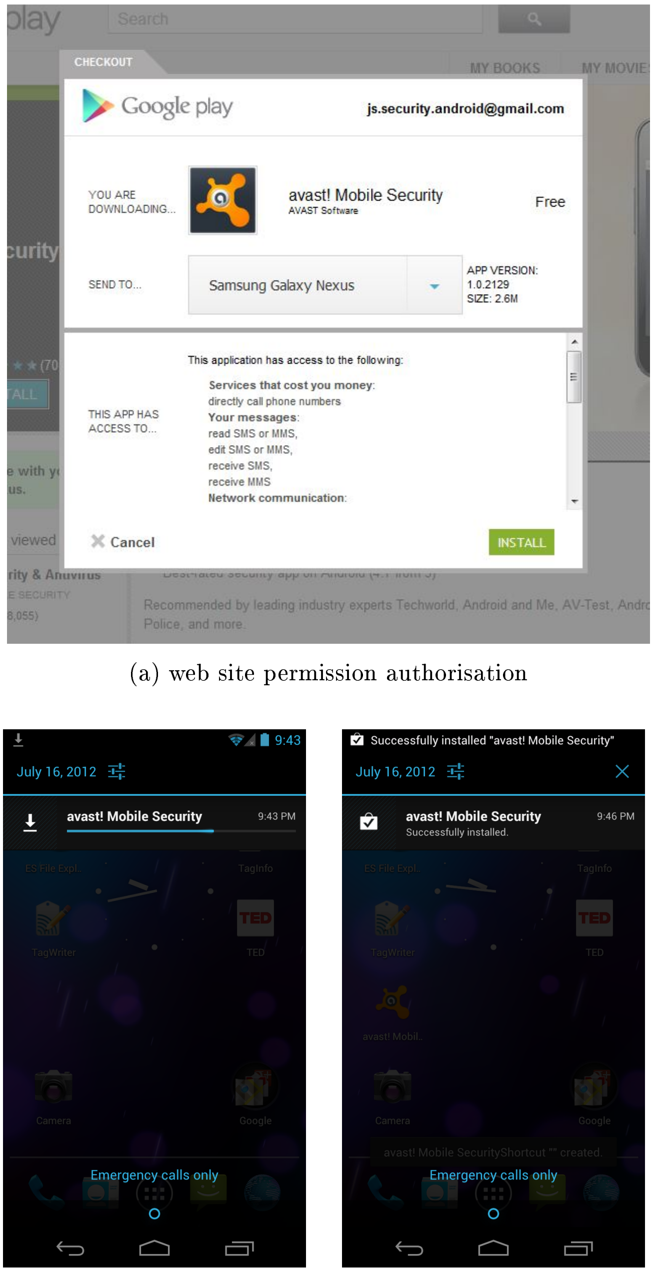
Google Play is made available to consumers via two interfaces — a pre-installed Android app and a web site. The two make available similar functionality with the web site able to send install and uninstall commands to the device following the mandatory authorisation of permissions, as seen in Figure 2.5. The Google Play app does provide some device specific security settings which are not accessible through the web site. These include controls for auto-updating, content filtering and the ability to set (and manage) a PIN used to restrict both the security settings and the ability to make purchases of paid content.

Whilst Google Play has many benefits, the barrier to entry for malicious developers is quite low with the Android Developer site informing developers they can "set up to start publishing on Google Play in only a few minutes" [27]. The process involves creating a developer profile, agreeing to the Developer Distribution Agreement (about a seven page document) and paying a registration fee of $25.00. For a malicious developer who will likely provide false information and probably not read the agreement, the entire process will indeed take minutes. Once signed up, a developer (malicious or otherwise) can upload an app for distribution with minimal effort.

In an attempt to control malware, Play does incorporate two security mechanisms. Firstly an automated system called 'Bouncer' is used to scan apps during the upload process. It does this in an attempt to identify malicious or undesirable activity within apps. This service was first announced in February of 2012 [5] and whilst no clear indication was given as to when it was introduced, the information provided would suggest this was sometime during the first half of 2011. The fact that Bouncer "actually run[s] every application on Google's cloud infrastructure and simulate[s] how it will run on an Android device" [5] has been utilised by two researchers as a means to identify how to subvert the automated scanning system [49]. They wrote test apps which called back to their servers when run and using this information they identified that the app-under-test runs in an emulator, which they then worked out how to subvert.

As a second layer of protection, Google performs post upload, house-keeping operations on Google Play, identifying or reacting to malicious apps being downloaded by consumers, which it then removes from the listings and can even forcibly uninstall from Android devices already 'infected'. Historically, malware has successfully gotten onto Google Play then to be downloaded by tens or hundreds of thousands of consumers. This secondary protection mechanism has then been employed to clear up the mess after the fact. For example, "Google removed more than 100 malicious apps from the Android Market in 2011" [103]. Reports suggest that malware on Android is growing where "a comparison between the number of malicious Android application package files (APKs) received in Q1 2011 and in Q1 2012 reveals a more staggering find - an increase from 139 to 3063 counts" [80]. Whilst there are claims that the number of malware variants is increasing on the Android platform, "most of the newly discovered or existing malwares have been created to reap profit, most commonly by sending premium-rate SMS messages" [79].

As an alternative to Google Play, consumers can install apps directly from any third-party source. In reality this can take any form of file transfer imaginable, whereby the result is that the Android package file (.apk) reaches the device and is then run. This can include downloading apps from an alternative app store (e.g. Amazon Appstore for Android), downloading from a web site, receiving through e-mail or direct copy via USB cable. In order for subsequent install to work, the user must first check a security setting which authorises 'Unknown sources - Allow installation of non-Market apps' as in Figures 2.6b and 2.6c. Without this setting enabled, the installation


CHECKOUT

Google play js.security.android@gmail.com

YOU ARE DOWNLOADING... avast! Mobile Security AVAST Software Free

SEND TO... Samsung Galaxy Nexus

APP VERSION: 1.0.2129
SIZE: 2.6M

THIS APP HAS ACCESS TO...

This application has access to the following:

Services that cost you money:
directly call phone numbers
Your messages:
read SMS or MMS,
edit SMS or MMS,
receive SMS,
receive MMS
Network communication:

Cancel INSTALL

(a) web site permission authorisation

9:43
July 16, 2012
avast! Mobile Security 9:43 PM
Emergency calls only

Successfully installed "avast! Mobile Security"
July 16, 2012
avast! Mobile Security 9:46 PM
Successfully installed.
avast! Mobile SecurityShortcut "" created.
Emergency calls only

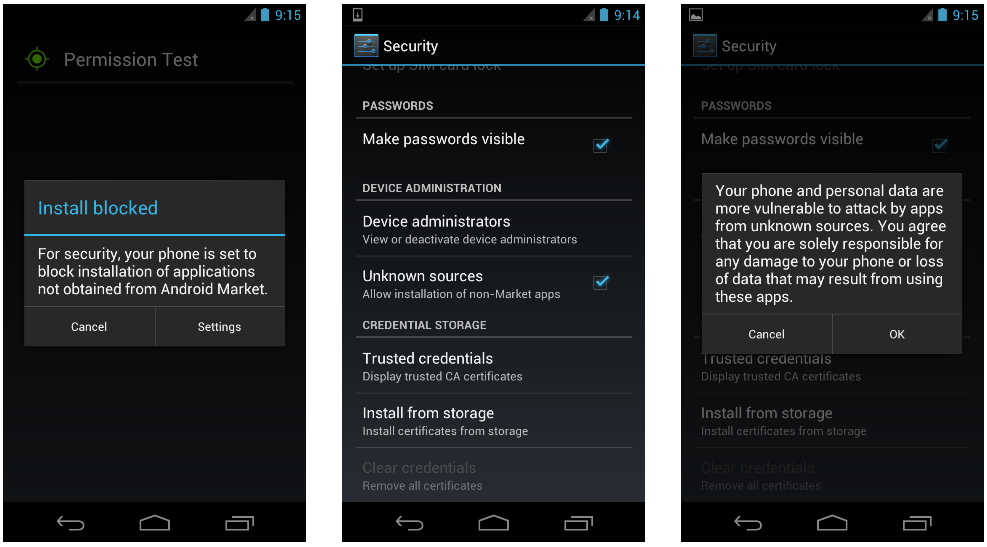
(b) app download (c) app installation complete

Figure 2.5: Google Play web site triggered app installation

will not proceed and a warning will be shown to the user (Figure 2.6a). Whilst these mechanisms provide a means of installing apps from sources other than Google Play, they do still require the user to perform the permission authorisation prior to installation. Once installed, these apps will not benefit from the same update management afforded apps installed through Google Play with the user responsible for app management themselves (although the app itself may assist if coded to do so).

During the investigations documented in Chapter 3 and 4, the test apps were installed on devices and emulators using these third-party sources techniques.

There is one other way of installing apps from a third-party source which also makes use of



Install blocked

For security, your phone is set to block installation of applications not obtained from Android Market.

PASSWORDS

Make passwords visible ☑

DEVICE ADMINISTRATION

Device administrators
View or deactivate device administrators

Unknown sources ☑
Allow installation of non-Market apps

CREDENTIAL STORAGE

Trusted credentials
Display trusted CA certificates

Install from storage
Install certificates from storage

Clear credentials
Remove all certificates

Your phone and personal data are more vulnerable to attack by apps from unknown sources. You agree that you are solely responsible for any damage to your phone or loss of data that may result from using these apps.

(a) unknown sources warning (b) unknown sources option (c) confirmation prompt

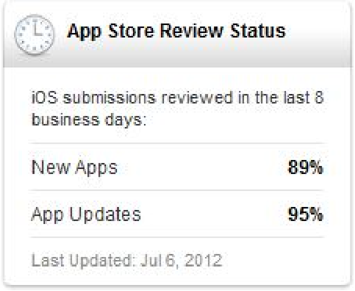
Figure 2.6: Unknown Sources - Allow installation of non-Market apps

a direct transfer via USB. This mechanism requires a further security setting to be enabled — ‘USB debugging - Debug mode when USB is connected’ — and allows for the silent installation of apps using the ‘Android Debug Bridge’ utility. When installed via this method, the user is not prompted to authorise permissions, with the app installing without any user interaction on the device. This is not intended to be a core route for installations by consumers but is instead intended for developers and testers. Whilst that may be the case, the option is available to consumers technically able to employ it or coerced to do so by computer based malware. With the ability for malicious apps to get onto Google Play, at least temporarily, it is currently unlikely that this convoluted and potentially suspicious technique will be employed by malware developers.

iOS Comparison In comparison to Google Play, the Apple App Store can be considered a more restrictive distribution point. The developer requirements for use of the App Store are far more rigorous for a start. In order to be able to distribute apps through the Apple App Store a developer must be part of the Apple Developer Program, costing a minimum of $99 per year [21]. A developer must also agree to the iOS Developer Program License Agreement, a 44 page document and provide basic developer information. Any app submitted to the App Store will then go through the App Store Approval Process before being made available to consumers. This process includes review by the App Review Team who may reject the app if it does not conform to either the ‘App Store Review Guidelines’ or the ‘iOS Human Interface Guidelines’ [11, 36]. In fact there is even an App Store Submission Tips page for developers, designed to help them avoid issues which have previously resulted in apps being blocked from the App Store [12]. If Apple reject an app, the developer is able to submit an appeal via the App Review Board.

It would seem likely that the cost of this process to potentially malicious developers could act as a reasonable deterrent and experience has so far shown that whilst the App Store has not been 100% free of malware, it has experienced far fewer instances than Google Play. This rigorous and taxing process does however impact throughput. Apple maintains a review process status on their approval process page [10] which most recently displayed the information shown in Figure 2.7.

A further difference between Android and iOS is that side-loading of apps on iOS (i.e. not downloading them from the main app store) is far more restricted and convoluted a process than on Android. Some may say that there is no side-loading on iOS, which is not quite true though. It is possible to load direct to an iOS device, but to do so requires membership of the Apple Developer Program (incurring the costs above) and the use of a custom provisioning profile which



App Store Review Status

iOS submissions reviewed in the last 8 business days:

New Apps	89%
App Updates	95%

Last Updated: Jul 6, 2012

Figure 2.7: App Store Review Status

must be signed by Apple and works on a device-by-device basis. In comparison to Android, where the only requirement is to check an option on the device, it is clear that this is not intended for anything other than developer testing of iOS apps. These restrictions over side-loading mean that officially sanctioned third-party app stores for iOS devices do not exist. As with almost everything in the mobile security space, there is a slight exception. Jailbroken devices, ones where an exploit has been used to breakdown the security protections and allow the device to be run as an administrator (known as rooting on Android), do enable this side-loading and commonly make use of an unofficial third-party app store called Cydia [22].

Concluding Remarks The security architecture components detailed in this chapter, along with defensive programming of apps, go a long way to protecting user information on their smartphone device. In the next chapter the permissions architecture of Android will be investigated further, in particular with a view to identifying the user interaction with permission authorisation requests and the history of permissions across recent platform iterations.

Before moving on there is one last security mechanism worth briefly mentioning as it is a protection against direct data access. Whilst smartphones currently rely on single-user operating systems, the fact that many smartphone apps store user credentials means that anyone able to access the device is likely able to then make use of those stored credentials to access specific services and information stores. In order to maintain confidentiality of information there is therefore a clear need for strong access control mechanisms to be employed on smartphone devices. As far as advice for smartphone security goes, there is a consistent theme from all manner of sources highlighting the need to employ the devices password/PIN/alternative locking mechanism [24, 47, 39, 55]. Just using a locking mechanism doesn't immediately mean consumers are 'safe'. The strength of the mechanism and the associated authentication factor (secret password for example) are two obvious factors which affect the level of security provided and on more than one occasion the locking mechanism has been found to be a source of vulnerability in a smartphone OS [34, 35]. Figure 2.8 shows the results of a question included in a study of 520 people by Sophos. A third of respondents have "no passwords on either phone" [103].

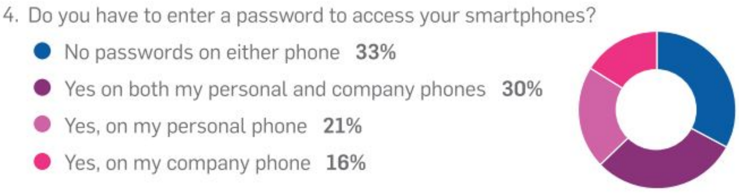


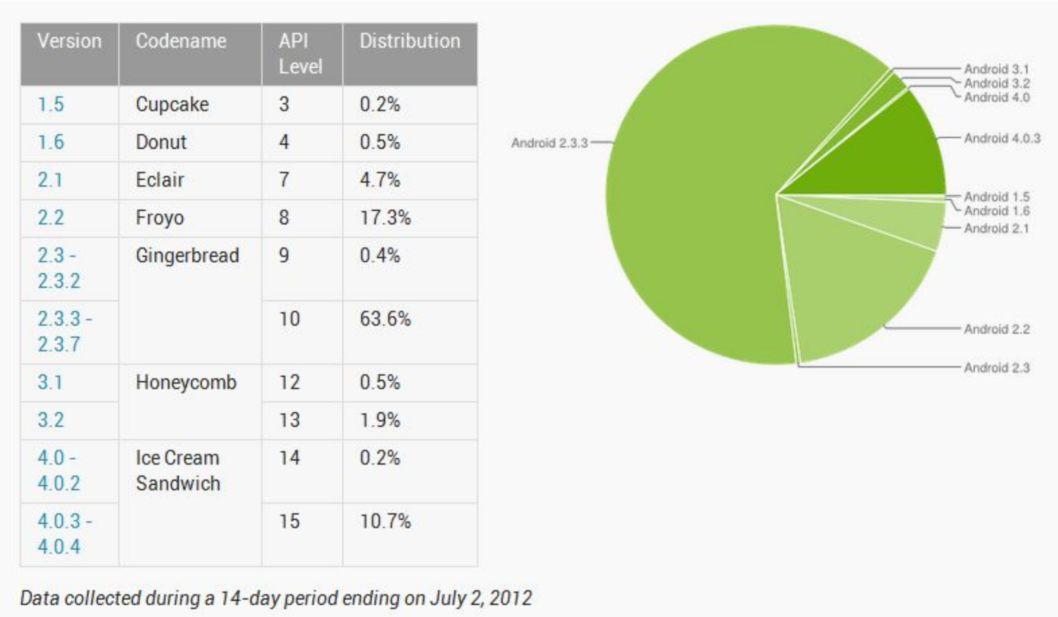
Figure 2.8: Do you have to enter a password to access your smartphones?

A recent study by Symantec highlighted the extent to which lost devices are accessed. The Smartphone Honey Stick Project identified that of 50 smartphones 'lost' with fake information and monitoring apps, 96% were accessed by the finder [104]; 89% of devices had personal related apps and information accessed whilst 83% had corporate related apps and information accessed; 70% had both types of information accessed and even after all this activity, 50% of the finders contacted the owner and provided contact information in order to return the lost device. Whilst some of this activity could be construed as trying to identify the owner, the specific monitoring capability highlighted that some activity was far more penetrating — attempts to access an online banking app were made in 43% of cases and a 'saved passwords' file was accessed on 57% of devices.

Chapter 3

Android Permissions Architecture

As has already been identified, and as further documented in Section 5.1, there has been a range of research performed into the permissions architecture of Android. A common complaint is that the Android documentation is limited and that developers and consumers suffer from this deficiency, exacerbated by the rate of change of the permissions. As a follow-on to the existing research, I have chosen to investigate the permissions from two further considerations. Firstly, I look to understand the categorisation of the current 130 permissions [42] both in terms of logical groupings and also in terms of protection level. Secondly, I look to understand how the permissions and their categorisations have changed across versions of the Android platform from 2.2 (Froyo) to 4.1 (Jelly Bean). Based on Google's own figures from July 2012, this covers almost 95% of Android devices (Figure 3.1).



Version	Codename	API Level	Distribution
1.5	Cupcake	3	0.2%
1.6	Donut	4	0.5%
2.1	Eclair	7	4.7%
2.2	Froyo	8	17.3%
2.3 - 2.3.2	Gingerbread	9	0.4%
2.3.3 - 2.3.7		10	63.6%
3.1	Honeycomb	12	0.5%
3.2		13	1.9%
4.0 - 4.0.2	Ice Cream Sandwich	14	0.2%
4.0.3 - 4.0.4		15	10.7%

Data collected during a 14-day period ending on July 2, 2012

Figure 3.1: Android platform versions (as at 02/07/2012)

3.1 Permission Categorisation Investigation

3.1.1 Preparation

In order to perform this investigation I created *Permission Test*, an Android app which requests every one of the currently documented 130 manifest permissions (as illustrated in Figure 3.2). During the installation of the app the user is prompted to authorise these permissions as discussed in Section 2.2. It is these pre-installation permission authorisation prompts, and the subsequent granted permissions once the app is installed, that hold the most easily accessible information regarding permission categorisation and protection level. Once installed, the test app itself does not make use of the permissions requested and instead solely displays a 'Hello World' message on the screen. This simplification was necessary in order to avoid the need to code the test app to make use of every one of those 130 permissions. Something which would obviously be extremely time consuming and result in an app of considerable size and complexity and thus with significant potential for errors.

The *Permission Test* app actually requested the `'INTERNET'` permission twice within its manifest. This is because the permission was used as a control, to ensure the correct permission request syntax was being employed and thus the correct information displayed. The choice of the `'INTERNET'` permission came from the fact that it is both a well-known and often-used `'dangerous'` permission and thereby a clear indicator for successful permission requests. Duplicate requests for a permission are ignored by the Android platform and so this control request did not affect any observations made during the analysis.

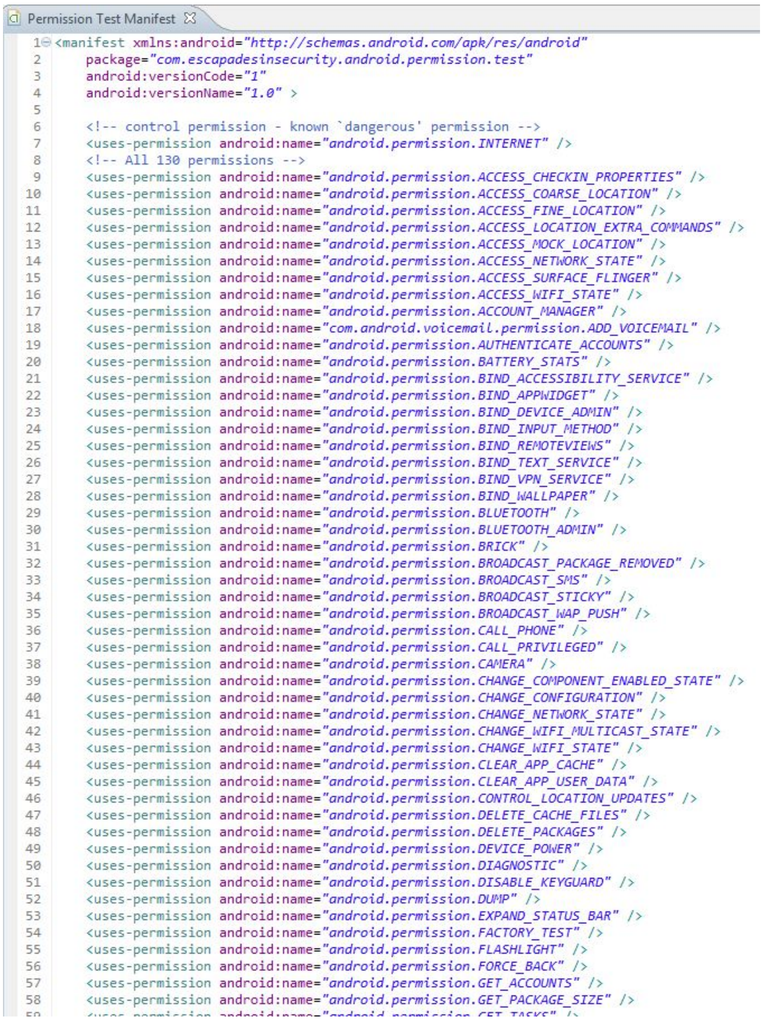
3.1.2 Testing

I first installed *Permission Test* on a Galaxy Nexus device (running Ice Cream Sandwich, specifically Android version 4.0.4). Once the app was installed, the app's app info screen was viewed and the screenshots from Figure 3.3 taken so as to document the granted permissions' labels, protection levels and groupings.

As can be seen from Figure 3.3 and as previously illustrated with Figure 2.2, the permissions are listed in two sections — the `'dangerous'` permissions first, followed by the `'normal'` permissions second (hidden by default). Within each section, the permissions are organised into logical groupings based on the type of feature they relate to.

Using the Android documentation, information from internal Android system files and the testing of singular permissions where necessary, the permission labels from Figure 3.3 were linked back to specific manifest permissions amongst the 130 requested. Of particular use was the Android system file `'/data/system/packages.xml'` which contains a listing of permissions defined within the device as well as information related to currently installed apps. Using a combination of this information, the protection levels and groupings were identified for each permission. These results are shown in the left hand table of Figure 3.4 with each source permission cross-referenced against a logical group through the use of a letter 'D' or 'N' depending on whether the permission's protection level was found to be `'dangerous'` or `'normal'` respectively. This left hand table contains 79 permissions which align with the 78 labels from Figure 3.3. The two lists are not exactly the same length due to the fact that both the `'KILL_BACKGROUND_PROCESS'` permission and the `'RESTART_PACKAGES'` permission utilise a single label: 'kill background processes'.

Whilst 130 permissions were requested, those listed in the right hand table of Figure 3.4 were not seen through the labels shown in the app's app info screen. This suggested that these permissions were not granted to *Permission Test* during installation. This fact was further confirmed by checking the 130 requested permissions against those permissions documented as granted to the app within its section of the `'/data/system/packages.xml'` file on the Nexus device. The 79 permissions listed in the `'/data/system/packages.xml'` file were identical to those in the left table of Figure 3.4, confirming that of the 130 permissions requested, 79 permissions had been granted. It is worth noting that access to the `'/data/system/packages.xml'` file requires elevated privileges and so during this portion of testing the Galaxy Nexus device was 'rooted'



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.escapadesinsecurity.android.permission.test"
    android:versionCode="1"
    android:versionName="1.0" >

    <!-- control permission - known `dangerous' permission -->
    <uses-permission android:name="android.permission.INTERNET" />
    <!-- All 130 permissions -->
    <uses-permission android:name="android.permission.ACCESS_CHECKIN_PROPERTIES" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
    <uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_SURFACE_FLINGER" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.ACCOUNT_MANAGER" />
    <uses-permission android:name="com.android.voicemail.permission.ADD_VOICEMAIL" />
    <uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS" />
    <uses-permission android:name="android.permission.BATTERY_STATS" />
    <uses-permission android:name="android.permission.BIND_ACCESSIBILITY_SERVICE" />
    <uses-permission android:name="android.permission.BIND_APPWIDGET" />
    <uses-permission android:name="android.permission.BIND_DEVICE_ADMIN" />
    <uses-permission android:name="android.permission.BIND_INPUT_METHOD" />
    <uses-permission android:name="android.permission.BIND_REMOTEVIEWS" />
    <uses-permission android:name="android.permission.BIND_TEXT_SERVICE" />
    <uses-permission android:name="android.permission.BIND_VPN_SERVICE" />
    <uses-permission android:name="android.permission.BIND_WALLPAPER" />
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.BRICK" />
    <uses-permission android:name="android.permission.BROADCAST_PACKAGE_REMOVED" />
    <uses-permission android:name="android.permission.BROADCAST_SMS" />
    <uses-permission android:name="android.permission.BROADCAST_STICKY" />
    <uses-permission android:name="android.permission.BROADCAST_WAP_PUSH" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.CALL_PRIVILEGED" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.CHANGE_COMPONENT_ENABLED_STATE" />
    <uses-permission android:name="android.permission.CHANGE_CONFIGURATION" />
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.CLEAR_APP_CACHE" />
    <uses-permission android:name="android.permission.CLEAR_APP_USER_DATA" />
    <uses-permission android:name="android.permission.CONTROL_LOCATION_UPDATES" />
    <uses-permission android:name="android.permission.DELETE_CACHE_FILES" />
    <uses-permission android:name="android.permission.DELETE_PACKAGES" />
    <uses-permission android:name="android.permission.DEVICE_POWER" />
    <uses-permission android:name="android.permission.DIAGNOSTIC" />
    <uses-permission android:name="android.permission.DISABLE_KEYGUARD" />
    <uses-permission android:name="android.permission.DUMP" />
    <uses-permission android:name="android.permission.EXPAND_STATUS_BAR" />
    <uses-permission android:name="android.permission.FACTORY_TEST" />
    <uses-permission android:name="android.permission.FLASHLIGHT" />
    <uses-permission android:name="android.permission.FORCE_BACK" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.GET_PACKAGE_SIZE" />
```

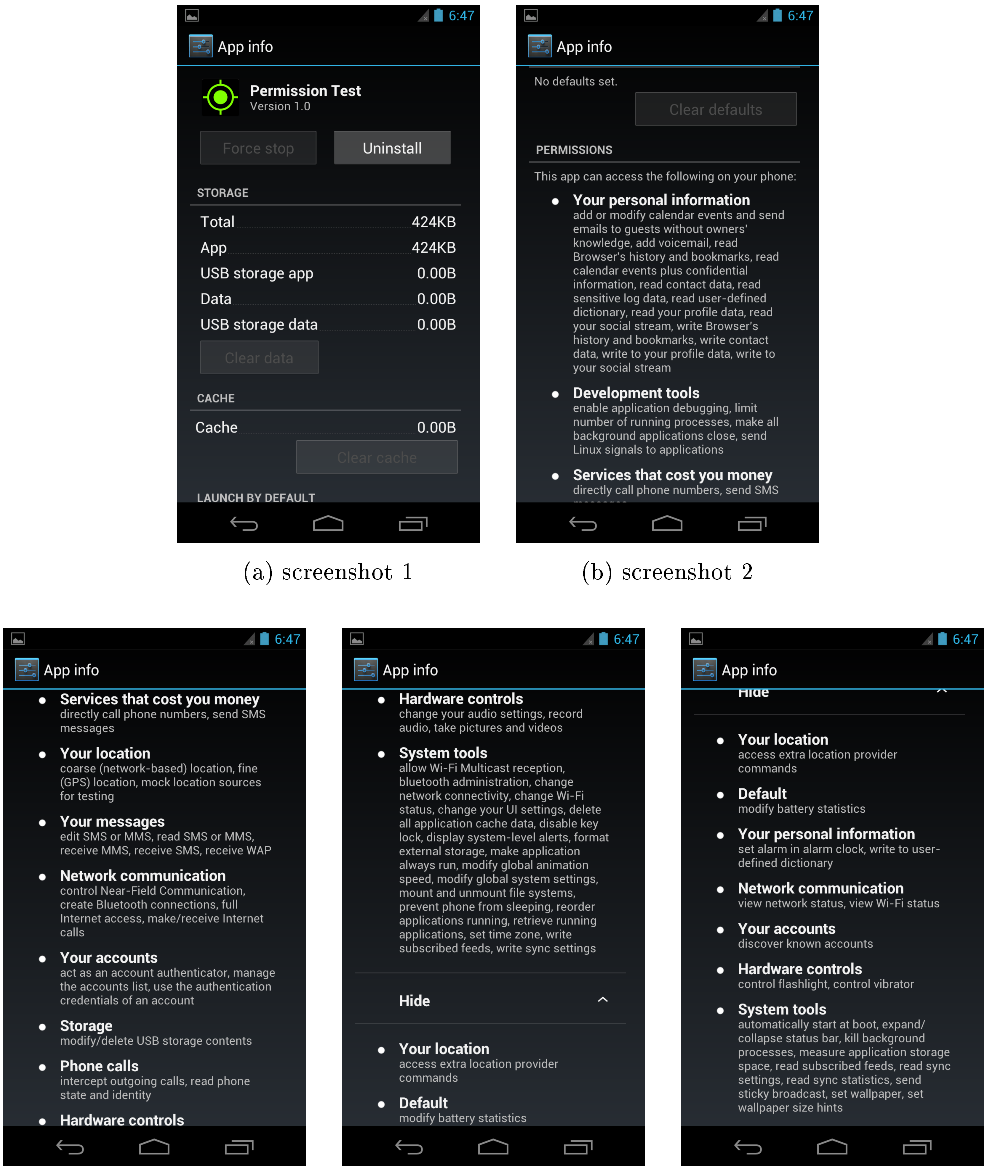
Figure 3.2: *Permission Test* '`AndroidManifest.xml`' (lines 1 to 58)

to enable data extraction. In order to ensure zero impact was experienced in the investigation's results, the 'rooting' of the device was performed manually with the minimum steps required to gain elevated privileges. The device operating system image was left untouched so as to ensure accurate and repeatable results.

In order to determine information about the 51 permissions not granted to *Permission Test*, the device's logging system was analysed to identify any Dalvik VM or Package Manager entries associated with the app or its permission requests. The Package Manager is the Android service responsible for installation and app management. This analysis identified a set of log entries, some of which can be seen in Figure 3.5, tagged 'PackageManager' and indicating one of three warning types.

- 'Unknown permission [...] in package com.escapadesinsecurity.android.permission.test'
- 'Not granting permission [...] to package com.escapadesinsecurity.android.permission.test (protectionLevel=2 flags=0x8be46)'
- 'Not granting permission [...] to package com.escapadesinsecurity.android.permission.test (protectionLevel=3 flags=0x8be46)'

The first warning type was seen four times, the second was seen twenty four times and the third



(a) screenshot 1

6:47

App info

Permission Test
Version 1.0

Force stop | Uninstall

STORAGE

Total	424KB
App	424KB
USB storage app	0.00B
Data	0.00B
USB storage data	0.00B

Clear data

CACHE

Cache	0.00B

Clear cache

LAUNCH BY DEFAULT

(b) screenshot 2

6:47

App info

No defaults set.

Clear defaults

PERMISSIONS

This app can access the following on your phone:

- **Your personal information**
  add or modify calendar events and send emails to guests without owners' knowledge, add voicemail, read Browser's history and bookmarks, read calendar events plus confidential information, read contact data, read sensitive log data, read user-defined dictionary, read your profile data, read your social stream, write Browser's history and bookmarks, write contact data, write to your profile data, write to your social stream
- **Development tools**
  enable application debugging, limit number of running processes, make all background applications close, send Linux signals to applications
- **Services that cost you money**
  directly call phone numbers, send SMS

(c) screenshot 3

6:47

App info

- **Services that cost you money**
  directly call phone numbers, send SMS messages
- **Your location**
  coarse (network-based) location, fine (GPS) location, mock location sources for testing
- **Your messages**
  edit SMS or MMS, read SMS or MMS, receive MMS, receive SMS, receive WAP
- **Network communication**
  control Near-Field Communication, create Bluetooth connections, full Internet access, make/receive Internet calls
- **Your accounts**
  act as an account authenticator, manage the accounts list, use the authentication credentials of an account
- **Storage**
  modify/delete USB storage contents
- **Phone calls**
  intercept outgoing calls, read phone state and identity
- **Hardware controls**

(d) screenshot 4

6:47

App info

- **Hardware controls**
  change your audio settings, record audio, take pictures and videos
- **System tools**
  allow Wi-Fi Multicast reception, bluetooth administration, change network connectivity, change Wi-Fi status, change your UI settings, delete all application cache data, disable key lock, display system-level alerts, format external storage, make application always run, modify global animation speed, modify global system settings, mount and unmount file systems, prevent phone from sleeping, reorder applications running, retrieve running applications, set time zone, write subscribed feeds, write sync settings

Hide ^

- **Your location**
  access extra location provider commands
- **Default**
  modify battery statistics

(e) screenshot 5

6:47

App info

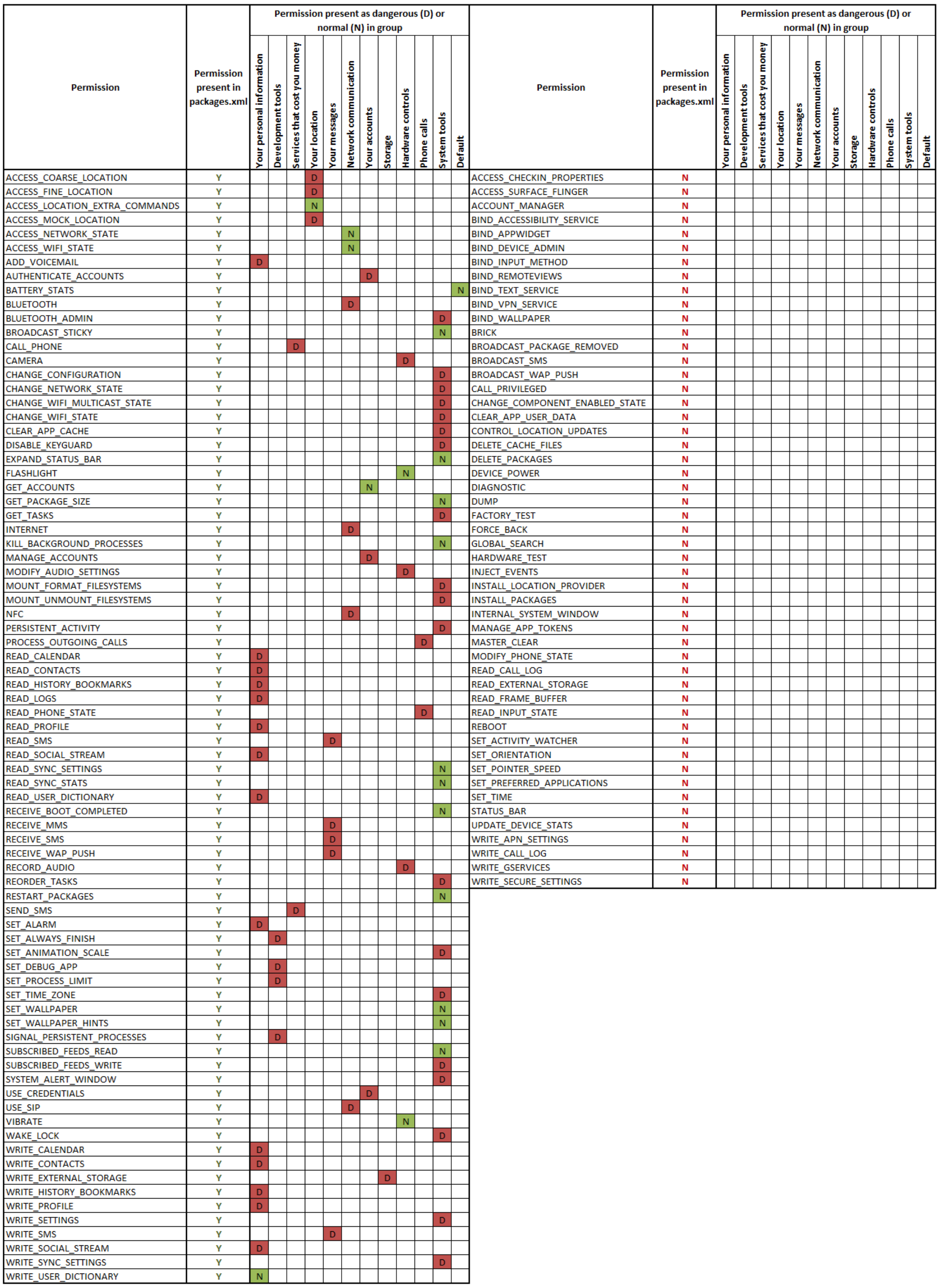
Hide ^

- **Your location**
  access extra location provider commands
- **Default**
  modify battery statistics
- **Your personal information**
  set alarm in alarm clock, write to user-defined dictionary
- **Network communication**
  view network status, view Wi-Fi status
- **Your accounts**
  discover known accounts
- **Hardware controls**
  control flashlight, control vibrator
- **System tools**
  automatically start at boot, expand/collapse status bar, kill background processes, measure application storage space, read subscribed feeds, read sync settings, read sync statistics, send sticky broadcast, set wallpaper, set wallpaper size hints

Figure 3.3: *Permission Test* app information (Galaxy Nexus - Ice Cream Sandwich) - dispersed over multiple screenshots due to vertical scrolling

type was seen twenty three times. These three warning types covered all of the 51 permissions not granted to the *Permission Test* app. A comparison with the Android documentation [42, 48] identified that the four permissions associated with the 'Unknown permission' warning were permissions newly introduced in API16 (Jelly Bean). Protection Level 2 is identified as the '`signature`' protection level whilst Protection Level 3 is the '`signatureOrSystem`' protection level. Whilst this information was insufficient to assign the remaining 51 permissions into their groupings, it had allowed the identification of the associated protection levels of 47 permissions. The classification of these permissions' protection levels also explained why they had not been granted to the *Permission Test* app — it was not signed by the necessary key to meet the requirements for protection levels 2 and 3 permissions.

It is worth mentioning at this point that whilst four permissions were attributed the 'Unknown

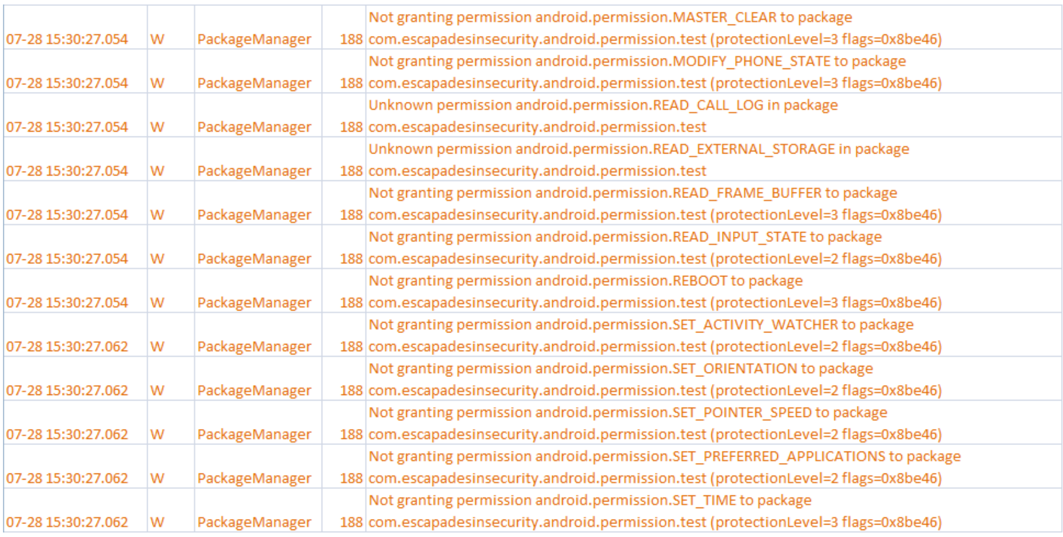


Permission	Permission present in packages.xml	Permission present as dangerous (D) or normal (N) in group											
		Your personal information	Development tools	Services that cost you money	Your location	Your messages	Network communication	Your accounts	Storage	Hardware controls	Phone calls	System tools	Default
ACCESS_COARSE_LOCATION	Y				D								
ACCESS_FINE_LOCATION	Y				D								
ACCESS_LOCATION_EXTRA_COMMANDS	Y				N								
ACCESS_MOCK_LOCATION	Y				D								
ACCESS_NETWORK_STATE	Y						N						
ACCESS_WIFI_STATE	Y						N						
ADD_VOICEMAIL	Y	D											
AUTHENTICATE_ACCOUNTS	Y							D					
BATTERY_STATS	Y												N
BLUETOOTH	Y						D						
BLUETOOTH_ADMIN	Y											D	
BROADCAST_STICKY	Y											N	
CALL_PHONE	Y			D									
CAMERA	Y									D			
CHANGE_CONFIGURATION	Y											D	
CHANGE_NETWORK_STATE	Y											D	
CHANGE_WIFI_MULTICAST_STATE	Y											D	
CHANGE_WIFI_STATE	Y											D	
CLEAR_APP_CACHE	Y											D	
DISABLE_KEYGUARD	Y											D	
EXPAND_STATUS_BAR	Y											N	
FLASHLIGHT	Y									N			
GET_ACCOUNTS	Y							N					
GET_PACKAGE_SIZE	Y											N	
GET_TASKS	Y											D	
INTERNET	Y						D						
KILL_BACKGROUND_PROCESSES	Y											N	
MANAGE_ACCOUNTS	Y							D					
MODIFY_AUDIO_SETTINGS	Y									D			
MOUNT_FORMAT_FILESYSTEMS	Y											D	
MOUNT_UNMOUNT_FILESYSTEMS	Y											D	
NFC	Y						D						
PERSISTENT_ACTIVITY	Y											D	
PROCESS_OUTGOING_CALLS	Y										D		
READ_CALENDAR	Y	D											
READ_CONTACTS	Y	D											
READ_HISTORY_BOOKMARKS	Y	D											
READ_LOGS	Y	D											
READ_PHONE_STATE	Y										D		
READ_PROFILE	Y	D											
READ_SMS	Y					D							
READ_SOCIAL_STREAM	Y	D											
READ_SYNC_SETTINGS	Y											N	
READ_SYNC_STATS	Y											N	
READ_USER_DICTIONARY	Y	D											
RECEIVE_BOOT_COMPLETED	Y											N	
RECEIVE_MMS	Y					D							
RECEIVE_SMS	Y					D							
RECEIVE_WAP_PUSH	Y					D							
RECORD_AUDIO	Y									D			
REORDER_TASKS	Y											D	
RESTART_PACKAGES	Y											N	
SEND_SMS	Y			D									
SET_ALARM	Y	D											
SET_ALWAYS_FINISH	Y		D										
SET_ANIMATION_SCALE	Y											D	
SET_DEBUG_APP	Y		D										
SET_PROCESS_LIMIT	Y		D										
SET_TIME_ZONE	Y											D	
SET_WALLPAPER	Y											N	
SET_WALLPAPER_HINTS	Y											N	
SIGNAL_PERSISTENT_PROCESSES	Y		D										
SUBSCRIBED_FEEDS_READ	Y											N	
SUBSCRIBED_FEEDS_WRITE	Y											D	
SYSTEM_ALERT_WINDOW	Y											D	
USE_CREDENTIALS	Y							D					
USE_SIP	Y						D						
VIBRATE	Y									N			
WAKE_LOCK	Y											D	
WRITE_CALENDAR	Y	D											
WRITE_CONTACTS	Y	D											
WRITE_EXTERNAL_STORAGE	Y								D				
WRITE_HISTORY_BOOKMARKS	Y	D											
WRITE_PROFILE	Y	D											
WRITE_SETTINGS	Y											D	
WRITE_SMS	Y					D							
WRITE_SOCIAL_STREAM	Y	D											
WRITE_SYNC_SETTINGS	Y											D	
WRITE_USER_DICTIONARY	Y	N											

Permission	Permission present in packages.xml	Permission present as dangerous (D) or normal (N) in group											
		Your personal information	Development tools	Services that cost you money	Your location	Your messages	Network communication	Your accounts	Storage	Hardware controls	Phone calls	System tools	Default
ACCESS_CHECKIN_PROPERTIES	N												
ACCESS_SURFACE_FLINGER	N												
ACCOUNT_MANAGER	N												
BIND_ACCESSIBILITY_SERVICE	N												
BIND_APPWIDGET	N												
BIND_DEVICE_ADMIN	N												
BIND_INPUT_METHOD	N												
BIND_REMOTEVIEWS	N												
BIND_TEXT_SERVICE	N												
BIND_VPN_SERVICE	N												
BIND_WALLPAPER	N												
BRICK	N												
BROADCAST_PACKAGE_REMOVED	N												
BROADCAST_SMS	N												
BROADCAST_WAP_PUSH	N												
CALL_PRIVILEGED	N												
CHANGE_COMPONENT_ENABLED_STATE	N												
CLEAR_APP_USER_DATA	N												
CONTROL_LOCATION_UPDATES	N												
DELETE_CACHE_FILES	N												
DELETE_PACKAGES	N												
DEVICE_POWER	N												
DIAGNOSTIC	N												
DUMP	N												
FACTORY_TEST	N												
FORCE_BACK	N												
GLOBAL_SEARCH	N												
HARDWARE_TEST	N												
INJECT_EVENTS	N												
INSTALL_LOCATION_PROVIDER	N												
INSTALL_PACKAGES	N												
INTERNAL_SYSTEM_WINDOW	N												
MANAGE_APP_TOKENS	N												
MASTER_CLEAR	N												
MODIFY_PHONE_STATE	N												
READ_CALL_LOG	N												
READ_EXTERNAL_STORAGE	N												
READ_FRAME_BUFFER	N												
READ_INPUT_STATE	N												
REBOOT	N												
SET_ACTIVITY_WATCHER	N												
SET_ORIENTATION	N												
SET_POINTER_SPEED	N												
SET_PREFERRED_APPLICATIONS	N												
SET_TIME	N												
STATUS_BAR	N												
UPDATE_DEVICE_STATS	N												
WRITE_APN_SETTINGS	N												
WRITE_CALL_LOG	N												
WRITE_GSERVICES	N												
WRITE_SECURE_SETTINGS	N												

Figure 3.4: Ice Cream Sandwich permissions breakdown (Galaxy Nexus - Ice Cream Sandwich)

permission' warning and not granted to the app, the Android documentation [42, 9] actually identifies six permissions as new to API16 (Jelly Bean). This will be revisited in Section 3.2 but at this time it is not clear whether there is an error in the Android documentation.



```
07-28 15:30:27.054  W  PackageManager  188  Not granting permission android.permission.MASTER_CLEAR to package
                                            com.escapadesinsecurity.android.permission.test (protectionLevel=3 flags=0x8be46)
07-28 15:30:27.054  W  PackageManager  188  Not granting permission android.permission.MODIFY_PHONE_STATE to package
                                            com.escapadesinsecurity.android.permission.test (protectionLevel=3 flags=0x8be46)
07-28 15:30:27.054  W  PackageManager  188  Unknown permission android.permission.READ_CALL_LOG in package
                                            com.escapadesinsecurity.android.permission.test
07-28 15:30:27.054  W  PackageManager  188  Unknown permission android.permission.READ_EXTERNAL_STORAGE in package
                                            com.escapadesinsecurity.android.permission.test
07-28 15:30:27.054  W  PackageManager  188  Not granting permission android.permission.READ_FRAME_BUFFER to package
                                            com.escapadesinsecurity.android.permission.test (protectionLevel=3 flags=0x8be46)
07-28 15:30:27.054  W  PackageManager  188  Not granting permission android.permission.READ_INPUT_STATE to package
                                            com.escapadesinsecurity.android.permission.test (protectionLevel=2 flags=0x8be46)
07-28 15:30:27.054  W  PackageManager  188  Not granting permission android.permission.REBOOT to package
                                            com.escapadesinsecurity.android.permission.test (protectionLevel=3 flags=0x8be46)
07-28 15:30:27.062  W  PackageManager  188  Not granting permission android.permission.SET_ACTIVITY_WATCHER to package
                                            com.escapadesinsecurity.android.permission.test (protectionLevel=2 flags=0x8be46)
07-28 15:30:27.062  W  PackageManager  188  Not granting permission android.permission.SET_ORIENTATION to package
                                            com.escapadesinsecurity.android.permission.test (protectionLevel=2 flags=0x8be46)
07-28 15:30:27.062  W  PackageManager  188  Not granting permission android.permission.SET_POINTER_SPEED to package
                                            com.escapadesinsecurity.android.permission.test (protectionLevel=2 flags=0x8be46)
07-28 15:30:27.062  W  PackageManager  188  Not granting permission android.permission.SET_PREFERRED_APPLICATIONS to package
                                            com.escapadesinsecurity.android.permission.test (protectionLevel=2 flags=0x8be46)
07-28 15:30:27.062  W  PackageManager  188  Not granting permission android.permission.SET_TIME to package
                                            com.escapadesinsecurity.android.permission.test (protectionLevel=3 flags=0x8be46)
```

Figure 3.5: Package Manager log entries post installation of *Permission Test* (Galaxy Nexus - Ice Cream Sandwich)

3.1.3 Results

From this investigation it has been possible to identify that of the 130 permissions currently documented, 79 of these are accessible to developer's apps running on the Ice Cream Sandwich (API15) version of the platform and 51 are restricted, either through being introduced in Jelly Bean (API16) or requiring that the app come from a source able to sign an app with the same key as used by the device producer. Of the 79 permissions available to developers, 59 of these are marked with a protection level of `'dangerous'` whilst 20 are marked as `'normal'`. The permissions are split amongst a set of logical groups as per Table 3.1.

Group	`'dangerous'` **permissions**	`'normal'` **permissions**	**Total permissions**
Your personal information	13	2	15
Development tools	4	0	4
Services that cost you money	2	0	2
Your location	3	1	4
Your messages	5	0	5
Network communication	4	2	6
Your accounts	3	1	4
Storage	1	0	1
Hardware controls	3	2	5
Phone calls	2	0	2
System tools	19	11	30
Default	0	1	1
-	59	20	79

Table 3.1: Ice Cream Sandwich permission groupings

In order to determine the protection levels and groupings that apply to the four 'Unknown permissions' on Ice Cream Sandwich, the *Permission Test* app was installed onto an emulator running Jelly Bean. This identified that one permission was actually assigned protection level 2 and hence was not then granted, but was logged. Two of the other three were identified as `'dangerous'` permissions in the 'Your personal information' group, whilst the third was a `'normal'` permission within the 'System tools' group.

The results for these four 'Unknown permissions' when installed on Jelly Bean suggested that permission categorisation may not be as straight forward as one might think across Android platform versions. This realisation led to the second stage of investigation, documented in the next section.

3.2 Permission Evolution Investigation

3.2.1 Testing

Having completed the initial Permission Categorisation investigation, the *Permission Test* app was installed, in turn, onto each of the following Android platform versions running on emulators. The emulators were created and managed using the Android Virtual Device (AVD) Manager which comes as part of the Android SDK.

- API8 - Android 2.2 - Froyo
- API10 - Android 2.3.3 - Gingerbread
- API13 - Android 3.2 - Honeycomb
- API14 - Android 4.0 - Ice Cream Sandwich
- API15 - Android 4.0.3 - Ice Cream Sandwich
- API16 - Android 4.1 - Jelly Bean

Having installed *Permission Test* on a particular Android platform version, the same information was collected as in Subsection 3.1.2. Additionally, the specific permission labels were noted, so that these could be compared across platform versions along with the protection levels and groupings. Where permissions were not granted to *Permission Test*, the system logs were once again consulted to identify the protection level associated with those permissions. As before, the grouping information and associated permission label were unavailable in the log entries for these ungranted permissions.

During the analysis of the system logs associated with the Jelly Bean version of Android, it became clear that changes had been made to the assignment of protection levels in this platform version. Whilst previously there had been four possible protection levels (as discussed in Section 2.2), this had now been expanded with the introduction of two flag values which are documented in the most recent Android documentation [48]. The initial four protection levels were `'normal'`, `'dangerous'`, `'signature'` and `'signatureOrSystem'`, and were assigned numerical values within the platform of 0, 1, 2 and 3 respectively. The two new flags have been assigned hex values of `'0x10'` for `'system'` and `'0x20'` for `'development'`. These flag values expand the number of possible protection levels considerably, although only two new protection levels were seen within the analysis. These levels were represented by the decimal values 18 and 50. The decimal value 18 (hex value `'0x12'`), indicates protection level 2 with the `'0x10'` flag set. This is interpreted as `'signature'` or `'system'`. It is not clear how or if this protection level (18) is different from the existing `'signatureOrSystem'` protection level (3), although the results shown in Table 3.2 suggest that it is a reasonably direct replacement. The decimal value 50 (hex value `'0x32'`), indicates protection level 2 with both the `'0x10'` and `'0x20'` flags set. This is interpreted as `'signature'` or `'system'` or `'development'` and represents a totally new protection level combination.

3.2.2 Summary Results

Table 3.2 shows a summary of how the current 130 permissions are distributed amongst the protection levels across the Android platform versions, with 'Unknown permissions' listing those permissions which were not available at that time. It also identifies at which version those previously 'Unknown permissions' were introduced and how many permission labels were revised with each new platform version. Considerable changes have been made with the introduction of Jelly

Bean (API16), with 47 permission labels being changed, four new permissions added and, as already highlighted, a change to the protection level assignments following the introduction of two new protection level flags. The most significant permissions change prior to that was with the initial introduction of Ice Cream Sandwich (API14) when seven new permissions were added.

	Permission Count					
Number of ...	**API8**	**API10**	**API13**	**API14**	**API15**	**API16**
‘`normal`’ (0) permissions	20	20	20	20	20	21
‘`dangerous`’ (1) permissions	54	55	55	59	59	55
‘`signature`’ (2) permissions	24	22	23	24	24	25
‘`signatureOrSystem`’ (3) permissions	16	20	21	23	23	0
‘`0x12`’ (18) permissions	0	0	0	0	0	21
‘`0x32`’ (50) permissions	0	0	0	0	0	8
‘Unknown permissions’	16	13	11	4	4	0
New permissions	-	3	2	7	0	4
Revised permission labels	-	2	0	2	0	47

Table 3.2: Android permissions breakdown across platform versions

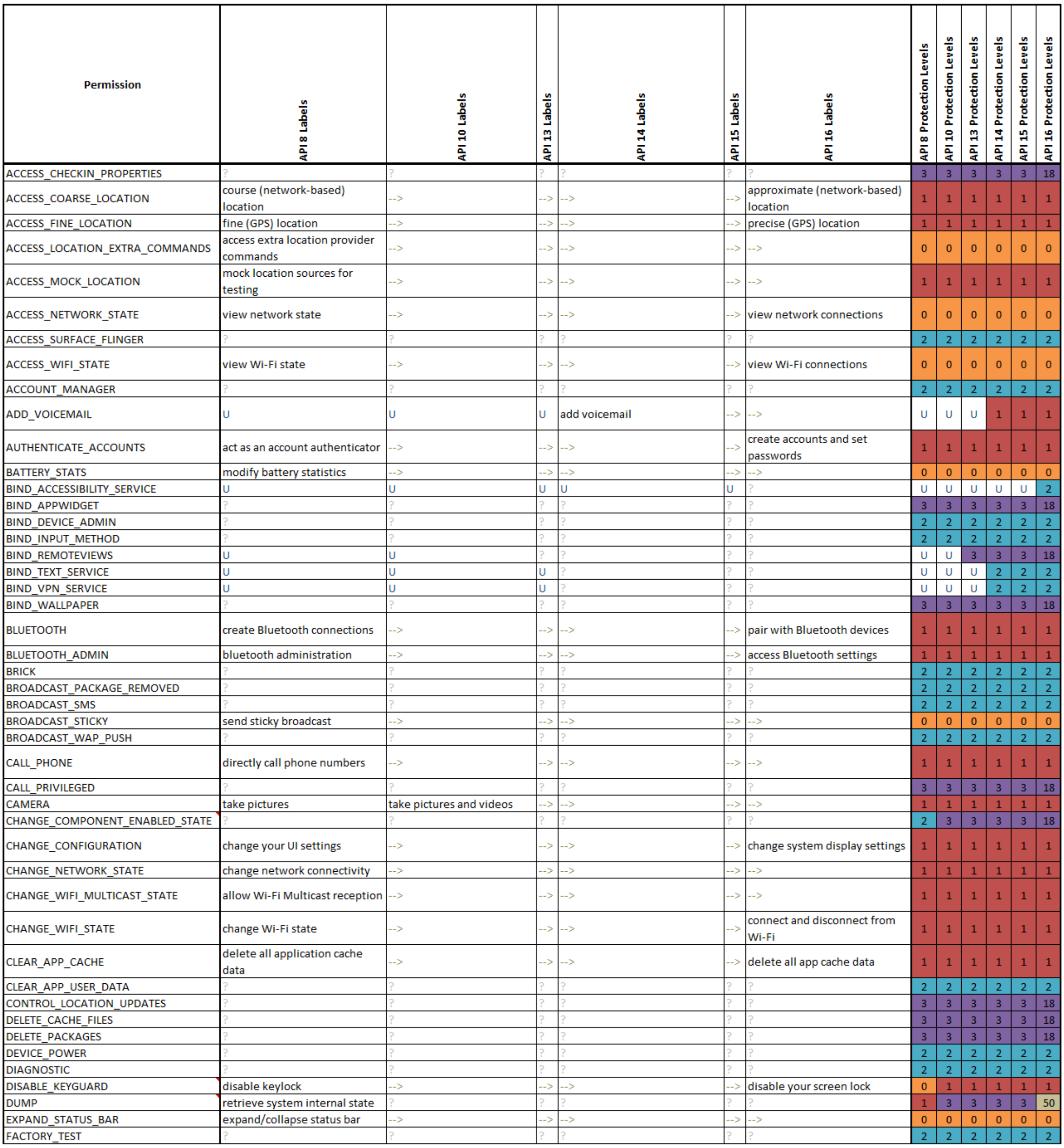
It is worth noting that in Table 3.2 the protection level values between API14 and API15 show no change with no new permissions added. There were seven new permissions observed between API13 and API14 however. Whilst this is experimentally the case, when using the SDK provided emulators, this does not align with the Android documentation — which identifies five permission additions in API14 and then two in API15 [7, 8]. As was previously discussed in Subsection 3.1.2 the documentation indicates six permission additions in API16 although I have only been able to experimentally identify four [9]. In both these cases, where permissions are documented as being introduced in a particular platform release, I have been granted those permissions on a previous release. This may be because they were introduced in the platform before officially being announced. At least in the case of the two permissions from API16, however, these were not just granted to my *Permission Test* app on the proceeding Android platform version, but were granted on every platform version I have tested.

3.2.3 Detailed Results

The detailed results of the Permission Evolution analysis are presented in the tables of Figures 3.6, 3.7, 3.8, 3.9, 3.10 and 3.11. Due to the complexity of the investigation and the results recorded, the key defined in Table 3.3 is required to interpret the results.

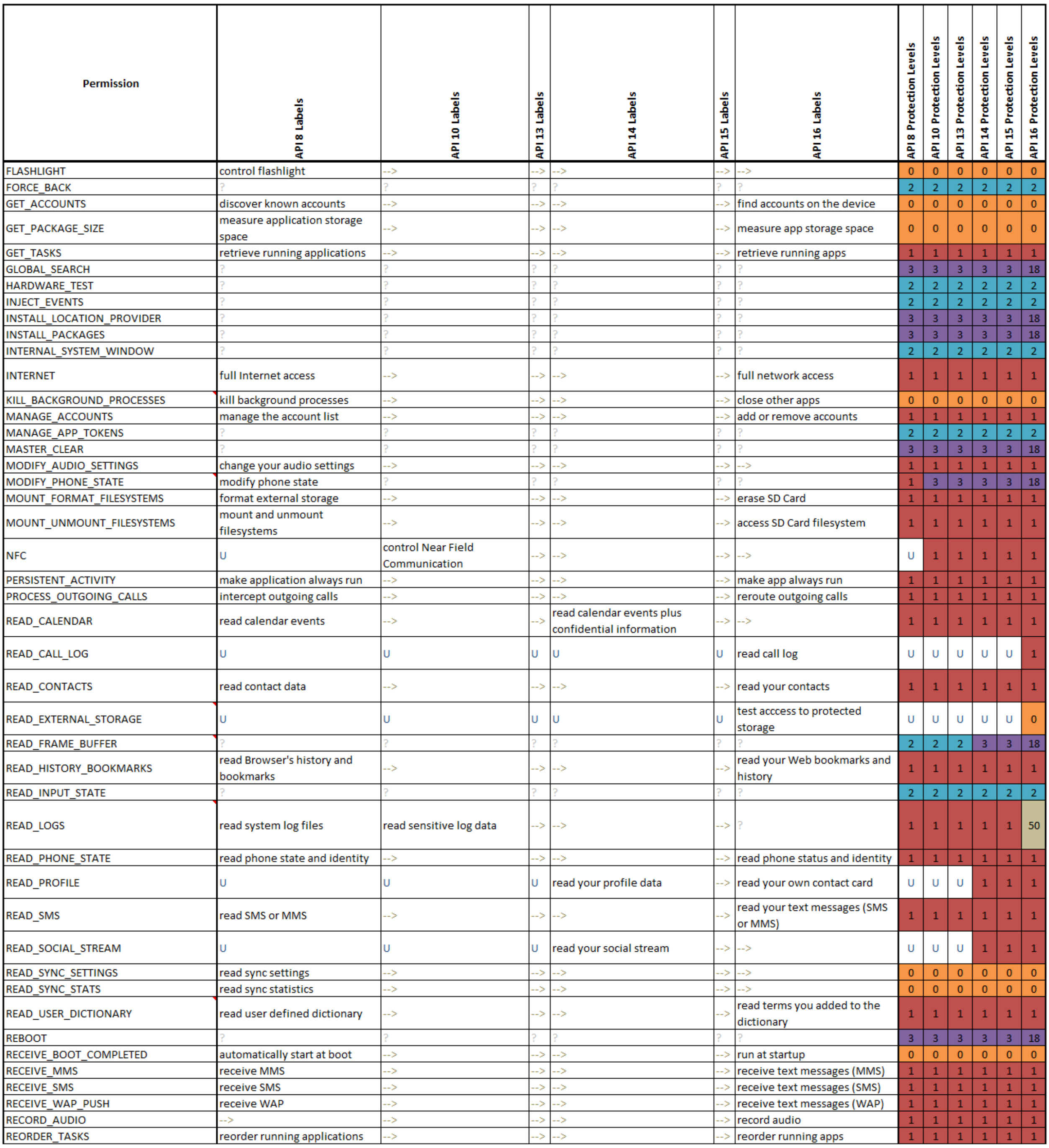
Column Group	**Entry**	**Description**
Permission		
	red corner flag	Row with result of particular interest
Labels		
	?	Undetermined permission label
	- - >	Permission label unchanged since last value
	U	‘Unknown permission’
Protection Levels		
	U	‘Unknown permission’
	0	`‘normal’` (0) permission
	1	`‘dangerous’` (1) permission
	2	`‘signature’` (2) permission
	3	`‘signatureOrSystem’` (3) permission
	18	`‘0x12’` (18) permission
	50	`‘0x32’` (50) permission
Group		
	UNKNOWN	Undetermined group
	Y	Permission present in this group

Table 3.3: Key for Figures 3.6 through 3.11



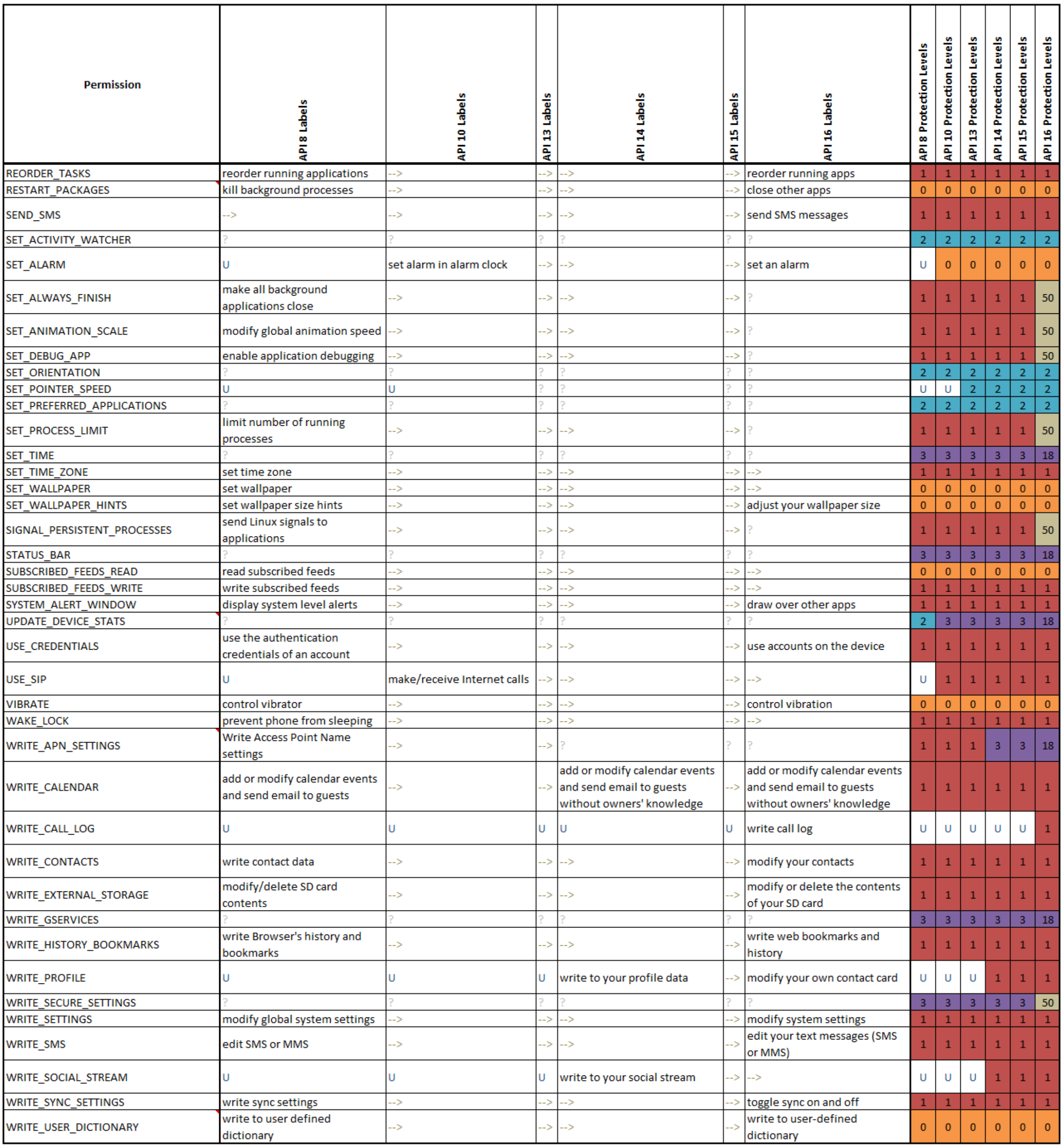
Permission	API 8 Labels	API 10 Labels	API 13 Labels	API 14 Labels	API 15 Labels	API 16 Labels	API 8 Protection Levels	API 10 Protection Levels	API 13 Protection Levels	API 14 Protection Levels	API 15 Protection Levels	API 16 Protection Levels
ACCESS_CHECKIN_PROPERTIES	?	?	?	?	?	?	3	3	3	3	3	18
ACCESS_COARSE_LOCATION	course (network-based) location	-->	-->	-->	-->	approximate (network-based) location	1	1	1	1	1	1
ACCESS_FINE_LOCATION	fine (GPS) location	-->	-->	-->	-->	precise (GPS) location	1	1	1	1	1	1
ACCESS_LOCATION_EXTRA_COMMANDS	access extra location provider commands	-->	-->	-->	-->	-->	0	0	0	0	0	0
ACCESS_MOCK_LOCATION	mock location sources for testing	-->	-->	-->	-->	-->	1	1	1	1	1	1
ACCESS_NETWORK_STATE	view network state	-->	-->	-->	-->	view network connections	0	0	0	0	0	0
ACCESS_SURFACE_FLINGER	?	?	?	?	?	?	2	2	2	2	2	2
ACCESS_WIFI_STATE	view Wi-Fi state	-->	-->	-->	-->	view Wi-Fi connections	0	0	0	0	0	0
ACCOUNT_MANAGER	?	?	?	?	?	?	2	2	2	2	2	2
ADD_VOICEMAIL	U	U	U	add voicemail	-->	-->	U	U	U	1	1	1
AUTHENTICATE_ACCOUNTS	act as an account authenticator	-->	-->	-->	-->	create accounts and set passwords	1	1	1	1	1	1
BATTERY_STATS	modify battery statistics	-->	-->	-->	-->	-->	0	0	0	0	0	0
BIND_ACCESSIBILITY_SERVICE	U	U	U	U	U	?	U	U	U	U	U	2
BIND_APPWIDGET	?	?	?	?	?	?	3	3	3	3	3	18
BIND_DEVICE_ADMIN	?	?	?	?	?	?	2	2	2	2	2	2
BIND_INPUT_METHOD	?	?	?	?	?	?	2	2	2	2	2	2
BIND_REMOTEVIEWS	U	U	?	?	?	?	U	U	3	3	3	18
BIND_TEXT_SERVICE	U	U	U	?	?	?	U	U	U	2	2	2
BIND_VPN_SERVICE	U	U	U	?	?	?	U	U	U	2	2	2
BIND_WALLPAPER	?	?	?	?	?	?	3	3	3	3	3	18
BLUETOOTH	create Bluetooth connections	-->	-->	-->	-->	pair with Bluetooth devices	1	1	1	1	1	1
BLUETOOTH_ADMIN	bluetooth administration	-->	-->	-->	-->	access Bluetooth settings	1	1	1	1	1	1
BRICK	?	?	?	?	?	?	2	2	2	2	2	2
BROADCAST_PACKAGE_REMOVED	?	?	?	?	?	?	2	2	2	2	2	2
BROADCAST_SMS	?	?	?	?	?	?	2	2	2	2	2	2
BROADCAST_STICKY	send sticky broadcast	-->	-->	-->	-->	-->	0	0	0	0	0	0
BROADCAST_WAP_PUSH	?	?	?	?	?	?	2	2	2	2	2	2
CALL_PHONE	directly call phone numbers	-->	-->	-->	-->	-->	1	1	1	1	1	1
CALL_PRIVILEGED	?	?	?	?	?	?	3	3	3	3	3	18
CAMERA	take pictures	take pictures and videos	-->	-->	-->	-->	1	1	1	1	1	1
CHANGE_COMPONENT_ENABLED_STATE	?	?	?	?	?	?	2	3	3	3	3	18
CHANGE_CONFIGURATION	change your UI settings	-->	-->	-->	-->	change system display settings	1	1	1	1	1	1
CHANGE_NETWORK_STATE	change network connectivity	-->	-->	-->	-->	-->	1	1	1	1	1	1
CHANGE_WIFI_MULTICAST_STATE	allow Wi-Fi Multicast reception	-->	-->	-->	-->	-->	1	1	1	1	1	1
CHANGE_WIFI_STATE	change Wi-Fi state	-->	-->	-->	-->	connect and disconnect from Wi-Fi	1	1	1	1	1	1
CLEAR_APP_CACHE	delete all application cache data	-->	-->	-->	-->	delete all app cache data	1	1	1	1	1	1
CLEAR_APP_USER_DATA	?	?	?	?	?	?	2	2	2	2	2	2
CONTROL_LOCATION_UPDATES	?	?	?	?	?	?	3	3	3	3	3	18
DELETE_CACHE_FILES	?	?	?	?	?	?	3	3	3	3	3	18
DELETE_PACKAGES	?	?	?	?	?	?	3	3	3	3	3	18
DEVICE_POWER	?	?	?	?	?	?	2	2	2	2	2	2
DIAGNOSTIC	?	?	?	?	?	?	2	2	2	2	2	2
DISABLE_KEYGUARD	disable keylock	-->	-->	-->	-->	disable your screen lock	0	1	1	1	1	1
DUMP	retrieve system internal state	?	?	?	?	?	1	3	3	3	3	50
EXPAND_STATUS_BAR	expand/collapse status bar	-->	-->	-->	-->	-->	0	0	0	0	0	0
FACTORY_TEST	?	?	?	?	?	?	2	2	2	2	2	2

Figure 3.6: Permission Evolution results - Protection Levels - part 1



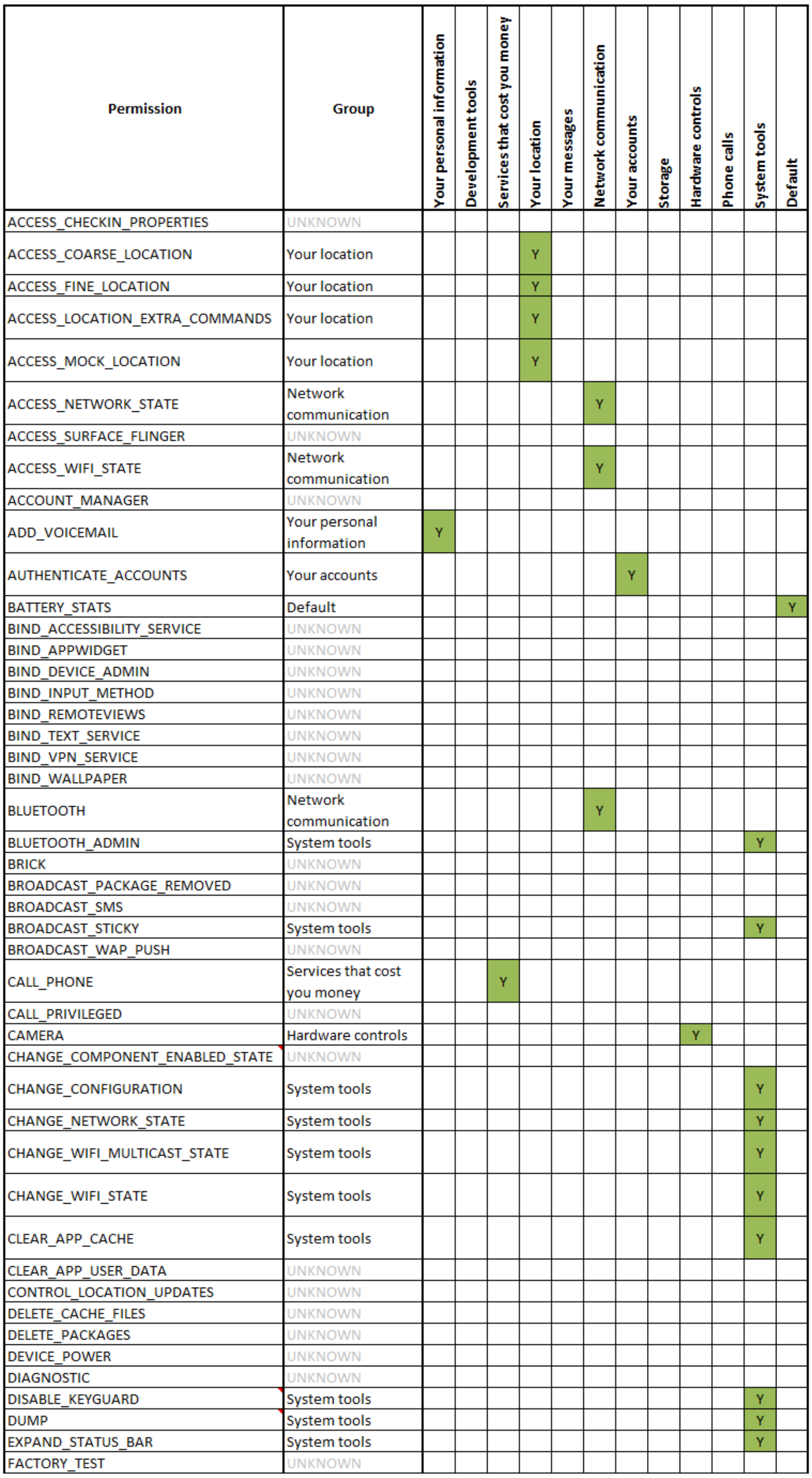
Permission	API 8 Labels	API 10 Labels	API 13 Labels	API 14 Labels	API 15 Labels	API 16 Labels	API 8 Protection Levels	API 10 Protection Levels	API 13 Protection Levels	API 14 Protection Levels	API 15 Protection Levels	API 16 Protection Levels
FLASHLIGHT	control flashlight	-->	-->	-->	-->	-->	0	0	0	0	0	0
FORCE_BACK	?	?	?	?	?	?	2	2	2	2	2	2
GET_ACCOUNTS	discover known accounts	-->	-->	-->	-->	find accounts on the device	0	0	0	0	0	0
GET_PACKAGE_SIZE	measure application storage space	-->	-->	-->	-->	measure app storage space	0	0	0	0	0	0
GET_TASKS	retrieve running applications	-->	-->	-->	-->	retrieve running apps	1	1	1	1	1	1
GLOBAL_SEARCH	?	?	?	?	?	?	3	3	3	3	3	18
HARDWARE_TEST	?	?	?	?	?	?	2	2	2	2	2	2
INJECT_EVENTS	?	?	?	?	?	?	2	2	2	2	2	2
INSTALL_LOCATION_PROVIDER	?	?	?	?	?	?	3	3	3	3	3	18
INSTALL_PACKAGES	?	?	?	?	?	?	3	3	3	3	3	18
INTERNAL_SYSTEM_WINDOW	?	?	?	?	?	?	2	2	2	2	2	2
INTERNET	full Internet access	-->	-->	-->	-->	full network access	1	1	1	1	1	1
KILL_BACKGROUND_PROCESSES	kill background processes	-->	-->	-->	-->	close other apps	0	0	0	0	0	0
MANAGE_ACCOUNTS	manage the account list	-->	-->	-->	-->	add or remove accounts	1	1	1	1	1	1
MANAGE_APP_TOKENS	?	?	?	?	?	?	2	2	2	2	2	2
MASTER_CLEAR	?	?	?	?	?	?	3	3	3	3	3	18
MODIFY_AUDIO_SETTINGS	change your audio settings	-->	-->	-->	-->	-->	1	1	1	1	1	1
MODIFY_PHONE_STATE	modify phone state	?	?	?	?	?	1	3	3	3	3	18
MOUNT_FORMAT_FILESYSTEMS	format external storage	-->	-->	-->	-->	erase SD Card	1	1	1	1	1	1
MOUNT_UNMOUNT_FILESYSTEMS	mount and unmount filesystems	-->	-->	-->	-->	access SD Card filesystem	1	1	1	1	1	1
NFC	U	control Near Field Communication	-->	-->	-->	-->	U	1	1	1	1	1
PERSISTENT_ACTIVITY	make application always run	-->	-->	-->	-->	make app always run	1	1	1	1	1	1
PROCESS_OUTGOING_CALLS	intercept outgoing calls	-->	-->	-->	-->	reroute outgoing calls	1	1	1	1	1	1
READ_CALENDAR	read calendar events	-->	-->	read calendar events plus confidential information	-->	-->	1	1	1	1	1	1
READ_CALL_LOG	U	U	U	U	U	read call log	U	U	U	U	U	1
READ_CONTACTS	read contact data	-->	-->	-->	-->	read your contacts	1	1	1	1	1	1
READ_EXTERNAL_STORAGE	U	U	U	U	U	test acccess to protected storage	U	U	U	U	U	0
READ_FRAME_BUFFER	?	?	?	?	?	?	2	2	2	3	3	18
READ_HISTORY_BOOKMARKS	read Browser's history and bookmarks	-->	-->	-->	-->	read your Web bookmarks and history	1	1	1	1	1	1
READ_INPUT_STATE	?	?	?	?	?	?	2	2	2	2	2	2
READ_LOGS	read system log files	read sensitive log data	-->	-->	-->	?	1	1	1	1	1	50
READ_PHONE_STATE	read phone state and identity	-->	-->	-->	-->	read phone status and identity	1	1	1	1	1	1
READ_PROFILE	U	U	U	read your profile data	-->	read your own contact card	U	U	U	1	1	1
READ_SMS	read SMS or MMS	-->	-->	-->	-->	read your text messages (SMS or MMS)	1	1	1	1	1	1
READ_SOCIAL_STREAM	U	U	U	read your social stream	-->	-->	U	U	U	1	1	1
READ_SYNC_SETTINGS	read sync settings	-->	-->	-->	-->	-->	0	0	0	0	0	0
READ_SYNC_STATS	read sync statistics	-->	-->	-->	-->	-->	0	0	0	0	0	0
READ_USER_DICTIONARY	read user defined dictionary	-->	-->	-->	-->	read terms you added to the dictionary	1	1	1	1	1	1
REBOOT	?	?	?	?	?	?	3	3	3	3	3	18
RECEIVE_BOOT_COMPLETED	automatically start at boot	-->	-->	-->	-->	run at startup	0	0	0	0	0	0
RECEIVE_MMS	receive MMS	-->	-->	-->	-->	receive text messages (MMS)	1	1	1	1	1	1
RECEIVE_SMS	receive SMS	-->	-->	-->	-->	receive text messages (SMS)	1	1	1	1	1	1
RECEIVE_WAP_PUSH	receive WAP	-->	-->	-->	-->	receive text messages (WAP)	1	1	1	1	1	1
RECORD_AUDIO	-->	-->	-->	-->	-->	record audio	1	1	1	1	1	1
REORDER_TASKS	reorder running applications	-->	-->	-->	-->	reorder running apps	1	1	1	1	1	1

Figure 3.7: Permission Evolution results - Protection Levels - part 2



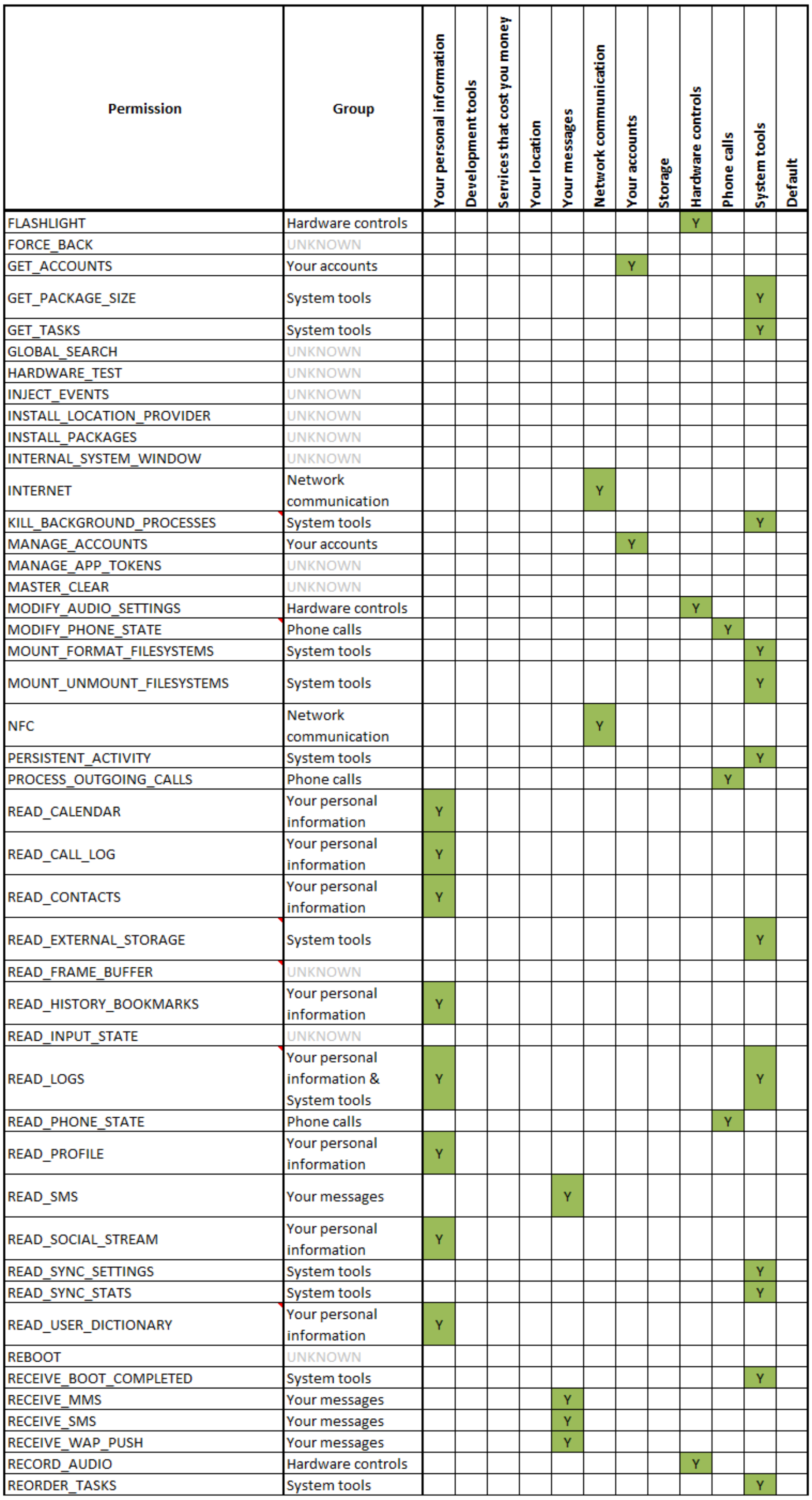
Permission	API 8 Labels	API 10 Labels	API 13 Labels	API 14 Labels	API 15 Labels	API 16 Labels	API 8 Protection Levels	API 10 Protection Levels	API 13 Protection Levels	API 14 Protection Levels	API 15 Protection Levels	API 16 Protection Levels
REORDER_TASKS	reorder running applications	-->	-->	-->	-->	reorder running apps	1	1	1	1	1	1
RESTART_PACKAGES	kill background processes	-->	-->	-->	-->	close other apps	0	0	0	0	0	0
SEND_SMS	-->	-->	-->	-->	-->	send SMS messages	1	1	1	1	1	1
SET_ACTIVITY_WATCHER	?	?	?	?	?	?	2	2	2	2	2	2
SET_ALARM	U	set alarm in alarm clock	-->	-->	-->	set an alarm	U	0	0	0	0	0
SET_ALWAYS_FINISH	make all background applications close	-->	-->	-->	-->	?	1	1	1	1	1	50
SET_ANIMATION_SCALE	modify global animation speed	-->	-->	-->	-->	?	1	1	1	1	1	50
SET_DEBUG_APP	enable application debugging	-->	-->	-->	-->	?	1	1	1	1	1	50
SET_ORIENTATION	?	?	?	?	?	?	2	2	2	2	2	2
SET_POINTER_SPEED	U	U	?	?	?	?	U	U	2	2	2	2
SET_PREFERRED_APPLICATIONS	?	?	?	?	?	?	2	2	2	2	2	2
SET_PROCESS_LIMIT	limit number of running processes	-->	-->	-->	-->	?	1	1	1	1	1	50
SET_TIME	?	?	?	?	?	?	3	3	3	3	3	18
SET_TIME_ZONE	set time zone	-->	-->	-->	-->	-->	1	1	1	1	1	1
SET_WALLPAPER	set wallpaper	-->	-->	-->	-->	-->	0	0	0	0	0	0
SET_WALLPAPER_HINTS	set wallpaper size hints	-->	-->	-->	-->	adjust your wallpaper size	0	0	0	0	0	0
SIGNAL_PERSISTENT_PROCESSES	send Linux signals to applications	-->	-->	-->	-->	?	1	1	1	1	1	50
STATUS_BAR	?	?	?	?	?	?	3	3	3	3	3	18
SUBSCRIBED_FEEDS_READ	read subscribed feeds	-->	-->	-->	-->	-->	0	0	0	0	0	0
SUBSCRIBED_FEEDS_WRITE	write subscribed feeds	-->	-->	-->	-->	-->	1	1	1	1	1	1
SYSTEM_ALERT_WINDOW	display system level alerts	-->	-->	-->	-->	draw over other apps	1	1	1	1	1	1
UPDATE_DEVICE_STATS	?	?	?	?	?	?	2	3	3	3	3	18
USE_CREDENTIALS	use the authentication credentials of an account	-->	-->	-->	-->	use accounts on the device	1	1	1	1	1	1
USE_SIP	U	make/receive Internet calls	-->	-->	-->	-->	U	1	1	1	1	1
VIBRATE	control vibrator	-->	-->	-->	-->	control vibration	0	0	0	0	0	0
WAKE_LOCK	prevent phone from sleeping	-->	-->	-->	-->	-->	1	1	1	1	1	1
WRITE_APN_SETTINGS	Write Access Point Name settings	-->	-->	?	?	?	1	1	1	3	3	18
WRITE_CALENDAR	add or modify calendar events and send email to guests	-->	-->	add or modify calendar events and send email to guests without owners' knowledge	-->	add or modify calendar events and send email to guests without owners' knowledge	1	1	1	1	1	1
WRITE_CALL_LOG	U	U	U	U	U	write call log	U	U	U	U	U	1
WRITE_CONTACTS	write contact data	-->	-->	-->	-->	modify your contacts	1	1	1	1	1	1
WRITE_EXTERNAL_STORAGE	modify/delete SD card contents	-->	-->	-->	-->	modify or delete the contents of your SD card	1	1	1	1	1	1
WRITE_GSERVICES	?	?	?	?	?	?	3	3	3	3	3	18
WRITE_HISTORY_BOOKMARKS	write Browser's history and bookmarks	-->	-->	-->	-->	write web bookmarks and history	1	1	1	1	1	1
WRITE_PROFILE	U	U	U	write to your profile data	-->	modify your own contact card	U	U	U	1	1	1
WRITE_SECURE_SETTINGS	?	?	?	?	?	?	3	3	3	3	3	50
WRITE_SETTINGS	modify global system settings	-->	-->	-->	-->	modify system settings	1	1	1	1	1	1
WRITE_SMS	edit SMS or MMS	-->	-->	-->	-->	edit your text messages (SMS or MMS)	1	1	1	1	1	1
WRITE_SOCIAL_STREAM	U	U	U	write to your social stream	-->	-->	U	U	U	1	1	1
WRITE_SYNC_SETTINGS	write sync settings	-->	-->	-->	-->	toggle sync on and off	1	1	1	1	1	1
WRITE_USER_DICTIONARY	write to user defined dictionary	-->	-->	-->	-->	write to user-defined dictionary	0	0	0	0	0	0

Figure 3.8: Permission Evolution results - Protection Levels - part 3



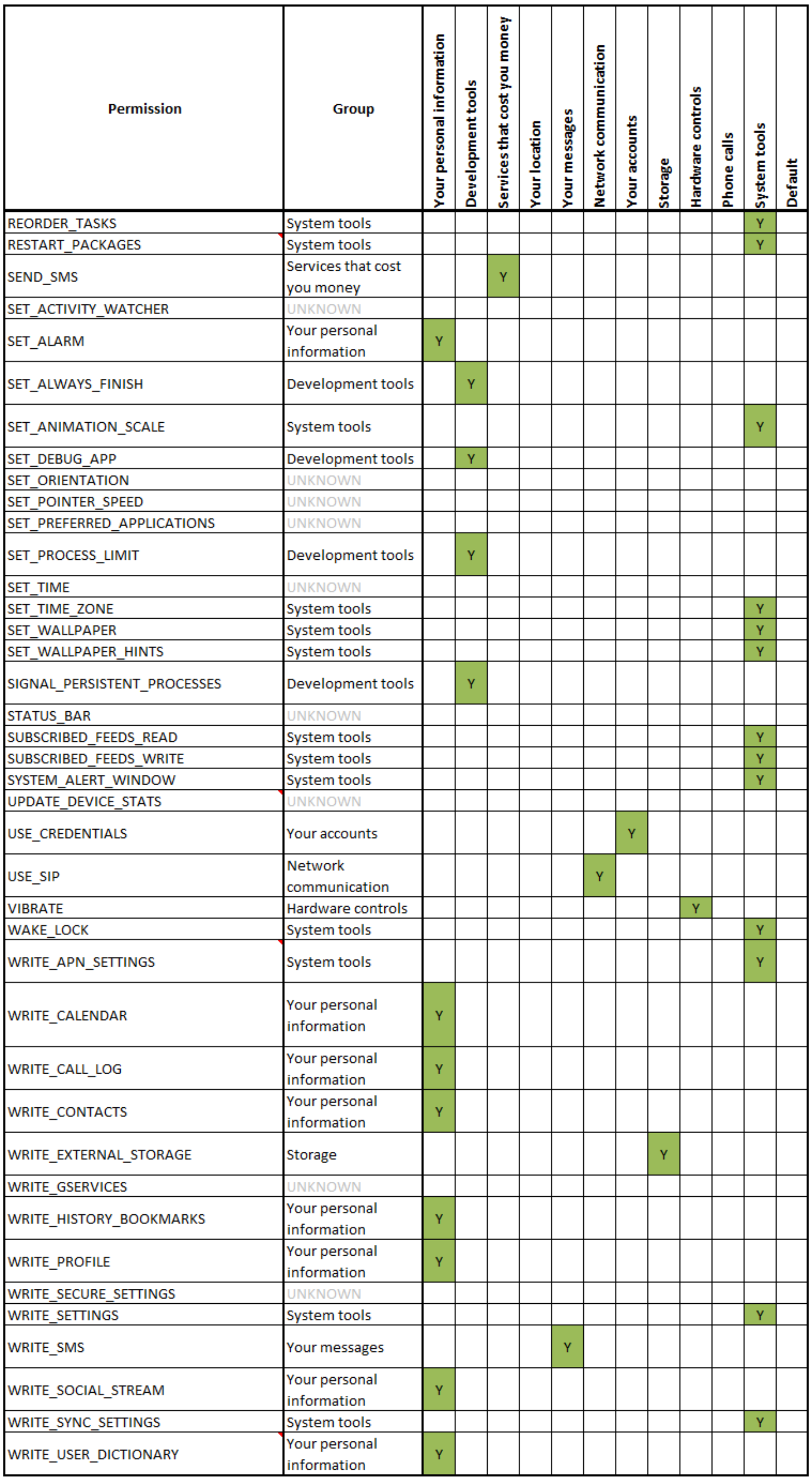
Permission	Group	Your personal information	Development tools	Services that cost you money	Your location	Your messages	Network communication	Your accounts	Storage	Hardware controls	Phone calls	System tools	Default
ACCESS_CHECKIN_PROPERTIES	UNKNOWN												
ACCESS_COARSE_LOCATION	Your location				Y								
ACCESS_FINE_LOCATION	Your location				Y								
ACCESS_LOCATION_EXTRA_COMMANDS	Your location				Y								
ACCESS_MOCK_LOCATION	Your location				Y								
ACCESS_NETWORK_STATE	Network communication						Y						
ACCESS_SURFACE_FLINGER	UNKNOWN												
ACCESS_WIFI_STATE	Network communication						Y						
ACCOUNT_MANAGER	UNKNOWN												
ADD_VOICEMAIL	Your personal information	Y											
AUTHENTICATE_ACCOUNTS	Your accounts							Y					
BATTERY_STATS	Default												Y
BIND_ACCESSIBILITY_SERVICE	UNKNOWN												
BIND_APPWIDGET	UNKNOWN												
BIND_DEVICE_ADMIN	UNKNOWN												
BIND_INPUT_METHOD	UNKNOWN												
BIND_REMOTEVIEWS	UNKNOWN												
BIND_TEXT_SERVICE	UNKNOWN												
BIND_VPN_SERVICE	UNKNOWN												
BIND_WALLPAPER	UNKNOWN												
BLUETOOTH	Network communication						Y						
BLUETOOTH_ADMIN	System tools											Y	
BRICK	UNKNOWN												
BROADCAST_PACKAGE_REMOVED	UNKNOWN												
BROADCAST_SMS	UNKNOWN												
BROADCAST_STICKY	System tools											Y	
BROADCAST_WAP_PUSH	UNKNOWN												
CALL_PHONE	Services that cost you money			Y									
CALL_PRIVILEGED	UNKNOWN												
CAMERA	Hardware controls									Y			
CHANGE_COMPONENT_ENABLED_STATE	UNKNOWN												
CHANGE_CONFIGURATION	System tools											Y	
CHANGE_NETWORK_STATE	System tools											Y	
CHANGE_WIFI_MULTICAST_STATE	System tools											Y	
CHANGE_WIFI_STATE	System tools											Y	
CLEAR_APP_CACHE	System tools											Y	
CLEAR_APP_USER_DATA	UNKNOWN												
CONTROL_LOCATION_UPDATES	UNKNOWN												
DELETE_CACHE_FILES	UNKNOWN												
DELETE_PACKAGES	UNKNOWN												
DEVICE_POWER	UNKNOWN												
DIAGNOSTIC	UNKNOWN												
DISABLE_KEYGUARD	System tools											Y	
DUMP	System tools											Y	
EXPAND_STATUS_BAR	System tools											Y	
FACTORY_TEST	UNKNOWN												

Figure 3.9: Permission Evolution results - Groups - part 1



Permission	Group	Your personal information	Development tools	Services that cost you money	Your location	Your messages	Network communication	Your accounts	Storage	Hardware controls	Phone calls	System tools	Default
FLASHLIGHT	Hardware controls									Y			
FORCE_BACK	UNKNOWN												
GET_ACCOUNTS	Your accounts							Y					
GET_PACKAGE_SIZE	System tools											Y	
GET_TASKS	System tools											Y	
GLOBAL_SEARCH	UNKNOWN												
HARDWARE_TEST	UNKNOWN												
INJECT_EVENTS	UNKNOWN												
INSTALL_LOCATION_PROVIDER	UNKNOWN												
INSTALL_PACKAGES	UNKNOWN												
INTERNAL_SYSTEM_WINDOW	UNKNOWN												
INTERNET	Network communication						Y						
KILL_BACKGROUND_PROCESSES	System tools											Y	
MANAGE_ACCOUNTS	Your accounts							Y					
MANAGE_APP_TOKENS	UNKNOWN												
MASTER_CLEAR	UNKNOWN												
MODIFY_AUDIO_SETTINGS	Hardware controls									Y			
MODIFY_PHONE_STATE	Phone calls										Y		
MOUNT_FORMAT_FILESYSTEMS	System tools											Y	
MOUNT_UNMOUNT_FILESYSTEMS	System tools											Y	
NFC	Network communication						Y						
PERSISTENT_ACTIVITY	System tools											Y	
PROCESS_OUTGOING_CALLS	Phone calls										Y		
READ_CALENDAR	Your personal information	Y											
READ_CALL_LOG	Your personal information	Y											
READ_CONTACTS	Your personal information	Y											
READ_EXTERNAL_STORAGE	System tools											Y	
READ_FRAME_BUFFER	UNKNOWN												
READ_HISTORY_BOOKMARKS	Your personal information	Y											
READ_INPUT_STATE	UNKNOWN												
READ_LOGS	Your personal information & System tools	Y										Y	
READ_PHONE_STATE	Phone calls										Y		
READ_PROFILE	Your personal information	Y											
READ_SMS	Your messages					Y							
READ_SOCIAL_STREAM	Your personal information	Y											
READ_SYNC_SETTINGS	System tools											Y	
READ_SYNC_STATS	System tools											Y	
READ_USER_DICTIONARY	Your personal information	Y											
REBOOT	UNKNOWN												
RECEIVE_BOOT_COMPLETED	System tools											Y	
RECEIVE_MMS	Your messages					Y							
RECEIVE_SMS	Your messages					Y							
RECEIVE_WAP_PUSH	Your messages					Y							
RECORD_AUDIO	Hardware controls									Y			
REORDER_TASKS	System tools											Y	

Figure 3.10: Permission Evolution results - Groups - part 2



Permission	Group	Your personal information	Development tools	Services that cost you money	Your location	Your messages	Network communication	Your accounts	Storage	Hardware controls	Phone calls	System tools	Default
REORDER_TASKS	System tools											Y	
RESTART_PACKAGES	System tools											Y	
SEND_SMS	Services that cost you money			Y									
SET_ACTIVITY_WATCHER	UNKNOWN												
SET_ALARM	Your personal information	Y											
SET_ALWAYS_FINISH	Development tools		Y										
SET_ANIMATION_SCALE	System tools											Y	
SET_DEBUG_APP	Development tools		Y										
SET_ORIENTATION	UNKNOWN												
SET_POINTER_SPEED	UNKNOWN												
SET_PREFERRED_APPLICATIONS	UNKNOWN												
SET_PROCESS_LIMIT	Development tools		Y										
SET_TIME	UNKNOWN												
SET_TIME_ZONE	System tools											Y	
SET_WALLPAPER	System tools											Y	
SET_WALLPAPER_HINTS	System tools											Y	
SIGNAL_PERSISTENT_PROCESSES	Development tools		Y										
STATUS_BAR	UNKNOWN												
SUBSCRIBED_FEEDS_READ	System tools											Y	
SUBSCRIBED_FEEDS_WRITE	System tools											Y	
SYSTEM_ALERT_WINDOW	System tools											Y	
UPDATE_DEVICE_STATS	UNKNOWN												
USE_CREDENTIALS	Your accounts							Y					
USE_SIP	Network communication						Y						
VIBRATE	Hardware controls									Y			
WAKE_LOCK	System tools											Y	
WRITE_APN_SETTINGS	System tools											Y	
WRITE_CALENDAR	Your personal information	Y											
WRITE_CALL_LOG	Your personal information	Y											
WRITE_CONTACTS	Your personal information	Y											
WRITE_EXTERNAL_STORAGE	Storage								Y				
WRITE_GSERVICES	UNKNOWN												
WRITE_HISTORY_BOOKMARKS	Your personal information	Y											
WRITE_PROFILE	Your personal information	Y											
WRITE_SECURE_SETTINGS	UNKNOWN												
WRITE_SETTINGS	System tools											Y	
WRITE_SMS	Your messages					Y							
WRITE_SOCIAL_STREAM	Your personal information	Y											
WRITE_SYNC_SETTINGS	System tools											Y	
WRITE_USER_DICTIONARY	Your personal information	Y											

Figure 3.11: Permission Evolution results - Groups - part 3

As identified in the key, certain rows within the results are marked with a small red corner flag in the Permission column. These rows have results of particular interest. The permissions in question are listed here for ease of reference.

- 'CHANGE_COMPONENT_ENABLED_STATE'
- 'DISABLE_KEYGUARD'
- 'DUMP'
- 'KILL_BACKGROUND_PROCESSES'
- 'MODIFY_PHONE_STATE'
- 'READ_EXTERNAL_STORAGE'
- 'READ_FRAME_BUFFER'
- 'READ_LOGS'
- 'READ_USER_DICTIONARY'
- 'RESTART_PACKAGES'
- 'UPDATE_DEVICE_STATS'
- 'WRITE_APN_SETTINGS'
- 'WRITE_USER_DICTIONARY'

Of these thirteen interesting permissions, eight undergo significant protection level changes across the platform versions tested and are detailed individually in Table 3.4. For these purposes, the definition of a significant protection level change is any change of protection level not including the introduction of a new permission (i.e. a change from 'Unknown') or a change from protection level 3 to 18 ('signatureOrSystem' to 'signature' or 'system'). In each case, the protection level changes which do occur have the result of increasing the severity of the permissions' marking. There is one change from 'normal' (0) to 'dangerous' (1), three changes from 'normal' (1) to 'signature' (3), one change from 'normal' (0) to '0x32' (50), three changes from 'dangerous' (2) to 'signature' (3) and one change from 'signature' (3) to '0x32' (50). Of these changes, five happen between API8 and API10, two between API13 and API14 and two between API15 and API16.

	Protection Level					
Permission	**API8**	**API10**	**API13**	**API14**	**API15**	**API16**
'CHANGE_COMPONENT_ENABLED_STATE'	2	3	3	3	3	18
'DISABLE_KEYGUARD'	0	1	1	1	1	1
'DUMP'	1	3	3	3	3	50
'MODIFY_PHONE_STATE'	1	3	3	3	3	18
'READ_FRAME_BUFFER'	2	2	2	3	3	18
'READ_LOGS'	1	1	1	1	1	50
'UPDATE_DEVICE_STATS'	2	3	3	3	3	18
'WRITE_APN_SETTINGS'	1	1	1	3	3	18

Table 3.4: Protection Level changes across platform versions

Within the analysis results, the 'READ_LOGS' permission is interesting for two reasons. As well as undergoing a protection level change as detailed in Table 3.4, it also is the only permission to undergo a group change. In API8 the 'READ_LOGS' permission is in the 'System tools' group, whilst in API10 and onwards it has been moved into the 'Your personal information' group. Interestingly, this change is made separate to any change in the protection level.

As has been identified in Subsection 3.1.2, the two permissions 'KILL_BACKGROUND_PROCESS' and 'RESTART_PACKAGES' utilise a single label: 'kill background processes'. Whilst this may be a planned and acceptable scenario, the permission names suggest different functionality which the user would be unable to distinguish between from the single label. In fact both of these permissions are assigned the 'normal' protection level and so by default would be hidden, in the collapsed view, during the pre-installation permission authorisation.

The permissions 'READ_USER_DICTIONARY' and 'WRITE_USER_DICTIONARY' are the two permissions mentioned in the summary results (Subsection 3.2.2) which are documented as being new to API16 along with the four permissions which were experimentally verified ('BIND_ACCESSIBILITY_SERVICE', 'READ_CALL_LOG', 'READ_EXTERNAL_STORAGE' and 'WRITE_CALL_LOG'). During the Permission Evolution analysis these two dictionary related permissions were granted by every version of the Android platform. It is unclear why these are documented as new to API16 based on this evidence.

The final interesting permission is 'READ_EXTERNAL_STORAGE'. This is one of the four recently introduced permissions and at this time, whilst the permission exists, the documentation explains that there is currently no mandatory enforcement of the capability associated with this permission. In fact Jelly Bean 4.1 (API16) currently has a developer option 'Protect SD card - Apps must request permission to read SD card' which is designed to allow the testing of this future functionality. The emulator included with revision 20 of the Android SDK does not seem to alter its behaviour in line with this option however, with reading from external storage possible without the 'READ_EXTERNAL_STORAGE' permission, whether the option is checked or not.

Concluding Remarks These various investigations have identified considerable information regarding the categorisation and evolution of permissions within the Android platform. As a supplement to the Android documentation, these results allow a far more detailed understanding of the inner workings of app permissions and have even called into question several pieces of the documentation which may need further checking.

Through the insights gained from these results, as well as the process of producing them a question has arisen which warrants further investigation. If duplicate permissions are ignored by the platform and misspelt permissions are assumed to be third-party permissions — what happens when a third-party permission is requested before the app which declares that third-party permission is installed? This question is considered in Chapter 4 along with a more important follow-up.

Chapter 4

Dormant Android Permission Requests

Having discussed the Android security architectures in Chapter 2 and then investigated the permissions architecture in Chapter 3, it was clear that the permissions management process in Android was both fundamental to security and complex in nature. To further understand the process of permission granting, I developed two new Android apps, *Permission Test Creator* and *Permission Test Requestor*, which enabled the investigation of third-party permission requests.

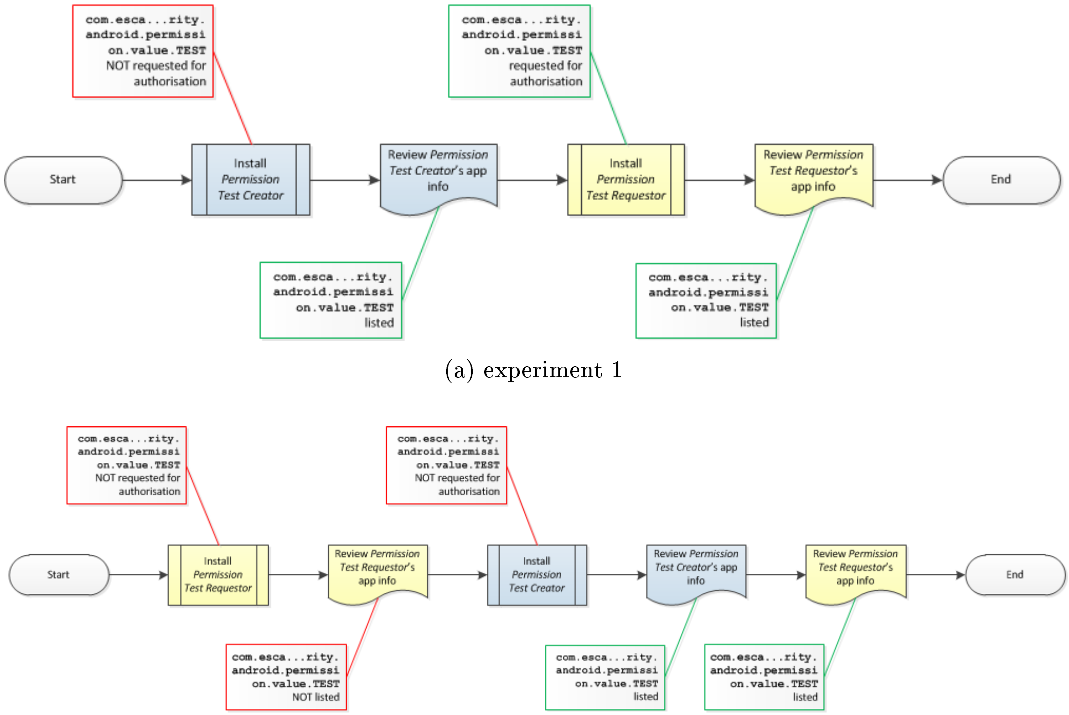
Permission Test Creator was written defining a third-party permission using the `<permission>` tag within its `'AndroidManifest.xml'` file [44]. The third-party permission was defined with the name `'com.escapadesinsecurity.android.permission.value.TEST'` and assigned the protection level `'dangerous'`. The label of the permission was set to 'This is the TEST permission's label' whilst the description was set to 'This is the TEST permission's description'. As well as defining this permission, *Permission Test Creator* also requested this permission and the `'INTERNET'` permission. The `'INTERNET'` permission was used here as a control permission, as had been the case with the *Permission Test* app in Chapter 3.

Permission Test Requestor does not define any new permissions, but as its name suggests its `'AndroidManifest.xml'` file includes a request for the permission defined by *Permission Test Creator* (`'com.escapadesinsecurity.android.permission.value.TEST'`) and no others.

4.1 Third-party Permission Requests Investigation

The investigation using *Permission Test Creator* and *Permission Test Requestor* was performed on the Galaxy Nexus device used within Subsection 3.1.2 (running Ice Cream Sandwich, Android version 4.0.4). This investigation took the form of two experiments using the two test apps installed in alternate orders. Figures 4.1a and 4.1b provide high level workflows for each of these experiments and include indications of the results which are shown through the various other figures accompanying this section.

Experiment 1 Firstly I installed *Permission Test Creator* on the Nexus device. During the installation, the pre-installation permission authorisation prompt shown in Figure 4.2a was displayed. You will notice that only one permission is prompted for user acceptance, even though the app requests two permissions — the `'INTERNET'` permission and the third-party permission it itself defines. Both permissions have a protection level of `'dangerous'` and so by protection level alone both permissions should be listed within the prompt. When the permissions granted to the app were checked post installation, it was clear to see, as in Figures 4.2b and 4.2c, that both permissions had been granted to the app even though only the `'INTERNET'` permission was displayed during authorisation. As a follow-up to the installation of *Permission Test Creator*, the *Permission Test Requestor* app was installed. When installed after *Permission Test Creator*,

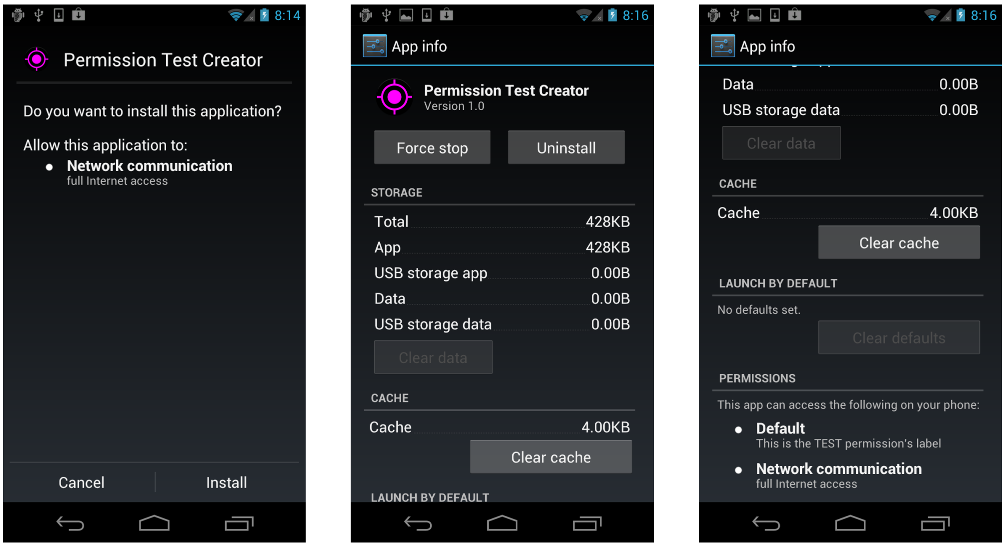

com.esca...rity.android.permission.value.TEST NOT requested for authorisation
com.esca...rity.android.permission.value.TEST requested for authorisation
Start
Install Permission Test Creator
Review Permission Test Creator's app info
Install Permission Test Requestor
Review Permission Test Requestor's app info
End
com.esca...rity.android.permission.value.TEST listed
com.esca...rity.android.permission.value.TEST listed
com.esca...rity.android.permission.value.TEST NOT requested for authorisation
com.esca...rity.android.permission.value.TEST NOT requested for authorisation
Start
Install Permission Test Requestor
Review Permission Test Requestor's app info
Install Permission Test Creator
Review Permission Test Creator's app info
Review Permission Test Requestor's app info
End
com.esca...rity.android.permission.value.TEST NOT listed
com.esca...rity.android.permission.value.TEST listed
com.esca...rity.android.permission.value.TEST listed

(a) experiment 1

(b) experiment 2

Figure 4.1: Third-party Permission Request Experiments

the permission authorisation prompt shown in Figure 4.3a was displayed for confirmation. This prompt includes the request for the third-party permission defined by *Permission Test Creator*.



Permission Test Creator

Do you want to install this application?

Allow this application to:

- **Network communication**
  full Internet access

Cancel | Install

App info

Permission Test Creator
Version 1.0

Force stop | Uninstall

STORAGE

Total	428KB
App	428KB
USB storage app	0.00B
Data	0.00B
USB storage data	0.00B

Clear data

CACHE

Cache	4.00KB

Clear cache

LAUNCH BY DEFAULT

App info

Data	0.00B
USB storage data	0.00B

Clear data

CACHE

Cache	4.00KB

Clear cache

LAUNCH BY DEFAULT

No defaults set.

Clear defaults

PERMISSIONS

This app can access the following on your phone:

- **Default**
  This is the TEST permission's label
- **Network communication**
  full Internet access

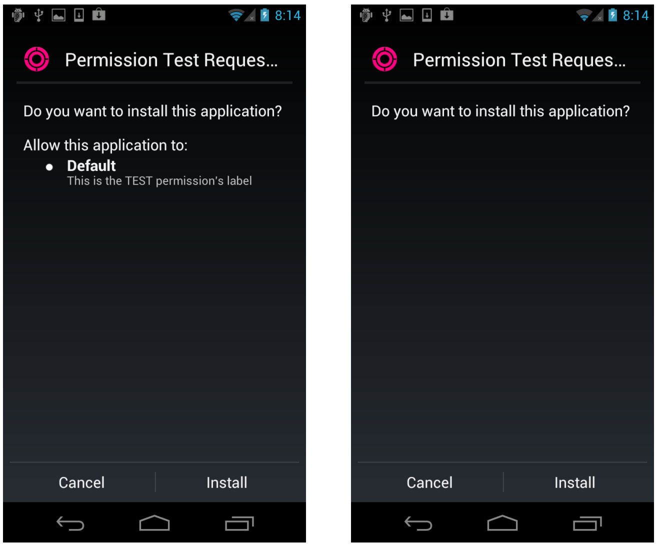
(a) install

(b) app info 1

(c) app info 2

Figure 4.2: *Permission Test Creator* installation and app information

These results suggest that the permission authorisation prompt could not display a permission that was not already registered within the system. They further suggest, that the process of installing an app which defines a third-party permission, completes a permission registration process and thus allows the newly declared permission to be shown for approval during installation of future apps who request it and subsequently as granted once they are installed.


Permission Test Reques...

Do you want to install this application?

Allow this application to:

- **Default**
  This is the TEST permission's label

Permission Test Reques...

Do you want to install this application?

(a) pre-existing permission (b) no permission

Figure 4.3: *Permission Test Requestor* installation

Experiment 2 In order to test my hypothesis — that permissions can only be displayed if they are already registered in the system — both apps where uninstalled and then the *Permission Test Requestor* app was re-installed first. During the installation, the permission authorisation prompt shown to the user was as in Figure 4.3b with no requested permissions displayed for authorisation. This was different to the its previous installation when it had been installed last. The *Permission Test Creator* app was then installed second, with its permission authorisation prompt displaying exactly as it had done before (Figure 4.2a).

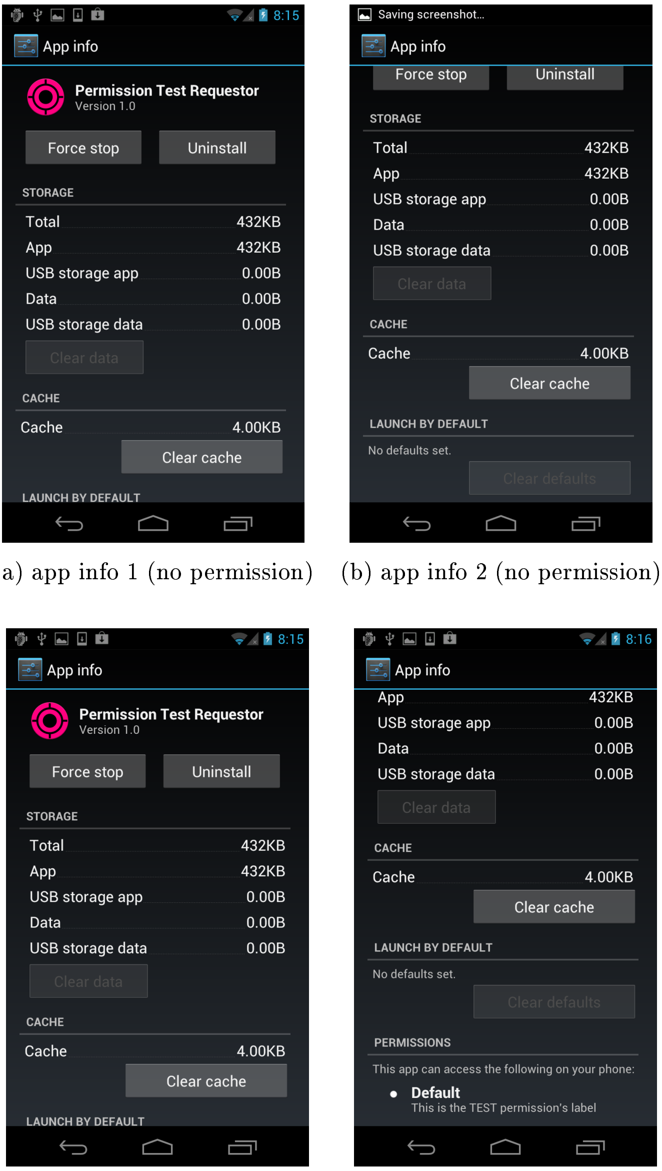
Aside from the proof of my hypothesis, of significant interest was the fact that having installed *Permission Test Creator*, the app info screen for *Permission Test Requestor* then showed the third-party permission granted, as in Figures 4.4c and 4.4d, compared to Figures 4.4a and 4.4b before the creator's installation. This `'dangerous'` permission seemed to have been granted, even though the user had never been prompted to authorise it.

4.2 Dormant Permission Requests Investigation

4.2.1 The Hypothesis

The observations from the previous section, when combined with the knowledge of the evolution of the permissions architecture from Chapter 3, raised a new and potentially far more sinister hypothesis. What if a malicious individual requested permissions within an app, to be installed on one version of the Android platform where those permissions do not yet exist, but enable sensitive activities on a future version of the Android platform? In other words, could a malicious app be created which includes dormant permissions requests, designed to make use of features of the Android platform which do not apply to the version of the platform the app is first installed onto, but which come alive at a later time, unbeknownst to the user, when the device's platform is upgraded?

Firstly, for such an app to be possible, the observations made in regard to a third-party permission, documented in the last section, would have to apply for newly defined manifest permissions



Total 432KB
App 432KB
USB storage app 0.00B
Data 0.00B
USB storage data 0.00B
Cache 4.00KB

(a) app info 1 (no permission)

Total 432KB
App 432KB
USB storage app 0.00B
Data 0.00B
USB storage data 0.00B
Cache 4.00KB
No defaults set.

(b) app info 2 (no permission)

Total 432KB
App 432KB
USB storage app 0.00B
Data 0.00B
USB storage data 0.00B
Cache 4.00KB

(c) app info 1 (existing permission)

App 432KB
USB storage app 0.00B
Data 0.00B
USB storage data 0.00B
Cache 4.00KB
No defaults set.
This app can access the following on your phone:
- **Default**
  This is the TEST permission's label

(d) app info 2 (existing permission)

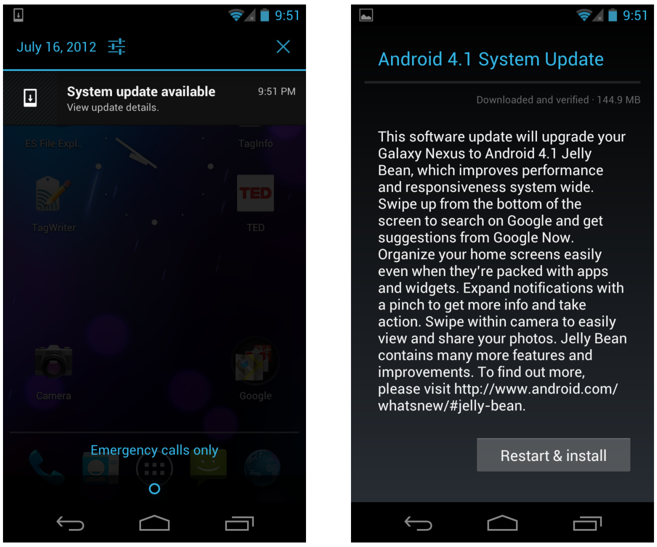
Figure 4.4: *Permission Test Requestor* app information

across platform upgrades. Secondly, for such an app to achieve a malicious end, some new permissions would need to be defined within a new version of the platform and these permissions would need to enable access to sensitive operations. Finally, for such an app to be possible to write, a malicious developer would need knowledge of those new permissions and be able to build them into an app in such a way that the app did not crash or otherwise raise undue attention when running on the older platform.

Requirement 3: Development Capability The third requirement actually involves the most common knowledge of the three (and this is the reason I mention it first). Developers already make use of the Android documentation and this documentation openly publicises the manifest permission list and differences between API versions [42, 7, 8, 9]. As well as this, the Android documentation and SDK are promptly updated and released before each new platform version reaches live devices, so that developers can begin tweaking and testing their apps for this new platform. Within this documentation is information on how to perform API detection in an app and with a simple call to ‘`Build.VERSION.SDK_INT`’ a developer can determine which Android API level an app is running on and tailor its actions accordingly.

Requirement 2: Permission Availability The second of these requirements is known to have already been met through the Permission Evolution investigation I performed in Chapter 3. Coincidentally, the latest version of Android, Jelly Bean, introduced a handful of new permissions, two of which, ‘`READ_CALL_LOG`’ and ‘`WRITE_CALL_LOG`’, are marked with a protection level of ‘`dangerous`’ and would likely be considered by many as controlling access to sensitive operations. Jelly Bean also introduced the ‘`READ_EXTERNAL_STORAGE`’ permission which sounds perfect for malicious activity as well and likely could be once the enforcement mechanism, which was discussed at the end of Subsection 3.2.3, comes out of test and into full use.

Requirement 1: Platform Behaviour In order to determine if the first requirement is met by the Android platform, such an app would need to be installed on a device and then a platform upgrade performed. Fortuitously, the Galaxy Nexus device used for my previous investigations had recently started prompting me to install a platform upgrade from Ice Cream Sandwich — Android 4.0.4 (API15) to Jelly Bean — Android 4.1 (API16) as shown in Figures 4.5a and 4.5b.



July 16, 2012

System update available 9:51 PM
View update details.

Android 4.1 System Update

Downloaded and verified · 144.9 MB

This software update will upgrade your Galaxy Nexus to Android 4.1 Jelly Bean, which improves performance and responsiveness system wide. Swipe up from the bottom of the screen to search on Google and get suggestions from Google Now. Organize your home screens easily even when they're packed with apps and widgets. Expand notifications with a pinch to get more info and take action. Swipe within camera to easily view and share your photos. Jelly Bean contains many more features and improvements. To find out more, please visit http://www.android.com/whatsnew/#jelly-bean.

(a) notification

(b) system update

Figure 4.5: Jelly Bean update on Ice Cream Sandwich

4.2.2 *Permission Test Jelly Bean*

In order to test this dormant permission request hypothesis I developed *Permission Test Jelly Bean*. This app requests three permissions within its ‘`AndroidManifest.xml`’ file — ‘`READ_CALL_LOG`’, ‘`WRITE_CALL_LOG`’ and ‘`READ_EXTERNAL_STORAGE`’ (each of which is new to Jelly Bean). Unlike my previous test apps which had no functionality other than displaying a ‘Hello World’ message on the screen, *Permission Test Jelly Bean* performs a series of operations when run (although these operations occur in the background with the only UI interaction being the ‘Hello World’ message as with my other test apps).

Firstly the app determines if it is running on API16 (Jelly Bean). If not, the app writes the message ‘ABC123: I’m not on Jelly Bean, I’m on SDK??, so I must be good - shhh, don’t draw attention to yourself!' to the system log and performs no further action (where SDK?? is the actual SDK version the app is running on). If the app identifies it is running on API16, however, it instead performs the following series of operations:

1. The app writes the message 'ABC123: I'm on Jelly Bean, so I can be bad - shhh, don't draw attention to yourself!' to the system log.
2. The app uses the '`READ_CALL_LOG`' permission to access the user's call log.
3. The app writes the phone numbers found in the call log to the system log.
4. The app uses the '`WRITE_CALL_LOG`' permission to write a new entry to the user's call log.
5. The app documents this new entry in the system log.
6. The app verifies whether external storage is present in the device.
7. The app uses the '`READ_EXTERNAL_STORAGE`' permission to read the folder and files in the root of the external storage (if present).
8. The app writes the names of any folders or files found into the system log.

Hopefully it should be clear that the functionality of the *Permission Test Jelly Bean* app meets the requirements for a malicious app as described in the hypothesis above. The app makes use of permissions unique to Jelly Bean, performs malicious functions only on the Jelly Bean (API16) platform and is designed to raise no visual indication of its behaviour to the user (other than that entailed in demonstrating use of the permissions in question - i.e. writing a false call log entry).

4.2.3 The Investigation

In preparation for testing the veracity of my hypothesis, the Galaxy Nexus device, still running Ice Cream Sandwich, was used to make and receive several calls so as to populate the call log. It then had files placed in the root of the external storage partition. These preparations were to replicate the usual circumstances of user devices and to enable the demonstration of the sensitive capabilities associated with the *Permission Test Jelly Bean* app's requested permissions.

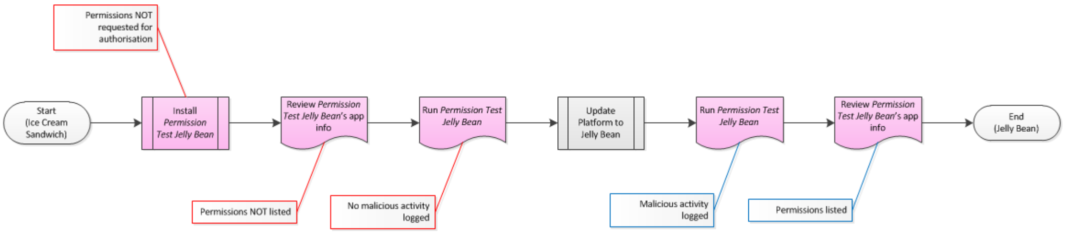
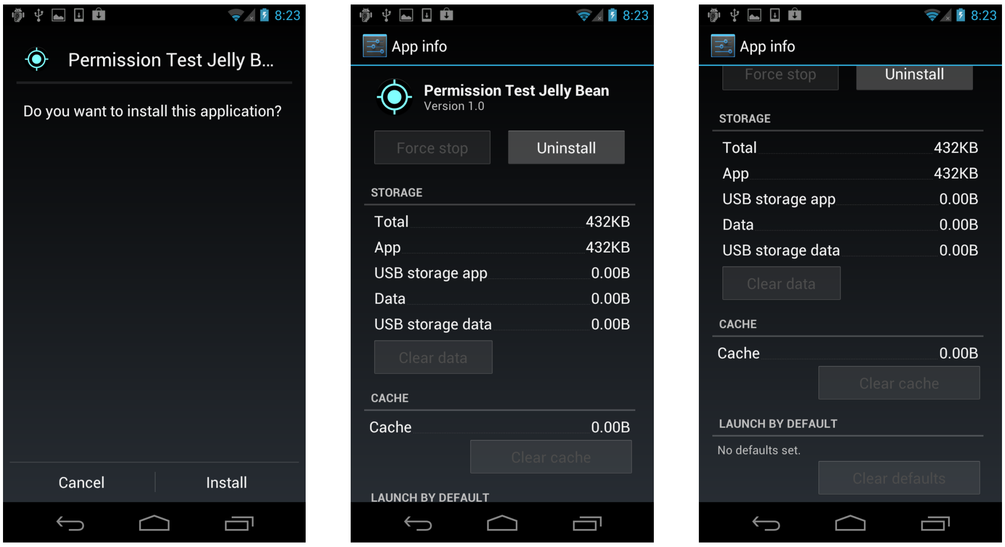

Permissions NOT requested for authorisation
Start (Ice Cream Sandwich)
Install *Permission Test Jelly Bean*
Review *Permission Test Jelly Bean's* app info
Run *Permission Test Jelly Bean*
Update Platform to Jelly Bean
Run *Permission Test Jelly Bean*
Review *Permission Test Jelly Bean's* app info
End (Jelly Bean)
Permissions NOT listed
No malicious activity logged
Malicious activity logged
Permissions listed

Figure 4.6: Dormant Permission Request Investigation

Figure 4.6 shows a high level workflow for the investigation which is documented in the text and figures of this section. To begin, the *Permission Test Jelly Bean* app was installed onto the Nexus device, a process which prompted for no permissions to be authorised and resulted in no permissions being granted (as shown in Figures 4.7a, 4.7b and 4.7c).

To demonstrate that the *Permission Test Jelly Bean* app was working correctly, it was run whilst the device was connected to the Android Debug Monitor (part of the Android SDK). The Android Debug Monitor allows the system log of a device to be easily monitored and as was expected from the functionality described in Subsection 4.2.2 above, the 'Hello World' message was displayed as per Figure 4.8a and the system log showed that a single entry had been written to it as shown in Figure 4.8b.

Now that the malicious test app was installed, the Jelly Bean update was triggered and the device was allowed to complete the update process normally. This update process ends with the device booting into the newly installed Jelly Bean version of the Android platform. Once the device restarted, the *Permission Test Jelly Bean* app was run with the device reconnected to the Android Debug Monitor for easy viewing of the system log. The app displayed the same screen



Permission Test Jelly B...

Do you want to install this application?

Permission Test Jelly Bean
Version 1.0

STORAGE

```
Total                 432KB
App                   432KB
USB storage app       0.00B
Data                  0.00B
USB storage data      0.00B
```

CACHE

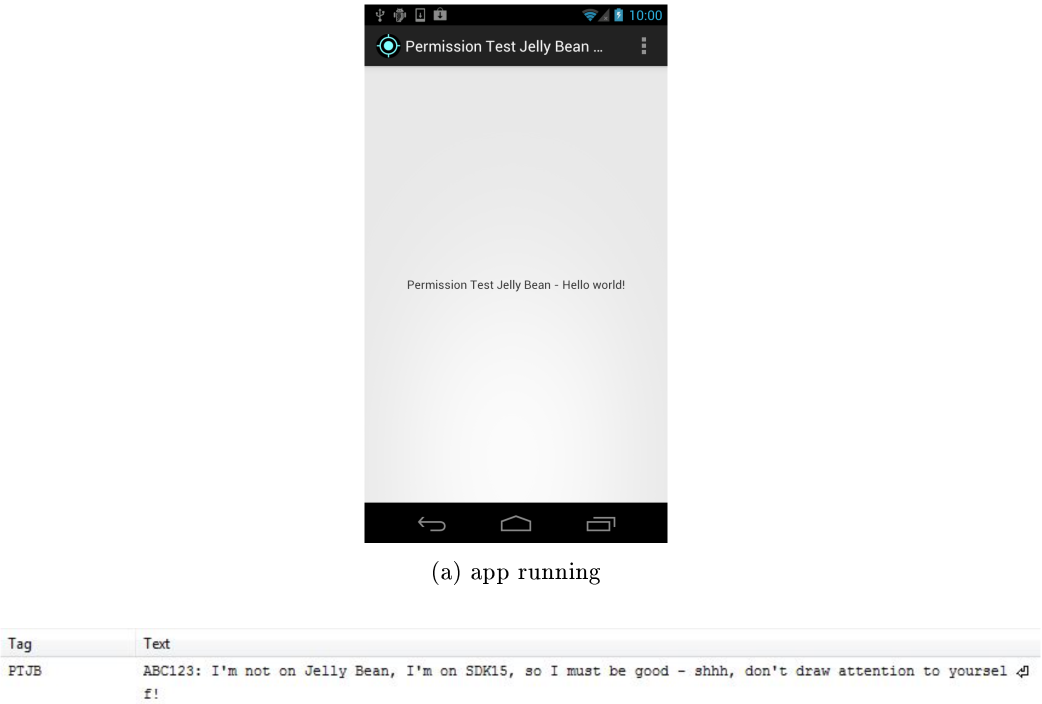
```
Cache                 0.00B
```

LAUNCH BY DEFAULT

No defaults set.

(a) installation (b) app info 1 (c) app info 2

Figure 4.7: *Permission Test Jelly Bean* on Ice Cream Sandwich



Permission Test Jelly Bean - Hello world!

(a) app running

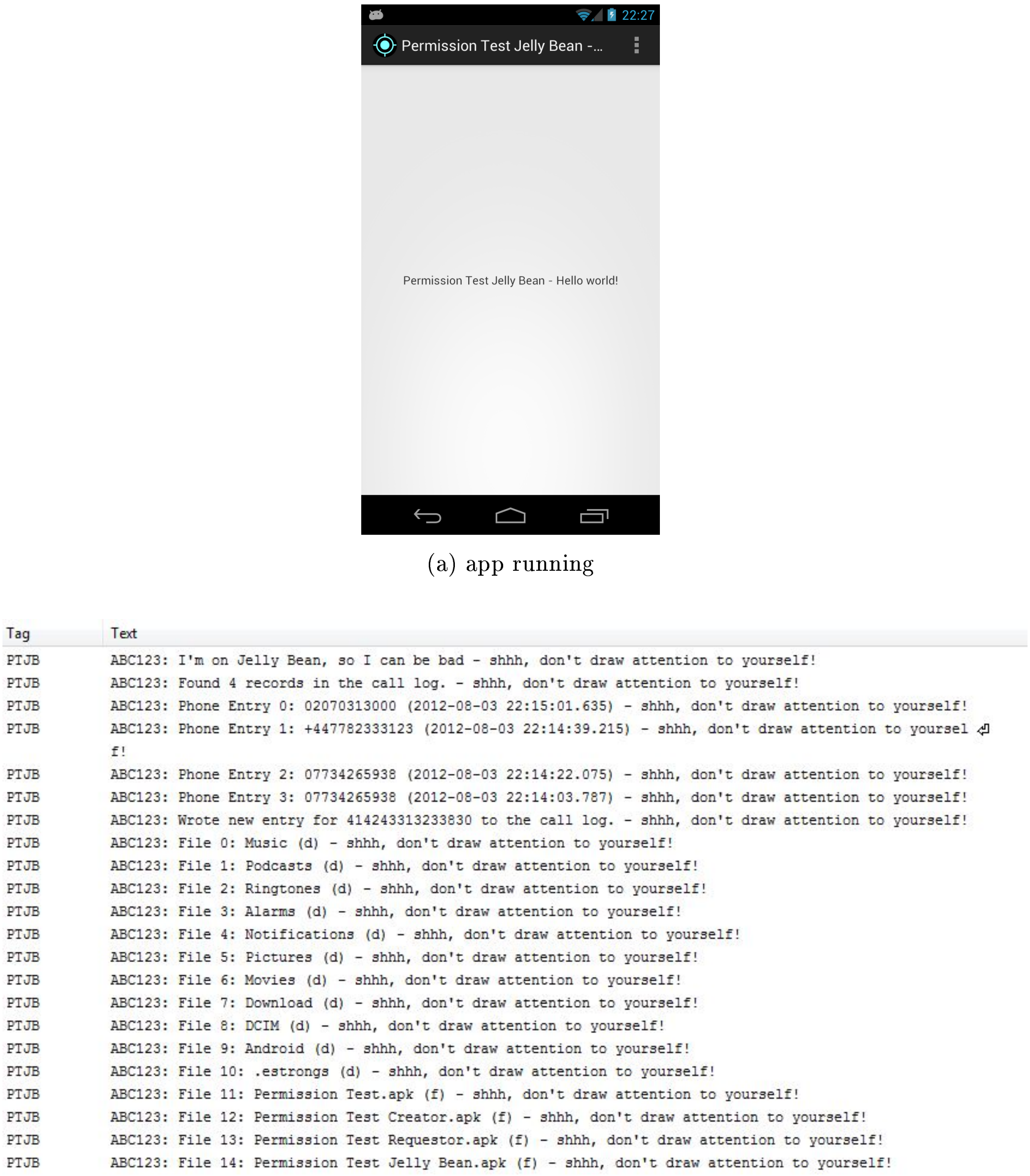
```
Tag        Text
PTJB       ABC123: I'm not on Jelly Bean, I'm on SDK15, so I must be good - shhh, don't draw attention to yoursel ⏎
           f!
```

(b) system log entry

Figure 4.8: *Permission Test Jelly Bean* running on Ice Cream Sandwich

as before (compare Figure 4.9a and Figure 4.8a) but this time, on Jelly Bean, the app was able to perform its malicious activities which it logged as shown in Figure 4.9b.

You can see from the system log shown in Figure 4.9b that the app identified and read four entries from the call log and wrote an additional random entry back to it. The app also found 11 directories within the root of the external storage, each marked with a (d), and four files marked



Permission Test Jelly Bean - Hello world!

(a) app running

```
Tag         Text
PTJB        ABC123: I'm on Jelly Bean, so I can be bad - shhh, don't draw attention to yourself!
PTJB        ABC123: Found 4 records in the call log. - shhh, don't draw attention to yourself!
PTJB        ABC123: Phone Entry 0: 02070313000 (2012-08-03 22:15:01.635) - shhh, don't draw attention to yourself!
PTJB        ABC123: Phone Entry 1: +447782333123 (2012-08-03 22:14:39.215) - shhh, don't draw attention to yoursel ⏎
            f!
PTJB        ABC123: Phone Entry 2: 07734265938 (2012-08-03 22:14:22.075) - shhh, don't draw attention to yourself!
PTJB        ABC123: Phone Entry 3: 07734265938 (2012-08-03 22:14:03.787) - shhh, don't draw attention to yourself!
PTJB        ABC123: Wrote new entry for 414243313233830 to the call log. - shhh, don't draw attention to yourself!
PTJB        ABC123: File 0: Music (d) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 1: Podcasts (d) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 2: Ringtones (d) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 3: Alarms (d) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 4: Notifications (d) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 5: Pictures (d) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 6: Movies (d) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 7: Download (d) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 8: DCIM (d) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 9: Android (d) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 10: .estrongs (d) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 11: Permission Test.apk (f) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 12: Permission Test Creator.apk (f) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 13: Permission Test Requestor.apk (f) - shhh, don't draw attention to yourself!
PTJB        ABC123: File 14: Permission Test Jelly Bean.apk (f) - shhh, don't draw attention to yourself!
```

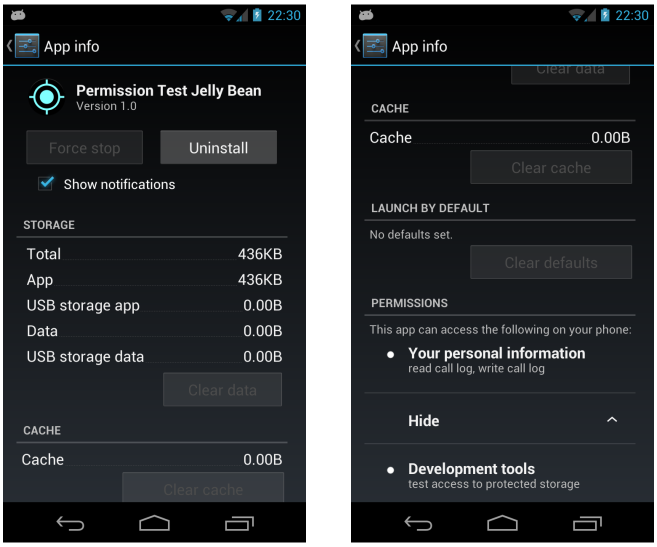
(b) system log entries

Figure 4.9: *Permission Test Jelly Bean* running on Jelly Bean

with (f).

At no time, from the point at which the *Permission Test Jelly Bean* app installation was started on Ice Cream Sandwich, to the point when the app collected this data on Jelly Bean, was any prompt made to the user to authorise the permissions which were used. The permissions had not existed within the Ice Cream Sandwich platform during app installation and so had not been prompted to the user during the pre-installation permission authorisation process (as shown in Figure 4.7a). During the platform update process these permissions had been introduced into the system and thus granted to the already installed app without user interaction or notification. The app information clearly shows the permissions as granted on Jelly Bean, as per Figures 4.10a and 4.10b.

This investigation has shown that each of the three requirements for my dormant permission



☑ Show notifications

STORAGE

Total	436KB
App	436KB
USB storage app	0.00B
Data	0.00B
USB storage data	0.00B

CACHE

Cache	0.00B

(a) app info 1

CACHE

Cache	0.00B

LAUNCH BY DEFAULT

No defaults set.

PERMISSIONS

This app can access the following on your phone:

- **Your personal information**
  read call log, write call log

- **Development tools**
  test access to protected storage

(b) app info 2

Figure 4.10: *Permission Test Jelly Bean* app information on Jelly Bean

request hypothesis (platform behaviour, permission availability and development capability) was achievable and that a test app could easily be developed to demonstrate the exploit of such dormant permission requests across an Android platform update boundary.

I have attempted, with considerable determination, to identify any existing knowledge of this issue within the Android developer or security researcher communities. I have been unable to find reference to it within any of the many Android security papers I have located and read, a list not limited to just those in this report's bibliography. I have also found no mention of it on popular Android forums through Google searches. A search of the Android Open Source Project's bugs list identified 363 issues mentioning 'permission', none of which seemed to discuss this issue [37]. After much searching I have come across one single related reference, in a comment buried within the Android platform's source code itself. The file '`PackageManagerService.java`' within the '`frameworks/base/services/java/com/android/server/pm/`' section of the code includes a comment on line 1189 which states:

> "// If the platform SDK has changed since the last time we booted,
> // we need to re-grant app permission to catch any new ones that
> // appear. This is really a hack, and means that apps can in some
> // cases get permissions that the user didn't initially explicitly
> // allow... it would be nice to have some better way to handle
> // this situation." [26]

Assuming this code comment is referencing the issue I have highlighted, which I believe it does, it would seem that at some point a developer foresaw some potential issue with the platform's permission processing — whether they envisaged quite the exploit I have now demonstrated I do not know.

After researching the history of the '`.../pm/PackageManagerService.java`' file (parent folder and filename specified for clarity) and specifically the comment quoted above, the following points were identified as significant. The file was originally created on the 22nd of March 2011 when another file, with the same name but located one folder up in the hierarchy, was moved into the '`frameworks/base/services/java/com/android/server/pm/`' folder. At the time of this move, the comment already existed in the source code. The '`.../server/PackageManagerService.java`' file (which was later moved to become '`.../pm/PackageManagerService.java`') was modified to introduce the comment on the 6th of April 2010, during a commit documented with the commit message:

> "* Fix issue #2569139: Cannot login to last.fm after upgrade from Donut to FRF01B
>
> This is a quick and dirty solution to re-assign permissions after booting from a platform update. It is not great, because it means that an app can have permissions that the user didn't get to see when they originally installed it. Unfortunately it's not clear what else to do here, nor is there time to do anything significant." [26]

Whilst I have been unable to identify the issue mentioned — the issue number (#2569139) doesn't match any bugs listed within the Android User Issue List [38] — this commit message, and the code change itself, identify there was a need to introduce permission re-evaluation, post platform update, to resolve it. The date of this commit would suggest that this change may have been implemented during a final development push towards the release of Froyo, Revision 1 (Android 2.2 - May 2010) [46]. This might explain why the issue number does not match any from the public issues list, in that it may have been identified in internal testing, and also why there was a need for a "quick and dirty solution" [26]. Of course in both these points I am purely speculating and only those involved would be able to confirm the exact situation.

4.2.4 Repercussions

The investigation performed in this chapter has successfully tested a hypothesis developed out of the previous investigations discussed within this report. The hypothesis — that dormant permission requests can be exploited, across an Android update boundary, to gain permission to sensitive operations without the user authorising them — was proved through the development and use of a test app exploiting three permissions newly introduced to Jelly Bean.

From the observations made throughout these investigations, the cause of this issue appears to be the combination of a number of factors. Firstly, the pre-installation permission authorisation prompt only displays permissions which are requested by the app AND currently defined within the system. This is noticeable both with the requesting of third-party permissions, as well as permissions undefined in the Android platform version the app is being installed onto. Secondly, the permissions granted to an app are initially evaluated during installation but are also re-evaluated during post-update boot processing of the Package Manager service.

When considering the case of platform updates and newly introduced manifest permissions, these factors result in an easily exploitable weakness in the Android permissions architecture. This issue must be considered a weakness due to the fact that `'dangerous'` permissions, which usually require authorisation by the user, can be granted without the user seeing them being requested at the time they authorise the app's installation. These permissions require user authorisation due to their sensitivity, as highlighted in the Android documentation which describes the `'dangerous'` protection level as:

> "A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user. Because this type of permission introduces potential risk, the system may not automatically grant it to the requesting application. For example, any dangerous permissions requested by an application may be displayed to the user and require confirmation before proceeding, or some other approach may be taken to avoid the user automatically allowing the use of such facilities." [48]

The Permission Evolution investigation performed in Chapter 3 highlights the fact that there are numerous platform upgrade opportunities and also that new manifest permissions are introduced with some of those new platform versions. Both situations add to the opportunities for the weakness to be exploited. The potential impact of any such exploit, depends completely on the capabilities protected by any newly introduced manifest permission. At one extreme, if no new permissions are ever introduced, then this weakness is unable to be exploited. This does not seem likely considering the platform history already discussed. If any new permissions only restrict access to capabilities which are not considered sensitive, then whilst the weakness could be exploited the resulting impact will likely be considered low. If, as in the case with several of the permissions tested here, any new permissions control access to sensitive capabilities, then depending on those capabilities and the information held on the user's device, the impact may be high.

The ultimate repercussion of this weakness, is therefore that malicious apps can be easily crafted to make use of an unobserved permission escalation at the time of platform update. Whilst my test app, *Permission Test Jelly Bean*, had no function separate to the exploit, a malicious app could be developed as a logic bomb with non-malicious functionality masking its true nature. For example, a malicious app could be constructed as a weather app, and could legitimately request the `'INTERNET'` and `'ACCESS_COARSE_LOCATION'` permissions in order to retrieve weather information for the user's location (two permissions commonly requested by weather apps and widgets [60]). These two permissions would allow the app access to network connectivity and approximate location information and could be used by the app, whilst running benignly, to provide weather information. During the installation it is unlikely that many users would reject such permission requests. Should the user update their device platform, which for many would simply entail clicking an install button such as that shown in Figure 4.5b, then the malicious app's dormant permission requests would become active. The app could then use those permissions to access sensitive information, assuming applicable new permissions, and could then misuse the two legitimate permissions to send that information along with the user's location over the Internet to the app's developer. It would only be through reviewing the app's app info screen after the update, and thus viewing the currently granted permissions, that the user would have any chance of becoming aware of the new capabilities available to the app. Whilst this review would enable the user to determine the discrepancy and uninstall the app, the likelihood that users regularly perform those checks at this time is, I believe, extremely low. I also believe that many users would fail to identify the new permissions as such and would most likely assume they had been present all the time, therefore being less likely to take any corrective action. Whilst research has been performed into user understanding and awareness of permissions [89, 97], at this time I am not aware of any research being performed into user's recollection of what permissions an app has (without reviewing) nor what an app should have (from a blank slate). These are appropriate areas for future research considering the weakness identified here.

4.2.5 Potential Mitigation

There are a number of ways in which the weakness associated with dormant permission requests could be mitigated. One of the original contributing factors is the fact that permissions are re-evaluated by the Package Manager post update installation, potentially resulting in the granting of additional permissions. One mitigation would be to remove this re-evaluation, although that seems likely to be problematic considering the mechanism seems to have been introduced to fix an issue related to platform updates (discussed at the end of Section 4.2.3).

Assuming that permission re-evaluation is necessary and bearing in mind that there is already a clear definition of the protection levels — which mandates user involvement in authorising permissions rated as `'dangerous'` [48] — it would seem appropriate to request authorisation from users for newly granted `'dangerous'` permissions, the first time an app is run following the re-evaluation.

There are bound to be numerous factors that should be considered before adopting such a strategy. The Android platform, whilst being an open source project, is still a product and no doubt has a defined direction and plan with regards to features, performance and usability. The addition of a security related message with which the user must interact can obviously be seen to reduce the usability of the product, especially if we believe that many users fail to take notice of such prompts anyway [41] and may not understand them at all [89]. Whilst that may be true, the automatic and silent granting of such permissions — especially with the potential impact discussed in the last section — puts the user at risk without them having any positive awareness of the change.

Such a post-installation permission authorisation prompt could be claimed should only need to show the permissions which have newly been requested. I believe, however, that a better strategy would be to highlight any newly requested permissions alongside any already granted ones. After all it is the entire set of resultant permissions which will determine the capabilities the app can make use of. As with the pre-installation permission authorisation prompt, any permissions rated as `'normal'` could be hidden by default. This does not conflict with the definition of that protection level.

An alternative to the above strategy could be to provide the user with a notification indicating a change in permissions for the applicable app and a direct method of accessing the relevant app's app info screen (here I use the term notification in the sense of the so-named Android UI component [57]). This could be considered a more subtle method of providing user awareness, although I cannot help but feel that it is less likely to gain mental traction with the user and as such would have limited benefit. It would also involve the automatic granting of the permissions, with the notification acting only as a warning to the user, who would have to then take action should they so desire. On the other hand, this method would have little impact on usability, providing no real interruption to the user experience. I personally feel this alternative is the minimum necessary improvement although I much rather that user control be re-established through the use of a post-installation permission authorisation prompt following post-update permission re-evaluation.

Chapter 5

Conclusion

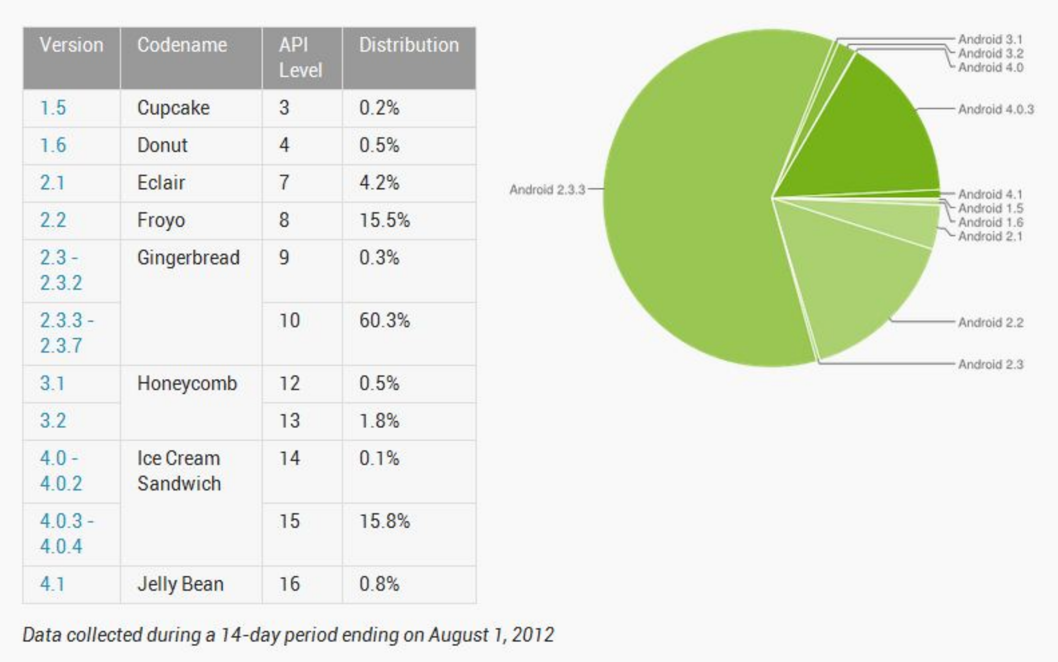
The specific goal of this report has been to document my investigations, the detailed knowledge that has resulted and the associated findings of hypotheses based on that knowledge. I have been fortunate to identify a weakness in the Android permissions architecture and even more fortunate to be able to successfully design and test my hypothesis as to how it may be exploited. The details of this exploit formed the later part of this report (Chapter 4) and much of the energy I have expended after my initial background research.

The smartphone device market has seen rapid growth over recent years, in no small part thanks to the introduction and popularity of centralised app stores and easily downloadable apps. The Android operating system is a prominent player in this market with a continuing growth both in numbers of devices and numbers of downloaded apps. The wide spread of versions of the Android platform lead to a diverse ecosystem of devices which constantly evolves. Figure 5.1 shows the relative distribution of Android platform versions as of the 1st of August 2012 and when compared to Figure 3.1 highlights the rate of change of this ecosystem. In the one month between these two data sets, Jelly Bean has been seen on 0.8% of devices whilst the number of Ice Cream Sandwich devices has risen from 10.9% (covering API14 and API15) to 15.9%. This growth has been made at the expense of Gingerbread (losing 3.4%) and Froyo (losing 1.8%) principally, with Eclair and Honeycomb taking far smaller loses of 0.5% and 0.1% respectively.

Whilst it is clear to see that devices running older platform versions are still frequently used, across this wide ecosystem platform updates do also frequently occur. A weakness in the Android permissions architecture which can exploited across the platform update boundary is therefore a significant concern, especially when the creation of such an exploit can be achieved by using nothing more than the same methods used to write any other Android app. It is just such a weakness, its investigation and demonstration, that is the most significant contribution of this report.

This report has also detailed the categorisation of the Android permissions architecture and highlighted several points of disagreement when compared to the official documentation. Firstly, the experimental evidence has shown the presence of the permissions '`READ_USER_DICTIONARY`' and '`WRITE_USER_DICTIONARY`' through versions of the Android platform upon which they supposedly do not exist. Secondly, the disparity between the documentation, which indicates five permission additions in API14 and a further two in API15, compared to my experimental results whereby all seven permission additions are seen in API14. Both these discrepancies were initially discussed in Section 3.2.2.

It is further hoped that the information identified regarding the evolution of the Android permissions in general, and their current breakdown specifically, will be of use to developers when making permission-related decisions. The tables in Figures 3.4, 3.6, 3.7, 3.8, 3.9, 3.10 and 3.11 all contain valuable information which is currently missing from the official Android documentation.



Version	Codename	API Level	Distribution
1.5	Cupcake	3	0.2%
1.6	Donut	4	0.5%
2.1	Eclair	7	4.2%
2.2	Froyo	8	15.5%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	60.3%
3.1	Honeycomb	12	0.5%
3.2		13	1.8%
4.0 - 4.0.2	Ice Cream Sandwich	14	0.1%
4.0.3 - 4.0.4		15	15.8%
4.1	Jelly Bean	16	0.8%

Data collected during a 14-day period ending on August 1, 2012

Figure 5.1: Android platform versions (as at 01/08/2012)

5.1 Related Work

As has been highlighted several times in this report, the Android platform and the permissions architecture specifically, has seen significant previous research. The Android platform and its security mechanisms have been considered as a whole in [100] and [101], with [90] and [93] focusing more on the hardware-based security side. Application security is well covered across research such as [78], [85], [81], [88] and [77] which place considerable focus on inter-app communication, information leakage and misuse. The concept of inter-app communication is considered further in [71], [75] and [74] which look at concepts core to the confused deputy problem. App stores and their role in the apps security and malicious app distribution has been covered in [110] and [108].

The topic of permission architectures specifically has been covered from a variety of angles. I list some of the more relevant examples here. Of these works, several have included surveys of the Android permissions through either static or dynamic analysis techniques. Whilst not covering exactly the same content as my Permission Categorisation and Evolution investigations, these works are closely related and complimentary. My investigation of Dormant Permission Requests is completely unique however, having not been covered in any of the related work current known to me.

'Short Paper: A Look at SmartPhone Permission Models' Au et al looked at the permission architectures of various smartphone platforms and Android in particular. They identified goals for security researchers and highlighted the overprivilege issue and stated the number of permissions added in early (up to 2.3) versions of Android [66].

'A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android' Barrera et al analysed the number of permissions requested across the 22 categories in the Android Market (now Google Play) as well as performing Component Plane Analysis of the permissions recorded [67].

'Android Permissions Demystified' Porter Felt et al modified the permission verification mechanism of Android 2.2 in order to identify which API calls result in permission checks. They then built 'Stowaway', a tool for static analysis of apps, and used it to determine permission errors (particularly overprivilege) within the apps tested [94].

'Curbing Android Permission Creep' Vidas et al also analysed apps to identify permission errors, highlighting duplicate permissions and overprivilege in particular [107].

'Is this App Safe? A Large Scale Study of Application Permissions And Risk Signals' Chia et al looked at apps developed for a range of platforms including Android apps. They focused on comparing the permissions requested by popular and newly added apps as well as considering the effectiveness of risk signals available to users [70].

'A Small but Non-negligible Flaw in the Android Permission Scheme' Shin et al identify a means of abusing third-party permission requests through the fact that permission strings can be easily copied and are the only identifying requirement for uniqueness [102].

'A Conundrum of Permissions: Installing Applications on an Android Smartphone' Kelly et al performed a user study to determine if users understand the permissions related security messages they are prompted with and to see what factors users use to select apps [89].

'Android Permissions: User Attention, Comprehension, and Behavior' Porter Felt et al also performed a user study regarding permission authorisation performance and comprehension [97].

'How to Ask For Permission' Porter Felt et al compared several mechanisms through which permission systems interact with users and proposed guidelines for platform designers with regards to the selection of the most appropriate mechanism for a specific permission [95].

'Choice Architecture and Smartphone Privacy: There's A Price for That' Egelman et al performed a user study to identify user willingness to pay for privacy as represented through fewer permissions being requested during app installation [76].

'I've Got 99 Problems, But Vibration Ain't One: A Survery of Smartphone Users' Concerns' Porter Felt et al performed a user study to identify what permission related security risks upset users and to what extent [96].

5.2 Ongoing and Future Work

The investigations within this report have each raised new questions and more ideas with regards to future areas of work. The investigations themselves may benefit from some refinement and repetition across retail devices, something I have already started to do. There are a number of specific investigations which follow on from the work in this report. Some have been mentioned in passing during the previous discussions.

5.2.1 Third-Party Permission Requests Revisited

Whilst the *Permission Test Creator* and *Permission Test Requestor* test apps used in Section 4.1 defined and requested permissions as previously discussed, they were coded as variants of the original *Permission Test* app. As in its case, they did not make use of the permissions they requested and were granted. At the time this simplification was of no concern, but an investigation

I later carried out suggested that my previous observations may have been misleading. The investigation in question was designed to monitor changes in the `'/data/system/packages.xml'` file, mentioned previously in Section 3.1.2, during the process of the third-party permission investigation's app installations. To do this investigation, the Galaxy Nexus device was once again 'rooted' enabling retrieval of the desired file.

The notable observation during this investigation was that when *Permission Test Creator* was installed after *Permission Test Requestor*, whilst the app info screen reflected that shown in Figures 4.4c and 4.4d, the `'/data/system/packages.xml'` file did not equally reflect the granting of the third-party permission to *Permission Test Requestor*. This caused me to question the information shown by these two dissenting sources, with the only means of resolution being to create versions of *Permission Test Creator* and *Permission Test Requestor* which actually attempt to employ the permissions concerned.

I therefore created *Permission Test Creator 3* and *Permission Test Requestor 3* (the v2 apps had already been created to further my understanding of Android's IPC and Intents separate to this report). I performed the same experiment with *Permission Test Requestor 3* installed prior to *Permission Test Creator 3*. With these two test apps I was able to confirm that whilst *Permission Test Requestor 3's* app info screen displayed the third-party permission, the actual app was unable to perform the operation which was restricted by that permission — in line with the content of the `'/data/system/packages.xml'` file. This outcome shows that there is no general re-evaluation of permissions following the declaration of a new permission. It also highlights the presence of an app installation order requirement amongst apps which declare and utilise third-party permissions, counter to that perceived in the third-party permission investigation. Finally, this result also pulled into question the information displayed on the app info screen. Whilst this screen showed a third-party permission for the *Permission Test Requestor 3* app, in reality it was not granted within the platform.

5.2.2 Other Work

Documentation Discrepancies In order to confirm the discrepancies identified in the documentation regarding permissions, the *Permission Test* app should be used to review the permissions of numerous retail devices across all versions of the Android platform. I have already begun this process with devices running Android 2.2 (API8), Android 2.3.5 (API10), Android 3.2 (API13), Android 4.0.3 (API15) and Android 4.0.4 (API15). So far the results seem to confirm those in this report but a complete analysis is required to confirm the discrepancies fully.

/data/system/packages.xml As has already been highlighted, this system file documents the permissions currently defined within the system and those granted to apps. An initial review of this file suggests there may be more than the currently documented 130 system permissions. I have already begun comparing the two lists and any permissions not documented in the manifest permissions documentation list will require testing and categorisation.

System Definitions Separate to the `'/data/system/packages.xml'` file it is not currently clear to me where the cross-referencing of permissions, permission labels, protection levels and groupings is performed. I assume this is somewhere in the source code, but as yet I have not completed a comprehensive review. When this cross-referencing is identified, the Permission Evolution investigation results can be extended to include permission labels and groupings for those permissions not defined as protection level `'normal'` or `'dangerous'`.

User Studies In order to understand user's interactions with the app info screen as it directly relates to the dormant permission requests weakness, user studies are required to determine if users perform regular or irregular permission reviews post installation and to determine if users show any awareness to changes in the permissions listed on that screen.

Each of these activities clearly follows on from investigations performed in this report, with many of the same techniques able to be directly applied, save for the case of user studies. It is therefore expected that whilst new knowledge may be gained from these future works, little new knowledge should be required to complete them. I therefore hope to report on these matters in the not too distant future, the results of which should go a long way to filling the few small gaps left from the investigations performed so far.

Bibliography

[1] 10 Billion Android Market downloads and counting. http://googleblog.blogspot.co.uk/2011/12/10-billion-android-market-downloads-and.html. [online] Accessed On: 2012-04-02.

[2] A chat with the man behind mobiles. http://news.bbc.co.uk/1/hi/uk/2963619.stm. [online] Accessed On: 2012-03-25.

[3] About HTC. http://www.htc.com/uk/about/#mission. [online] Accessed On: 2012-03-29.

[4] About Research in Motion. http://us.blackberry.com/company.jsp. [online] Accessed On: 2012-03-29.

[5] Android and Security - Official Google Mobile Blog. http://googlemobile.blogspot.com/2012/02/android-and-security.html. [online] Accessed On: 2012-07-16.

[6] Android@Mobile World Congress: It's all about the ecosystem. http://googlemobile.blogspot.co.uk/2012/02/androidmobile-world-congress-its-all.html. [online] Accessed On: 2012-04-02.

[7] API Differences between 13 and 14. http://developer.android.com/sdk/api_diff/14/changes.html. [online] Accessed On: 2012-06-28.

[8] API Differences between 14 and 15. http://developer.android.com/sdk/api_diff/15/changes.html. [online] Accessed On: 2012-06-28.

[9] API Differences between 15 and 16. http://developer.android.com/sdk/api_diff/16/changes.html. [online] Accessed On: 2012-06-28.

[10] App Store Approval Process. https://developer.apple.com/appstore/resources/approval/index.html. [online] Accessed On: 2012-07-28.

[11] App Store Review Guidelines. https://developer.apple.com/appstore/resources/approval/guidelines.html. [online] Accessed On: 2012-07-28.

[12] App Store Submission Tips. https://developer.apple.com/appstore/resources/submission/tips.html. [online] Accessed On: 2012-07-28.

[13] Apple Introduces the New iPhone 3G. http://www.apple.com/pr/library/2008/06/09Apple-Introduces-the-New-iPhone-3G.html. [online] Accessed On: 2012-03-29.

[14] Apple Investor Relations - FAQs. http://investor.apple.com/faq.cfm?FaqSetID=6. [online] Accessed On: 2012-03-29.

[15] Apple iPad Technical Specifications. http://www.apple.com/uk/ipad/specs/. [online] Accessed On: 2012-04-01.

[16] Apple Press Info - iPhone. http://www.apple.com/pr/products/iphone/iphone.html. [online] Accessed On: 2012-03-29.

[17] Apple Requires User Permission Before Apps Can Access Personal Data in iOS 6. http://www.macrumors.com/2012/06/14/apple-requires-user-permission-before-apps-can-access-personal-data-in-ios-6/. [online] Accessed On: 2012-07-14.

[18] Apple's App Store Downloads Top 25 Billion. http://www.apple.com/pr/library/2012/03/05Apples-App-Store-Downloads-Top-25-Billion.html. [online] Accessed On: 2012-04-02.

[19] Bell Labs Historical Timeline. http://www.alcatel-lucent.com/wps/portal/!ut/p/kcxml/04_Sj9SPykssy0xPLMnMz0vM0Y_QjzKLt4z3DADJmMU7xhu5mupHIgsZxDvCBXw98nNT9YOAEpHmQLWm7t76IfqRbvre-gEgk2Bi4UWpeSmpRfoFuaER5X5pHrm0iooAM-c3Cw!!/delta/base64xml/L0lNN3VhQ1NXWUEhIS9JTmhBQ0lpRWlBaU13cUFBd0FxZ0FNQUEvNExFNVJPZ3JnSUEhLzdfQV81R0s!?decade=1940s&innovation=History%2FTimeline%2FTimeline_Innovation_000078.jsp#7_A_5GK. [online] Accessed On: 2012-03-25.

[20] Brief History of GSM & the GSMA. http://www.gsma.com/history/. [online] Accessed On: 2012-03-29.

[21] Choosing an iOS Developer Program. https://developer.apple.com/programs/start/ios/. [online] Accessed On: 2012-07-28.

[22] Cydia. http://cydia.saurik.com/. [online] Accessed On: 2012-07-28.

[23] Designing for Security | Android Developers. http://developer.android.com/guide/practices/security.html. [online] Accessed On: 2012-07-03.

[24] Eight Ways to Keep your Smartphone Safe. http://www.bullguard.com/bullguard-security-center/mobile-security/mobile-protection-resources/8-ways-to-keep-your-smartphone-safe.aspx. [online] Accessed On: 2012-06-23.

[25] "feature phone" - Oxford Dictionaries. http://oxforddictionaries.com/definition/feature%2Bphone. [online] Accessed On: 2012-04-01.

[26] frameworks/base/services/java/com/android/server/pm/PackageManagerService.java. http://source.android.com/. Android Open Source Project - Source Code.

[27] Get Started with Publishing | Android Developers. http://developer.android.com/distribute/googleplay/publish/register.html. [online] Accessed On: 2012-07-16.

[28] Google Wallet Security: PIN Exposure Vulnerability. http://zvelo.com/blog/entry/google-wallet-security-pin-exposure-vulnerability. [online] Accessed On: 2012-06-23.

[29] Guglielmo Marconi Biography. http://www.nobelprize.org/nobel_prizes/physics/laureates/1909/marconi-bio.html. [online] Accessed On: 2012-03-12.

[30] History - Ericsson. http://www.ericsson.com/uk/thecompany/company_facts/history. [online] Accessed On: 2012-03-29.

[31] How Are Smartphones Being Used? [Infographic]. http://www.tatango.com/blog/how-are-smartphones-being-used/. [online] Accessed On: 2012-06-04.

[32] How People Use Smartphones [INFOGRAPHIC]. http://60secondmarketer.com/blog/2011/08/16/how-people-use-smartphones-infographic/. [online] Accessed On: 2012-06-04.

[33] IBM 608 Calculator. http://www-03.ibm.com/ibm/history/exhibits/vintage/vintage_4506VV2214.html. [online] Accessed On: 2012-03-25.

[34] iOS 5 Security Flaw Allows Access To Contacts List, Recent Calls & Text Messages Without Passcode. http://www.cultofmac.com/147700/ios-5-security-flaw-allows-access-to-contacts-list-recent-calls-text-messages-without-passcode/. [online] Accessed On: 2012-06-23.

[35] iOS Bug Unlocks iPhone sans Password. http://www.theregister.co.uk/2010/10/26/iphone_password_bypass/. [online] Accessed On: 2012-06-23.

[36] iOS Human Interface Guidelines. https://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html. [online] Accessed On: 2012-07-28.

[37] Issues - android - Android - An Open Handset Alliance Project - Google Project Hosting. https://code.google.com/p/android/issues/list?can=2&q=permission&colspec=ID%20Type%20Status%20Owner%20Summary%20Stars&groupby=&sort=&num=500&start=0. [online] Accessed On: 2012-08-03.

[38] Issues - android - Android - An Open Handset Alliance Project - Google Project Hosting. http://code.google.com/p/android/issues/list. [online] Accessed On: 2012-08-05.

[39] ITRC Fact Sheet 144 - Smartphone Safety. http://www.idtheftcenter.org/artman2/publish/v_fact_sheets/ITRC_Fact_Sheet_144.shtml. [online] Accessed On: 2012-06-23.

[40] Key Global Telecom Indicators for the World Telecommunication Service Sector. http://www.itu.int/ITU-D/ict/statistics/at_glance/KeyTelecom.html. [online] Accessed On: 2012-03-29.

[41] Late-Night Poll: Do You Read App Permissions Before Installing? http://www.androidcentral.com/late-night-poll-do-you-read-app-permissions-installing. [online] Accessed On: 2012-07-14.

[42] Manifest.permission | Android Developers. http://developer.android.com/reference/android/Manifest.permission.html. [online] Accessed On: 2012-07-10.

[43] Mobile Cellular Subscriptions per 100 Inhabitants 2011. http://www.itu.int/ITU-D/ict/statistics/material/excel/2011/Mobile_Cellular_reg-11.xls. [online] Accessed On: 2012-03-29.

[44] <permission> | Android Developers. http://developer.android.com/guide/topics/manifest/permission-element.html. [online] Accessed On: 2012-07-11.

[45] Permissions | Android Developers. http://developer.android.com/guide/topics/security/permissions.html. [online] Accessed On: 2012-07-03.

[46] Platforms | Android Developers. http://developer.android.com/tools/revisions/platforms.html. [online] Accessed On: 2012-08-05.

[47] Protect Mobile Phones. http://www.getsafeonline.org/nqcontent.cfm?a_id=1850. [online] Accessed On: 2012-06-23.

[48] R.attr | Android Developers. http://developer.android.com/reference/android/R.attr.html#protectionLevel. [online] Accessed On: 2012-07-14.

[49] Researchers Find Methods for Bypassing Google's Bouncer Android Security. http://threatpost.com/en_us/blogs/researchers-find-methods-bypassing-googles-bouncer-android-security-060412. [online] Accessed On: 2012-07-16.

[50] Samsung's History. http://www.samsung.com/uk/aboutsamsung/corporateprofile/history03.html. [online] Accessed On: 2012-03-29.

[51] ‘smartphone’ - Collins. http://www.collinsdictionary.com/dictionary/english/smartphone. [online] Accessed On: 2012-04-01.

[52] ’smartphone’ - Dictionary.com. http://dictionary.reference.com/browse/smartphone. [online] Accessed On: 2012-04-01.

[53] ‘smartphone’ - Merriam Webster. http://www.merriam-webster.com/dictionary/smartphone. [online] Accessed On: 2012-04-01.

[54] ‘smartphone’ - Oxford Dictionaries. http://oxforddictionaries.com/definition/smartphone. [online] Accessed On: 2012-04-01.

[55] Smartphone Security: How to Keep Your Handset Safe. http://www.pcworld.com/businesscenter/article/216420/smartphone_security_how_to_keep_your_handset_safe.html. [online] Accessed On: 2012-06-23.

[56] Smartphone Users Around the World - Statistics and Facts [INFOGRAPHIC]. http://www.go-gulf.com/blog/smartphone. [online] Accessed On: 2012-06-04.

[57] Status Notifications | Android Developers. http://developer.android.com/guide/topics/ui/notifiers/notifications.html. [online] Accessed On: 2012-08-05.

[58] The Nokia Story. http://www.nokia.com/global/about-nokia/company/about-us/story/the-nokia-story/. [online] Accessed On: 2012-03-29.

[59] This Month in Physics History, Nov 17 to Dec 23, 1947: Invention of the First Transistor. http://www.aps.org/publications/apsnews/200011/history.cfm. [online] Accessed On: 2012-03-25.

[60] weather - Google Play. https://play.google.com/store/search?q=weather&c=apps. [online] Accessed On: 2012-08-05.

[61] YouTube Statistics. http://www.youtube.com/t/press_statistics. [online] Accessed On: 2012-04-01.

[62] YouTube Timeline. http://www.youtube.com/t/press_timeline. [online] Accessed On: 2012-04-01.

[63] Apple. iOS App Programming Guide. Technical report, March 2012.

[64] Apple. iOS Security. Technical report, May 2012.

[65] Apple. Security Overview. Technical report, February 2012.

[66] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, Phillipa Gill, and David Lie. Short Paper: A Look at SmartPhone Permission Models. In *Proceedings of the First ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM’11. ACM, 2011.

[67] David Barrera, P.C. van Oorschot, H.G. Kayacik, and Anil Somayaji. A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android. In *Proceedings of the Seventeenth ACM Conference on Computer and Communications Security*, CCS’10. ACM, 2010.

[68] Zinaida Benenson, Nadina Hintz, Olaf Kroll-Peters, and Matthias Krupp. Poster: Attitudes to IT-Security When Using a Smartphone. In *Eighth Symposium on Usable Privacy and Security (SOUPS)*, SOUPS’12, 2012.

[69] Alistair R. Beresford, Andrew Rice, Ripduman Sohan, and Nicholas Skehin. MockDroid: Trading Privacy for Application Functionality on Smartphones. In *Proceedings of the Twelfth International Workshop on Mobile Computing Systems and Applications*, HotMobile’11. ACM, 2011.

[70] Pern Hui Chia, Yusuke Yamamoto, and N. Asokan. Is this App Safe? A Large Scale Study on Application Permissions and Risk Signals. In *Proceedings of the Twenty-First International World Wide Web Conference*, WWW'12. ACM, 2012.

[71] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing Inter-Application Communication in Android. In *Proceedings of the 9th International Conference on Mobile Systems, Application, and Services*, MobiSys'11. ACM, 2011.

[72] Cisco. The Future of Work: Information Access Expectations, Demands, and Behaviour of the World's Next-Generation Workforce - Chapter 2. Technical report, November 2011.

[73] comScore. 2012 Mobile Future in Focus. Technical report, February 2012.

[74] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadehi, and Marcel Winandy. Privilege Escalation Attacks on Android, 2010.

[75] Michael Dietz, Shashi Shekhar, Yuliy Pisetsky, Anhei Shu, and Dan S. Wallach. QUIRE: Lightweight Provenance for Smart Phone Operating Systems. In *Proceedings of the 20th Usenix Security Symposium*, Sec'11. USENIX Association, 2011.

[76] Serge Egelman, Adrienne Porter Felt, and David Wagner. Choice Architecture and Smartphone Privacy: There's A Price for That. In *Proceedings of the 11th Annual Workshop on the Economics of Information Security*, WEIS'12, 2012.

[77] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, OSDI'10. USENIX Association, 2010.

[78] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A Study of Android Application Security. In *Proceedings of the 20th USENIX conference on Security*, SEC'11. USENIX Association, 2011.

[79] F-Secure. Mobile Threat Report 2011. Technical report, Q4 2011.

[80] F-Secure. Mobile Threat Report 2012. Technical report, Q1 2012.

[81] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale, 2012.

[82] James Gillies and Robert Calliau. *How the Web Was Born: The Story of the World Wide Web*. Oxford University Press, 2000.

[83] Google. Our Mobile Planet: United Kingdom. Technical report, May 2012.

[84] Google. Our Mobile Planet: United States. Technical report, May 2012.

[85] Michael Grace, Yajin Zhou, Zhi Wang, and Xuxian Jiang. Systematic Detection of Capability Leaks in Stock Android Smartphones. In *Proceedings of the Nineteenth Annual Network & Distributed System Security Symposium*, NDSS'12. ISOC, 2012.

[86] Uwe Hansmann, Lothar Merk, Martin S. Nicklous, and Thomas Stober. *Pervasive Computing*. Springer, second edition, 2003.

[87] Andrew Hoog. *Android Forensics: Investigation. Analysis and Mobile Security for Google Android*. Syngress, 2011.

[88] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. "These Aren't the Droids You're Looking For": Retrofitting Android to Protect Data from Imperious Applications. In *Proceedings of the Eighteenth ACM Conference on Computer and Communications Security*, CCS'11. ACM, 2011.

[89] Patrick Gage Kelley, Sunny Consolvo, Lorrie Faith Cranor, Jaeyeon Jung, Norman Sadeh, and David Wetherall. A Conundrum of Permissions: Installing Applications on an Android Smartphone. In *Proceedings of the Workshop on Usable Security*, USEC'12, 2012.

[90] Kari Kostiainen, Elena Reshetova, Jan-Erik Ekberg, and N. Asokan. Old, New, Borrowed, Blue - A Perspective on the Evolution of Mobile Platform Security Architectures. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, CP-DASPY'11. ACM, 2011.

[91] Mike Gibson (Director, Enterprise: UK & Ireland at RIM). Speaking at Securing Mobile Devices (EEMA), July 2012.

[92] Mohammad Nauman, Sohail Khan, and Xinwen Zhang. Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In *Proceedings of the Fifth ACM Symposium on Information Computer and Communnications Security*, ASIACCS'10. ACM, 2010.

[93] Jon Oberheide and Farnam Jahanian. When Mobile is Harder Than Fixed (and Vice Versa): Demystifying Security Challenges in Mobile Environments. In *Proceedings of the Eleventh International Workshop on Mobile Computing Systems and Applications*, HotMobile'10. ACM, 2010.

[94] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android Permissions Demystified. In *Proceedings of the Eighteenth ACM Conference on Computer and Communications Security*, CCS'11. ACM, 2011.

[95] Adrienne Porter Felt, Serge Egelman, Matthew Finifter, Devdatta Akhawe, and David Wagner. How to Ask for Permission, 2012.

[96] Adrienne Porter Felt, Serge Egelman, and David Wagner. I've Got 99 Problems, But Vibration Ain't One: A Survery of Smartphone Users' Concerns, 2012.

[97] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android Permissions: User Attention, Comprehension, and Behavior. In *Eighth Symposium on Usable Privacy and Security (SOUPS)*, SOUPS'12, 2012.

[98] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid Android: Versatile Protection For Smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC'10. ACM, 2010.

[99] Jerome H. Saltzer and Michael D. Schroeder. The Protection of Information in Computer Systems. In *Proceedings of the Fourth ACM Symposium on Operating System Principles*, SOSP'74. ACM, 1974.

[100] Asaf Shabtai, Yuval Fledel, Uri Kanonov, Yuval Elovici, and Shlomi Dolev. Google Android: A State-of-the-Art Review of Security Mechanisms. In *arXiv:0912.5101v1*, arXiv. Cornell University Library, 2009.

[101] Asaf Shabtai, Yuval Fledel, Uri Kanonov, Yuval Elovici, Shlomi Dolev, and Chanan Glezer. Google Android: A Comprehensive Security Assessment. In *IEEE Security & Privacy - March/April 2010*, IEEE Security & Privacy. IEEE, 2010.

[102] Wook Shin, Sanghoon Kwak, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka. A Small but Non-negligible Flaw in the Android Permission Scheme. In *Proceedings of the 2010 IEEE International Symposium on Policies for Distributed Systems and Networks*, POLICY'10. IEEE, 2010.

[103] Sophos. Security Threat Report 2012. Technical report, March 2012.

[104] Symantec. The Symantec Smartphone Honey Stick Project. Technical report, March 2012.

[105] Walter Kellogg Towers. *Masters of Space: Morse, Thompson, Bell, Marconi, Carty.* Public Domain Books, 2004. (Originally published by Harper and Brothers in 1917).

[106] viaForensics. White Paper: appWatchdog Findings - Sensitive User Data Stored on Android and iPhone Devices. Technical report, July 2011.

[107] Timothy Vidas, Nicolas Christin, and Lorrie Faith Cranor. Curbing Android Permission Creep. In *Proceedings of the Web 2.0 Security and Privacy 2011*, W2SP'11, 2011.

[108] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. DroidMOSS: Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY'12. ACM, 2012.

[109] Yajin Zhou and Xuxian Jiang. Dissecting Android Malware: Characterization and Evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP'12. IEEE, 2012.

[110] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, NDSS'12. ACM, 2012.