University of Colombo School of Computing

SCS4124 - Final Year Project in Computer Science

Web of Trust for Better Privacy Protection in Android Applications

Researcher:
K.J.S. Fernando
Index Number:
12000434

Supervisor:
Dr. Kasun de Zoysa
Co-supervisor:
Mr. Primal Wijesekara

June, 2016

Contents

Acronyms **1**

1 Introduction **2**
1.1 Preamble . . . 2
1.2 Background to the Problem . . . 3
1.3 Problem Statement . . . 3
1.3.1 Privacy and Security . . . 3
1.3.2 Comparison of Application Approval Process . . . 4
1.3.3 Permission Issues Before Android 6.0 . . . 5
1.3.4 Context of Permission Requests . . . 5
1.3.5 Other Issues . . . 6
1.3.6 Problem in Summary . . . 7
1.4 Research Question . . . 7
1.5 Goal and Objectives . . . 8
1.5.1 Goal . . . 8
1.5.2 Objectives . . . 8
1.6 Assumptions and Scope . . . 8
1.6.1 Scope . . . 8
1.6.2 Delimitation . . . 9
1.7 Outline of the Thesis . . . 9

2 Literature Review **10**
2.1 The Android Permission Model . . . 10
2.1.1 Comparison of Application Approval Process . . . 10
2.1.2 Application Security and Privacy . . . 11
2.1.3 Before Android Marshmallow . . . 12
2.1.4 After Android Marshmallow . . . 13
2.2 Problems with the existing system . . . 13
2.2.1 Least Privilege . . . 13
2.2.2 Capability Leaks . . . 14
2.2.3 Data Availability After Uninstallation . . . 14
2.2.4 Permission Creep . . . 14
2.3 Proposed Solutions . . . 15

2.3.1 Facilitating Informed Decisions 15
2.3.2 Code Analysis to Curb Permission Creep 15
2.3.3 User Driven Solutions 16
2.4 Transitivity of Trust 16

Appendix 01 - Normal and Dangerous Permissions **17**
Normal Permissions 17
Dangerous Permissions 17

References **19**

Acronyms

OS	Operating System
AOSP	Android Open Source Project
Android L	Android Lollipop
Android M	Android Marshmallow
SDK	Software Development Kit
XML	Extensible Markup Language
PGP	Pretty Good Privacy
GPS	Global Positioning System
WiFi	Wireless Fidelity
SSID	Service Set Identifier
RIM	Research In Motion
UID	User Identifier
PIN	Personal Identification Number
UI	User Interface
URL	Uniform Resource Locator
VM	Virtual Machine
HAL	Hardware Abstraction Layer

1

Introduction

1.1 Preamble

Android is an open source mobile operating system currently developed and maintained by Google Inc. It was first developed as an advanced operating system for digital cameras, by a team of engineers in Palo Alto, California, and was acquired by Google in 2005. The Android Open Source Project(AOSP) is the collective name for the Linux kernel, middleware components and applications which form the Android Operating System [1]. Analysis of global smartphone market share indicates that as at March 2016, Android with a market share of 60.99% is the leading mobile operating system with a lead of 29.23% over the next most popular operating system, iOS(market share 31.76%)[3].

The current version of Android is Android 6.0 M (or Marshmallow), with the SDK version 23. Versions are traditionally named alphabetically after desserts; starting from Cupcake, Doughnut, Eclair to the most recent versions; KitKat, Lollipop and Marshmallow. Each release provides an upgraded version of the OS in terms of software, performance, functionality, features, security and privacy etc.

Users of the Android platform can install "Applications" of different types from digital distribution platforms such as Google Play, Amazon Appstore, SlideME, Mobango etc. As at July 2015, Google's own online marketplace for applications, Google Play, had more then 1.6 million different applications available for download[39]. Through Google Play, developers can publish and distribute their applications to users of Android compatible smart-phones. When uploading applications, developers are expected to specify which critical resources their applications will need access to in an XML file called the Android Manifest, which is typically found in the root folder of every Android application. Users are required to grant permission for this resource access either before installation(in versions older than Android 5.0) or during runtime(in Android 6.0 and newer devices).

1.2 Background to the Problem

Users of Android applications are expected to decide on how an application will be allowed to use data without a prior guideline to act as an indicator[18]. Users are asked to make privacy decisions before they start using the application, at which time they are not equipped with enough knowledge to do so. Up to Android version 5.0(Lollipop) permissions follow a "Do or Die" model, where users are required to grant all the required permissions at install time. Permissions could not be selected or removed individually and choosing not to grant a permission resulted in the application download being canceled. Once installed, app permissions cannot be revoked.

In versions newer than Android 6.0(Marshmallow) users are allowed to install applications granting selective privacy options, which can later be toggled depending on individual needs. This has mitigated problems that existed in previous versions where users were not given the opportunity to revoke permissions that have already been granted, or choose to grant permission only when required etc. Permissions have been classified based on how "dangerous" they are and certain types of permissions are granted automatically, while other types have to be approved by a user at runtime. The Android framework is built on a Linux kernel, and each application is assigned a UID(Linux User ID) upon installation. Permission requests are connected to the UID, which provides process isolation.

Similar situations with privacy on the internet has brought forth standards such as PGP(Pretty Good Privacy), the concepts of which include "Web of Trust". A Web of Trust is a cryptographic term for a security model where participants authenticate the identity of fellow users.The model in its simplest form is used by social networks including Facebook, LinkedIn and Google+ for user validation and networking[17]. In the context of Android security, there are research applying PGP principles to secure message passing, but not for Application permission privacy configuration[8].

Android applications require users to approve a list of permissions that will be accessible by the app before installation. This is a shortcoming in an operating system with such a wide user base as users are not equipped with enough knowledge to make a decision, which leads to making uninformed decisions and compromising their own privacy later on since they have no guideline or benchmark available as an indication of how an application will use data once it is downloaded with the necessary permissions granted. Application ratings can be given by users on most marketplaces including Google PlayStore, but this is based on a host of factors which may or may not have taken privacy and security into consideration.

1.3 Problem Statement

1.3.1 Privacy and Security

Privacy and security, although related are different concepts. Privacy is subjective; the user can decide on how private they want their data to be.However security is objective;

it is concerned with 'guarding' something that is universally accepted as confidential, such as password, credit card details, pin number etc.

Due to uninformed privacy decisions taken by users when installing apps, both privacy and security are compromised. However security is primarily threatened through malware apps which access permissions without authority, whereas privacy is compromised through users unknowingly granting permissions for applications which then misuse these privileges. Therefore we will be concentrating on privacy violations that occur through permissions which have been granted by users, as there are many research currently focused on malicious code detection and improved security of Android applications.

Users have different needs with regard to privacy, which is why they should be allowed to make their own decisions. Therefore a centralized monitoring system which blocks each privacy infraction would not be ideal since Android does not have information to predict what each user wants beforehand. Privacy infractions are therefore more difficult to predict than security threats, since user preferences also have to be taken into account. Therefore an ideal solution would be to let the users make their own decisions, while providing enough information for them to do so. In the Android platform, users are allowed to make decisions upto a certain extent, but these decisions are usually uninformed since there is no indicator as to what level of privacy will be provided by an application beforehand.

1.3.2 Comparison of Application Approval Process

The leading mobile operating systems apart from Android are iOS by Apple, Windows Mobile and BlackBerry OS. Each of these have different processes and methodologies for developers to follow before submitting an application to the store. A comparison between application submission processes for these platforms shows that each has a centralized process for validation and/or verifying an application before it is made available for users to download. However these processes are usually in place for checking security or applications, and not privacy related issues.

To submit an application to iTunes, the marketplace for iOS applications, the developer is first required to create an App ID and a Distribution Provisioning Profile and then submit an application through iTunes connect with detailed information on the app.Three different certificates have to be submitted along with the application; the Distribution Certificate, Push Notification Certificate and Mobile Provisioning Certificate. The app has to be submitted through the Publication Center, where a checklist of complying standards that need to be adhered to has to be filled in.The approval process on average takes six days to one week.[25]

Windows store applications also go through a centralized process before being released. A submission has to be created for each application with a checklist of information. Once the submission is complete and the application has been preprocessed without errors, it is submitted for certification through the Windows Certification Kit. The certification process focuses on three core areas; security tests, technical compliance

tests and content compliance tests. The amount of time taken for an application to receive approval fluctuates based factors such as the code and logic complexity, visual content, rating of the developer, other applications in the queue etc. Applications which fail the certification process will be returned with a report indicating where the compliance standards were not met. Developers are allowed to resubmit applications following the same process.[11]

RIM(Research In Motion) BlackBerry requires developers to submit applications for a complex process of reviewing, testing and "readying for publication" before being awarded a Approved/Up For Sale rating. Apps with this rating can be submitted and released on the BlackBerry marketplace; BlackBerry App World.[28]

Android developers add applications to marketplaces including Google PlayStore, Amazon AppStore, GetJar, SlideMe and F-Droid, which can then be downloaded by users. The problem lies in there being no centralized security measurement for applications on such marketplaces. Developers are trusted to prepare an application for release and then release it through a marketplace, email or website.[15] The Android operating system imposes some security and privacy restrictions, including an install-time permission system, where each application declares what permissions it requires upon installation.[2] This can provide users with control over their privacy since the choice to cancel installation lies with the user.

1.3.3 Permission Issues Before Android 6.0

Up to Android version 5 (Lollipop),users were not given an option to choose which permissions to grant upon application installation. Android v6 (Marshmallow) allows granting and revoking certain permissions upon installation, however, this model is not perfect, and even though it has been almost a year since Marshmallow was launched, it is only running on around 2.3% of Android devices. [14] The current all or nothing model forces users to either refrain from installing an application (no permissions granted) or to grant all permissions requested by an application. This can create problems since studies have shown that users tend to ignore the grant permission dialog since they have no choice except to avoid installing the app altogether. [44] The issue exists for inbuilt applications as well (most of which are classifiable as bloatware [30]), for example the Flashlight application inbuilt on devices running HTCs Android based Sense UI, requests all permissions that can be granted to an application, and since the app is inbuilt there is no way to uninstall/disable it other than rooting the device.

1.3.4 Context of Permission Requests

In the latest version of Android, Marshmallow, permissions can be toggled. However some issues still exist.

Research has shown that over 75% of permission requests rake place while the application is running in the foreground, background or as a service[44]. The context of a permission access, or specifically the time a permission is requested by an application

can contribute towards finding whether the request is legitimate. For example research has further shown that the GPS lication indicator is only visible for 0.04% of all location access, whereas in reality applications use other permissions such as WifiState to determine the location if a device through SSID information or network tower location. Data collected through such requests breach users privacy and are used for purposes such as targeted marketing [36].

Research has shown that people are moved to base decisions on what they perceive as the reason for an application to access data[44]. This may negatively affect the revenue generation model of some applications, since legitimate permission requests, sometimes connected to the revenue generation model of an application, are denied by users who assume that the request malicious. This has affected applications such as AngryBirds, where users have denied the SMS permission to the app, resulting in a failure to unlock levels users have paid for with in-app purchase payments, and even Google's own Google+ application, where thumbnails and images become invisible to users who choose not to grant the Storage permission[33] [35].

In an ideal situation a user should know why an application is requesting a particular permission; as part of its core functionality, secondary functionality or as a method of revenue generation. However, this is not possible with the current model, since the level of information made available to users regarding application permissions is decided on by the developer.

1.3.5 Other Issues

This section includes a short summary of existing issues in the Android permission model that will be examined further in chapter two; the literature review.

Permission Creep

Developers are expected to adhere to the concept of "least privilege" when requesting permissions, namely that only the minimum number of permissions required for an application to function should be requested. This is not always the case, as most developers request permissions which they feel may be required for the applications functionality in future, or just request for unwanted extra permissions due to being unaware or misinformed of these restrictions.

Capability Leaks

Capability leaks, also known as permission re-delegation is where an app uses "intents" to exploit the permission granted to another application. In such cases applications do not strictly adhere to their own permissions, but use another more privileged application to access resources. For example when a URL is received through a text message, clicking on it will lead to it being opened in a browser; even though the text message application does not have permission to open URLs it is able to do so by accessing the browser which does have that privilege.

Data Availability After Uninstallation

Upon uninstallation of an application, the Linux UID is recycled and assigned to another application when the device is next rebooted. However research has shown that even though the UID is deleted, the permissions associated with it are not deleted and data still exists as "orphans" without a unique identifier. This may later be exploited by malicious applications which can mimic the UID and access permissions, proving that the consequences of granting permissions can exist even after the application in question is uninstalled from the device.

1.3.6 Problem in Summary

The current permission model used in Android applications does not include a centralized process for certification or testing before an app is released, and instead relies on the developer to act responsibly and request the least number of permissions that are necessary. However in most cases the privileges given to developers are misused causing capability leaks, permission creep and some other issues, the consequences of which cannot be reversed even after app uninstallation. The simplest way to stop the occurrence of privacy breaches caused by the permission model would be to let the user have more control over permissions to be granted. The problem in a nutshell is that at present users are asked to make uninformed decisions, that can have wide reaching consequences, which they are not equipped to answer.

1.4 Research Question

Different types of interactions take place in this domain; users interact with applications to use the functionality and applications interact with the Operating System to access data(this is where permissions are called on-through the Linux Kernel where a unique user ID is created for each app upon installation). Further, to confirm the level of privacy provided by an app, users could interact with their peers to share the experience of whether the privacy provided by a particular application is adequate or not. Such interactions between applications and the operating system, applications and users, and users and users can be used to create a Web of Trust, focused on creating an improved permission architecture for Android applications and improving flexibility and control for users of the platform.

Our research goal is to **propose and evaluate a user-driven privacy model to overcome the problems caused due to improper permission handling in android applications**.

1.5 Goal and Objectives

1.5.1 Goal

A preliminary step of letting users have more control over granting permissions would be to provide a way for users to determine whether the application keeps to the expected permissions or accesses more than it is supposed to. This is primarily a security issue since apps access permissions without authorization. However some accesses that violate user privacy occur through permissions that *have* been granted by the user. There are many applications to detect unauthorized access and malicious code, but privacy violations occur through accesses that are not unauthorized and are hence not monitored through these applications.Therefore through this research, the expectation is to determine a method through which users will be informed about the trustworthiness of an application based on peer experience with the application.

1.5.2 Objectives

- Study applications access of permissions
- Determine a simple scale to rate apps based on privacy breaches that are authorized
- Determine a method to connect a network of users and assign 'trust' ratings to these users
- Define threshold values for the 'Web of Trust' including reach of the web, stopping point etc
- Develop a function to compute the final rating based on the application rating and user's trust rating
- Present the information in a meaningful way for the user about to install the application

1.6 Assumptions and Scope

1.6.1 Scope

The research will focus on Android smartphones with access to the Google PlayStore. Monitoring will be carried out for a selected number of applications from the PlayStore, and not from apps installed using third party websites, promotional links on social media or email, external markets such as the Amazon Appstore, F-Droid etc. The scale to determine trustworthiness of an application will be assigned simply based on factors such as frequency of uninstallation, whether uninstallation happened after a major privacy breach etc. Depending on resource constraints this rating may be equalized to a user assigned rating for an application as well. The information provided to the user will be

limited and focused on the core data needed to determine whether an application can be installed without trust issues.

The research will focus on privacy violations that occur through authorized access. Therefore applications which ask for X permission and then use X to cause some privacy violation will be the focus of this research. We will be operating on the assumptions that trust cannot be misplaced, and that the Linux kernel on which the Android framework is built is completely secure.

1.6.2 Delimitation

The period assigned for the research is limited, and hence the amount of work will be limited by time constraints. Since the purpose is to propose a framework through which information flow to users before installing applications can be improved and not to provide a complete practical implementation of the system, some features will not be included during the research period for clarity and ease of understanding.

We concentrate the research on the Google PlayStore since different markets follow different methodologies, and monitoring applications downloaded from various sources can lead to malware and security breaches. For the purpose of this research we will assume that the option to install applications from 'Unknown Sources' is disabled.The reasoning for concentrating on privacy is that there are existing research on security violations which occur through an application requesting only permissions X and Y upon installation and then using a loophole to access permission Z without authorization, typically malware and spyware. Further, the information provided will be limited by generated data during the course of analysis, and the ultimate decision on whether or not to install an application will be left to the user.

1.7 Outline of the Thesis

To be completed.

2

Literature Review

In this chapter we will explore the theoretical background of our work, including research conducted on problems caused by current Android permission models, other solutions found and related work, and applications of trust transitivity in other domains.

2.1 The Android Permission Model

The Android permission model has been developed and updated since the introduction of Android as an operating system. In this section we will examine the theoretical applications of how the permission model is expected to provide privacy and security, and the methods used by similar systems to achieve better privacy protection.

2.1.1 Comparison of Application Approval Process

The leading mobile operating systems apart from Android are iOS by Apple, Windows Mobile and BlackBerry OS. Each of these have different processes and methodologies for developers to follow before submitting an application to the store. A comparison between application submission processes for these platforms shows that each has a centralized process for validation and/or verifying an application before it is made available for users to download. However these processes are usually in place for checking security or applications, and not privacy related issues.

To submit an application to iTunes, the marketplace for iOS applications, the developer is first required to create an App ID and a Distribution Provisioning Profile and then submit an application through iTunes connect with detailed information on the app.Three different certificates have to be submitted along with the application; the Distribution Certificate, Push Notification Certificate and Mobile Provisioning Certificate. The app has to be submitted through the Publication Center, where a checklist of complying standards that need to be adhered to has to be filled in.The approval process on average takes six days to one week.[25]

Windows store applications also go through a centralized process before being released. A submission has to be created for each application with a checklist of infor-

mation. Once the submission is complete and the application has been preprocessed without errors, it is submitted for certification through the Windows Certification Kit. The certification process focuses on three core areas; security tests, technical compliance tests and content compliance tests. The amount of time taken for an application to receive approval fluctuates based factors such as the code and logic complexity, visual content, rating of the developer, other applications in the queue etc. Applications which fail the certification process will be returned with a report indicating where the compliance standards were not met. Developers are allowed to resubmit applications following the same process.[11]

RIM(Research In Motion) BlackBerry requires developers to submit applications for a complex process of reviewing, testing and "readying for publication" before being awarded a Approved/Up For Sale rating. Apps with this rating can be submitted and released on the BlackBerry marketplace; BlackBerry App World.[28]

Android developers add applications to marketplaces including Google PlayStore, Amazon AppStore, GetJar, SlideMe and F-Droid, which can then be downloaded by users. The problem lies in there being no centralized security measurement for applications on such marketplaces. Developers are trusted to prepare an application for release and then release it through a marketplace, email or website.[15] The Android operating system imposes some security and privacy restrictions, including an install-time permission system, where each application declares what permissions it requires upon installation.[2] This can provide users with control over their privacy since the choice to cancel installation lies with the user.

2.1.2 Application Security and Privacy

The Android OS is built on top of the Linux kernel, which manages the hardware including drivers and power management. From the bottom-up, the next layers include HAL or the Hardware Abstraction Layer, Native libraries and the runtime, which includes the Dalvik virtual machine, the framework with activity managers, system view and content providers, and a layer of applications(Refer Figure 2.1). Each time a new application is installed, the operating system assigns it a unique Linux user ID, which will be persistent for the lifetime of the application on that device. On uninstallation of an application, the user ID will be freed and possibly reassigned to another new application after the device is rebooted for the first time following an uninstallation.

Permissions are tied to the user IDs, attemting to provide application sandboxing and process isolation by limiting communication between applications to methods overseen by the operating system such as passing of intents. Applications get a dedicated part of the file system connected to its UID whiich can be used to write private data and store databases and raw files, including permission information. According to the description on the Android Security website, the OS "seeks to be the most secure and usable operating system for mobile platforms by re-purposing traditional operating system security controls to protect user data, protext sstem resources and provide application isolation by providing robust security at the OS level through the Linux kernel, mandatory ap-

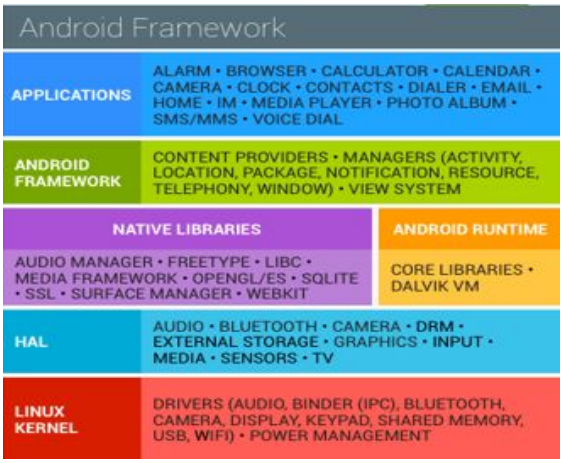

Figure 2.1: The Android Software Stack

plication sandbox for all applications, secure interprocess communication, application signing and application-defined and user-granted permissions"[5]. However, this system has several loopholes, and malicious applications can access sensitive resources even in cases where sandboxing is done by the user ID system[31].

The Android OS allows users to install both free and paid third party Java applications through markets such as Google Play. As discussed Android applications do not go through a review process to measure compliance to guidelines, although most competitors including iOS, Windows and RIM do. Access to sensitive resources, also known as "permissions" are controlled by the operating system to let aplications request access to sysytem functionality through an XML manifest, forming a computer-supported-access-system(CSAS)[40]. Applications are allowed to define their own extra permissions in addition to the 134 core permissions provided by Android[1]. (This research will not focus on third party defined permissions.)

Google Play application listings show information about each application including screenshots, marketing jargon, compatibility with user devices, app information, developer information, content rating, reviews and other related applications. However this does not include a list of permissions used by the app, or any privacy or security related information on the application page. Once a user clicks on the install button, they are expected to approve or disapprove the permission list that appears when downloading on devices older than Android M, without any background information. Permissions and privacy information are not highlighted on both the mobile application of Google Play or the store website[27].

2.1.3 Before Android Marshmallow

Before Android sdk 23, the permission model was based on a "do-or-die" concept. Users were given a list of permissions that would be required upon clicking the "install" link on an application on Google Play. There were no options to grant permissions selectively

or revoke when not needed, and choosing not to grant a particular permission resulted in the application installation process being terminated[19]. User choice was limited to whether they wanted to use the application, or not, and this has caused habituation even in later versions where permissions can be toggled[44], since users choose to approve permissions by reflex as part of the installation process of an application. Research has shown that 83% of application installations happen without the user consciously reading the permission list[20].

2.1.4 After Android Marshmallow

The current permission model assigns permissions into two categories; normal permissions and dangerous permissions(See Appendix 01 for a list of normal and dangerous permissions). Normal permissions are granted automatically when requested by a user, and can alter basic low-risk elements of a device's settings, such as the display brightness or wallpaper. Dangerous permissions are any permissions that can alter sensitive data on the device or use the device's higher-risk functions, such as connecting to the Internet, and are further classified into groups. If a permission from the same group has previously been approved by the user, then the permission will be automatically granted(i.e. if READ_SMS permission has been approved by the user then the device will not require permission to access the WRITE_SMS permission either). However, if a dangerous permission is in a category which has not been approved by the user earlier, a popup notification is generated at runtime, asking the user to approve the permission request. The permission, once granted, can be revoked by toggling a control in the application settings page later on. However, it should be noted that there has been criticism of the classification of permissions, and may researchers such as Felt have pointed out in earlier experiments that the classification should be based on risk to user privacy[18], resulting in classification systems that do not correlate with what is currently implemented by Google[4].

2.2 Problems with the existing system

The current permission model is problematic and offers loopholes that can be manipulated to compromise privacy of user data. This section will include a condensed view of research in this domain which highlight and offer solutions to some of these issues.

2.2.1 Least Privilege

Both the most popular mobile operating systems, iOS and Android, require that developers follow the principle of "least privilege" with regard to permissions. The principle of least privilege states that "Every program and every user of the system should operate using the least set of privileges necessary to complete the job"[37]. In the context of Android applications, the principle states that developers should only ask for the very minimum of permissions that are required for the application to function[16]. Applications on the Apple AppStore are screened to check adherence to this principle(and

other factors) before being made available for download[21]. However since there is no such screening process for Android applications, research has shown that these permission guidelines are not generally adhered to by developers[41]. Applications which do follow the permission guidelines are not necessarily popular with users, since permission requests either tend to be ignored due to lack of understanding[18] [26] or granted regardless of whether they are privacy sensitive or not due to habituation[20]. This results in many popular applications not following the principle of least privilege[43], with research showing that more than 33% end up asking for more permissions than are required[18].

2.2.2 Capability Leaks

Applications can sometimes access permissions which are not requested at install time. Such violations of the permission architecture to access data are referred to as 'capability leaks' [23] [22]. A tool named Woodpecker, which analyzes each application to detect readability of permissions from unguarded interfaces, is frequently used in research in this domain [47]. Through Woodpecker, two different types of capability leaks are identified; explicit leaks which find loopholes and access data without actually requesting permission and implicit leaks which let applications inherit permissions from another application, generally through intents. Other tools used to detect capability leaks include DroidChecker and IntentFuzzer [45] [12]. Capability leaks can also be exploited by malicious applications which use permissions which access permissions which have not been consiously granted to an application by a user for privilege escalation, using this to bypass restrictions on application functionality imposed due to the sand-boxing system[13].

2.2.3 Data Availability After Uninstallation

Since application permissions once granted are not revoked even upon uninstallation of an application, the data collected through the permissions granted while the application was installed may still be accessible once the app is uninstalled. Upon uninstallation, the user identity belonging to the application is deleted, but the permissions allowed are not revoked, and data still exists as orphans without a unique identifier (or parent). These orphans may later be exploited by malware causing privacy breaches and leaking of sensitive data [46]. Users misunderstanding or choosing not to read permission requests before granting create lasting consequences, the effects of which continue to compromise privacy even after uninstallation of the problematic applications.

2.2.4 Permission Creep

Permission creep occurs in applications that do not follow the principle of least privilege and ask for extra permissions[42]. Applications may sometimes require permissions that are not required for the core functionality of the application, but rather due to revenue generation methods, since most 'free' applications available on the PlayStore require

in-app purchases for extra functionality. Some of these 'free' and low cost applications may sell data to advertisers to generate revenue, without explicit permission from the user. Extra permissions may also be requested in cases where developers have difficulty trying to align permission requests with the functionality required for the application, resulting in genuinely having to request extra permissions that seems unnecessary on analysis, but are mandatory for certain functions [42]. For example an update for the popular game Angry Birds caused controversy by requesting permission to send SMS messages, which is not part of the expected functionality of the application. However Rovio (the company behind Angry Birds) later explained that this is due to the payment methodology needed to purchase new levels, where an SMS message is sent to Rovio from the device to be billed later by the carrier [35] [33].

Studies have shown that over 50% of applications that request location access do so with the intent of sharing the information with advertisers for targeted marketing[36]. However, completely disallowing such requests would negatively impact the quality of applications available for Android since the revenue generation model would not survive. [32] In an ideal situation a user should be informed as to why an application is requesting a particular permission; as part of its core functionality, secondary functionality, as a method of revenue generation or any other usage for a permission to be requested. Research has shown that people tend to base their decisions on the reason behind data access[29]. However this is not possible with the current model since the level of information made available to users regarding application permission requests is decided on by the developer.

2.3 Proposed Solutions

Apart from tools developed to identify and provide solutions to the issues discussed above, there have been several research related to alternate models or methodologies that could be followed to mitigate privacy risks in the current model.

2.3.1 Facilitating Informed Decisions

Researchers have suggested hierarchical solutions[9] and better breakdown of Android applications to create more fine-grained permission toggles[10] as solutions to the problems caused by the do-or-die model in Android versions older than M. The current model does not allow users access to information as to why a particular permission is required, and studies have shown that this could be improved through analyzing human factors more effectively when allowing users to make these decisions, with personal examples and more privacy information having been proven to provide better understanding of permission requests and result in less breaches of privacy[27] [24].

2.3.2 Code Analysis to Curb Permission Creep

Applications requesting unecessary permissions and causing permission creep can be analyzed through static code analysis, by using tools such as pScout[7], and FlowDroid[6].

Static code analysis generally involves reaing the Android Manifest and matching permissions which have been requested to those actually used by functions that provide core functionality of the application. Dynamic analysis of applications, runtime monitoring and Java code analysis have also been successful in identifying applications with permission creep [38]. Context of a permission request; time of the request, whether the screen is on or off, whether the application is running visibly as a background or foreground application or service, what the device was displaying while the permission request was taking place, frequency of repitive permission requests(for example searching for network or wifi information) have been shown to influence users when deciding whether a particular permission request is appropriate [44]. This conforms to the Theory of Contextual Integrity[34] with regard to privacy, since the context and flow contribute towards the attitude of a user towards a privacy sensitive request, and not just information such as usage, reason etc.

2.3.3 User Driven Solutions

2.4 Transitivity of Trust

Appendix 01

Normal Permissions

ACESS_LOCATION_EXTRA_COMMANDS

ACCESS_NETWORK_STATE

ACCESS_NOTIFICATION_POLICY

ACCESS_WIFI_STATE

BLUETOOTH

BLUETOOTH_ADMIN

BROADCAST_STICKY

CHANGE_NETWORK_STATE

CHANGE_WIFI_MULTICAST_STATE

CHANGE_WIFI_STATE

DISABLE_KEYGUARD

EXPAND_STATUS_BAR

GET_PACKAGE_SIZE

INSTALL_SHORTCUT

INTERNET

KILL_BACKGROUND_PROCESSES

MODIFY_AUDIO_SETTINGS

NFC

READ_SYNC_SETTINGS

READ_SYNC_STATS

RECEIVE_BOOT_COMPLETED

REORDER_TASKS

REQUEST_IGNORE_BATTERY_OPTIMIZATIONS

REQUEST_INSTALL_PACKAGES

SET_ALARM

SET_TIME_ZONE

SET_WALLPAPER

SET_WALLPAPER_HINTS

TRANSMIT_IR

UNINSTALL_SHORTCUT

USE_FINGERPRINT

VIBRATE

WAKE_LOCK

WRITE_SYNC_SETTINGS

Dangerous Permissions

READ_CALENDAR

WRITE_CALENDAR

CAMERA

READ_CONTACTS

WRITE_CONTACTS

GET_ACCOUNTS

ACCESS_FINE_LOCATION

ACCESS_COARSE_LOCATION

RECORD_AUDIO

READ_PHONE_STATE

CALL_PHONE

READ_CALL_LOG

WRITE_CALL_LOG

ADD_VOICEMAIL

USE_SIP

PROCESS_OUTGOING_CALLS

BODY_SENSORS

SEND_SMS

RECEIVE_SMS

READ_SMS

RECEIVE_WAP_PUSH

RECEIVE_MMS

READ_EXTERNAL_STORAGE

WRITE_EXTERNAL_STORAGE

Bibliography

[1] The android open source project. https://source.android.com/.

[2] Patterns: Permissions. https://www.google.com/design/spec/patterns/permissions.html#.

[3] Market share statistics for internet technologies; mobile/tablet operating system market share. https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1, 2016.

[4] AOSP. Android security 2015 year in review - android open source project. https://static.googleusercontent.com/media/source.android.com/en//security/reports/Google_Android_Security_2015_Report_Final.pdf.

[5] AOSP. Security program overview. https://source.android.com/security/.

[6] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *ACM SIGPLAN Notices*, volume 49, pages 259–269. ACM, 2014.

[7] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228. ACM, 2012.

[8] Benjamin Aziz and Mario Tejedor-Gonzalez. An android pgp manager: towards bridging end-user cryptography to smart phones. *International Journal of Security*, 6(5), 2012.

[9] David Barrera, H Güneş Kayacik, Paul C van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 73–84. ACM, 2010.

[10] Sven Bugiel, Stephan Heuser, and Ahmad-Reza Sadeghi. Flexible and fine-grained mandatory access control on android for diverse security and privacy policies. In *Usenix security*, pages 131–146, 2013.

[11] Windows Dev Center. Msdn, the app certification process. `https://msdn.microsoft.com/en-us/windows/uwp/publish/the-app-certification-process`.

[12] Patrick PF Chan, Lucas CK Hui, and Siu-Ming Yiu. Droidchecker: analyzing android applications for capability leak. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pages 125–136. ACM, 2012.

[13] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Marcel Winandy. Privilege escalation attacks on android. In *Information Security*, pages 346–360. Springer, 2010.

[14] Android Developers. Android developers: Dashboards. `http://developer.android.com/about/dashboards/index.html`.

[15] Android Developers. Publishing overview. `http://developer.android.com/tools/publishing/publishing_overview.html`.

[16] William Enck, Machigar Ongtang, and Patrick McDaniel. Understanding android security. *IEEE security & privacy*, (1):50–57, 2009.

[17] Patrick Feisthammel. Pgp: Explanation of the web of trust of pgp. `https://www.rubin.ch/pgp/weboftrust.en.html`, 2004.

[18] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, 2011.

[19] Adrienne Porter Felt, Kate Greenwood, and David Wagner. The effectiveness of application permissions. In *Proceedings of the 2nd USENIX conference on Web application development*, pages 7–7, 2011.

[20] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 3. ACM, 2012.

[21] Peter Gilbert, Byung-Gon Chun, Landon P Cox, and Jaeyeon Jung. Vision: automated security validation of mobile apps at app markets. In *Proceedings of the second international workshop on Mobile cloud computing and services*, pages 21–26. ACM, 2011.

[22] Michael Grace, Yajin Zhou, Zhi Wang, and Xuxian Jiang. Detecting capability leaks in android-based smartphones. Technical report, Technical Report North Carolina State University, 2011.

[23] Michael C Grace, Yajin Zhou, Zhi Wang, and Xuxian Jiang. Systematic detection of capability leaks in stock android smartphones. In *NDSS*, 2012.

[24] Marian Harbach, Markus Hettig, Susanne Weber, and Matthew Smith. Using personal examples to improve risk communication for security & privacy decisions. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pages 2647–2656. ACM, 2014.

[25] Apple Inc. ios developer library-app distribution guide. `https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/SubmittingYourApp/SubmittingYourApp.html`.

[26] 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 *Financial Cryptography and Data Security*, pages 68–79. Springer, 2012.

[27] Patrick Gage Kelley, Lorrie Faith Cranor, and Norman Sadeh. Privacy as part of the app decision-making process. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3393–3402. ACM, 2013.

[28] Alex Kinsella. How to submit your apps for blackberry 10. `http://devblog.blackberry.com/2012/10/submit-apps-blackberry-10/l`.

[29] Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I Hong. Modeling users mobile app privacy preferences: Restoring usability in a sea of permission settings. In *Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 199–212, 2014.

[30] Patrick McDaniel. Bloatware comes to the smartphone. *IEEE Security & Privacy*, (4):85–87, 2012.

[31] PD Meshram and RC Thool. A survey paper on vulnerabilities in android os and security of android devices. In *Wireless Computing and Networking (GCWCN), 2014 IEEE Global Conference on*, pages 174–178. IEEE, 2014.

[32] T. Moynihan. Wired: Apps snoop on your location way more than you think. `http://www.wired.com/2015/03/apps-snoop-location-way-think/`, 2015.

[33] P. Nickinson. Android central: Rovio explains new permissions in angrybirds seasons update. `http://www.androidcentral.com/rovio-explains-newpermissions-angry-birds-seasons-update`, 2014.

[34] Helen Nissenbaum. Privacy as contextual integrity. *Wash. L. Rev.*, 79:119, 2004.

[35] A. Russakovskii. Rovio explains why the sms permission was introduced in angry birds v1.5.1. `http://www.androidpolice.com/2011/02/06/rovio-explainswhy-the-sms-permission-was-introduced-in-angry-birds-v1-5-1/`, 2011.

[36] N Saint. 50% of android apps with internet access that ask for your location send it to advertisers, 2010.

[37] Fred B Schneider. Least privilege and more. In *Computer Systems*, pages 253–258. Springer, 2004.

[38] Michael Spreitzenbarth, Felix Freiling, Florian Echtler, Thomas Schreck, and Johannes Hoffmann. Mobile-sandbox: having a deeper look into android applications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1808–1815. ACM, 2013.

[39] Statistica. Number of apps available in leading app stores as at july 2015. `http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/`, 2016.

[40] Gunnar Stevens and Volker Wulf. Computer-supported access control. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 16(3):12, 2009.

[41] Ryan Stevens, Jonathan Ganz, Vladimir Filkov, Premkumar Devanbu, and Hao Chen. Asking for (and about) permissions used by android apps. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 31–40. IEEE Press, 2013.

[42] Timothy Vidas, Nicolas Christin, and Lorrie Cranor. Curbing android permission creep. In *Proceedings of the Web*, volume 2, pages 91–96, 2011.

[43] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 31–40. ACM, 2012.

[44] Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. Android permissions remystified: a field study on contextual integrity. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 499–514, 2015.

[45] Kun Yang, Jianwei Zhuge, Yongke Wang, Lujue Zhou, and Haixin Duan. Intentfuzzer: detecting capability leaks of android applications. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 531–536. ACM, 2014.

[46] Xiao Zhang, Kailiang Ying, Yousra Aafer, Zhenshen Qiu, and Wenliang Du. Life after app uninstallation: Are the data still alive? data residue attacks on android. 2016.

[47] 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 *NDSS*, 2012.