Curbing Android Permission Creep

Timothy Vidas
Carnegie Mellon ECE/CyLab
tvidas@ece.cmu.edu

Nicolas Christin
Carnegie Mellon INI/CyLab
nicolasc@andrew.cmu.edu

Lorrie Faith Cranor
Carnegie Mellon CS/CyLab
lorrie@cs.cmu.edu

***Abstract*—The Android platform has about 130 application level permissions that govern access to resources. The determination of which permissions to request is left solely to the application developer. Users are prompted to approve all application permissions at install time, and permissions are silently enforced at execution time. Although many applications make use of a wide range of permissions, we have observed that some applications request permissions that are not required for the application to execute, and that existing developer APIs make it difficult for developers to align their permission requests with application functionality. In this paper we describe a tool we developed to assist developers in utilizing least privilege.**

***Index Terms*—Android Framework, Privacy, Software Development, Least Privilege**

I. Introduction

Android, the Google-backed mobile software framework, is reportedly enjoying a larger market share and growth factor than Apple's iPhone [1], [10], quickly making Android a major player in the mobile market. The central repository for mobile applications, the "Android market," enjoys more freedom than the Apple moderated iTunes Store for iPhone applications. A would-be Android developer need only register for an account and pay the $25 fee. Further, Android applications can be installed from third party websites circumventing the Android market altogether. Users historically make poor privacy and security decisions especially when a warning is difficult to understand and/or acts as a barrier to immediate gratification [5]. Like many Google products, Android seeks to limit the volume of privacy and security related warnings presented to the user. Application privacy and security settings are accepted by the user prior to install, and the user typically never again has to make a privacy or security related decision pertaining to that application[1]. An example of an install time prompt can be seen in Figure 1 in which a battery monitoring application is requiring access to the GPS device, phone state, Bluetooth, Internet access and a multitude of other permissions. The install time permission requirements are determined by the developer and may or may not accurately reflect permissions that the application actually requires for proper operation.

While the Android platform already provides copious amounts of information to developers as well as a reasonably rich development environment, there is no straightforward way for a developer to determine appropriate permissions to request. Developer specified permission requirements may in fact be a superset of the permissions the application actually requires resulting in violation of the principle of least privilege [18]. Least privilege is an important aspect of system design, benefiting system security and fault tolerance. The main contributions of this paper are to describe a tool, Permission Check Tool, that aids the developer in specifying a minimum set of permissions required for a given mobile application, and to describe the creation of an associated API-permission database for Android. The tool analyzes application source code and automatically infers the minimal set of permissions required to run the application. This approach is unique in the method used to determine permission requirements and, to encourage adoption, is implemented alongside the existing IDE recommended for Android development.

First we discuss Android's permission model in section II. In section III we describe the inner workings of the Permission Check Tool. Finally we present related work in section IV and have a concluding discussion providing some avenues for future work in section V.

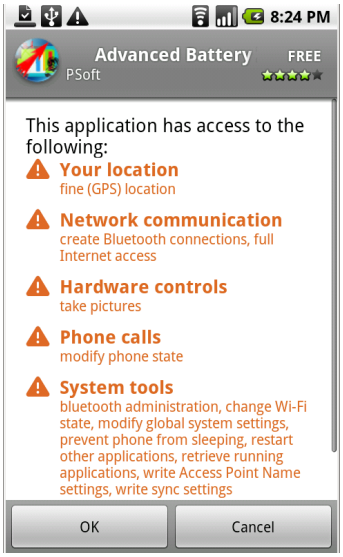

Fig. 1. **Android permission prompt during application install** Why does a Battery application require so many invasive permissions?

II. Android Permission Model

Built upon a Linux kernel, Android uses operating system primitives (such as processes and user IDs) as well as a Java Virtual Machine (Dalvik) to isolate applications providing a safety sandbox [22]. The Android platform introduces about 130 application level permissions [8], [21] that are requested

[1]Unless the permissions change between version upgrades, in which case the user will be prompted once for the new versions permissions

at install time and silently enforced any time the application is executed [14]. Unlike other privacy models, such as the iPhone, the user is not prompted when an application requests a resource during execution.

These application level permissions generalize access controls into categories like "access location information" or "access the network." In some cases this generality may force an application to request more access than needed. Consider, for instance, an Internet radio application that only needs access to a single URL on a single domain, but must request access to all network resources. Other applications request large sets of controls, many of which intuitively have little or no use to the advertised functionality of an application. The ambiguity, generality and misuse of Android application permissions have led to statistics such as "50% of applications send info to third parties" [15]. This information disclosure is asserted to be "clearly wrong" [15] even though the practice is likely in accordance with Google's privacy policy[2] [2]. Other studies claim 1 in 5 applications are a privacy threat, 1 in 20 can make arbitrary phone calls, 3% can arbitrarily send text messages, and 383 can access authentication information from other applications and service [23].

Users have become habituated to clicking through terms of service and warnings, and Android users are thus unlikely to pay much attention to notices about Android permissions. Average users cannot be expected to understand the semantics of approximately 130 permissions. Similarly, users likely tend to be unaware of the privacy impacts of their decisions. The disconnect between the user accepting security and privacy settings, once, during install and the enforcement of these settings upon subsequent application execution, along with ambiguous permission descriptions can lead to a fundamentally unaware user base. Given that there is no central moderator of the Android market, the policing of applications is largely left to these same users.

We suggest that applications that request more permission than needed fit into two general sets: applications that "do something 'questionable' or 'malicious' on purpose" (e.g., quietly collect information for advertising purposes), or applications that inadvertently request additional permissions due to lack of understanding, laziness, or expected future use of a capability. In the first case, the mobile application actually requires permissions that may seem unnecessary to a user. We take a simplistic definition of "proper operation" assuming that all API calls that require a permission are required for proper operation. For example, an application marketed as a battery monitor possibly *should* not require the GPS permission that the application requested. For our purposes, if the application makes use of the API interface for location, then the permission request is legitimate. However in the second case, when superfluous permissions are requested, the application is clearly requesting more permission that required, a plain violation of the principle of least privilege [18]. Note that under our definition, if the previously mentioned battery monitor application actually makes use of all associated API calls, it may actually have correctly specified permissions. Of course, such an application may not be classified similarly under other metrics such as EULA or user expectation, If the second set was reduced to near empty, then users would only have be wary of "questionable" applications.

Much of the application-specific burden of identifying requested permission falls with application developers, who are required to specify permissions that an application will request. The Android platform has no straightforward way for a developer to determine appropriate permissions to request. Either the developer needs to observe in the online API documentation that a permission is needed by a particular function call, or, more likely, observe in the emulator that a Java error is thrown when such a function is called. This problem posed to the developer is compounded by the granularity and ambiguity of the permission mnemonics and associated descriptions. The permission `READ_SMS` is a fairly straight-forward and specific, but `INTERNET`, while straightforward, is very general. Other permissions may be considered obtuse (`ACCESS_SURFACE_FLINGER`[3]) or ambiguous (`DIAGNOSTIC`). Even the clear permissions have created serious user confusion, as evidenced by the user outrage when Rovio added the SMS permission to their popular game Angry Birds. The permission was required because Rovio had added mobile payment capabilities allowing in-game purchasing over SMS [12].

Even though applications obtained from the market are in a compiled form that does not readily permit source code analysis, the requested permissions can be extracted from the binary XML with relative ease. From a corpus of 34,000 free Android applications obtained from the Android market in March 2011, we observed indicators that developers are currently not specifying permissions according to least privilege. For example, More than four percent of the applications specify *duplicate* permissions. That is, the application manifest contained the same permission more than once (for some, many times). Table I shows the number of applications that request duplicate permissions by market category, demonstrating that even reference libraries and medical applications contain some duplicates. Table II shows the permissions most often duplicated.

III. Permission Check Tool

We have created a tool that aids the developer in assessing appropriate application permission requests. To encourage widespread adoption, this tool is implemented as an Eclipse

[2]Google's Privacy Policy is fairly general and vague allowing for many forms of information collection and sharing. For example, the policy contains verbiage such as *"We may share with third parties certain pieces of aggregate, non-personal information," "we provide such information to our subsidiaries, affiliated companies or other trusted businesses or persons for the purpose of processing personal information on our behalf," "We may process personal information to provide our own services. In some cases, we may process personal information on behalf of and according to the instructions of a third party, such as our advertising partners,"* that allow for a wide variety information collection and sharing.

[3]The extended description of this permission is "Allows an application to use SurfaceFlinger's Low Level Features."

Market Category	Total Apps	With Duplicates
Arcade and Action	1344	17
Books and Reference	1452	24
Brain and Puzzle	1352	14
Business	1092	38
Cards and Casino	842	85
Casual	966	4
Comics	838	11
Communication	1311	77
Education	1305	21
Entertainment	1522	40
Finance	1354	44
Health and Fitness	1258	36
Libraries and Demo	1156	21
Lifestyle	1489	48
Live Wallpaper	537	14
Media and Video	1360	49
Medical	527	2
Music and Audio	1124	89
News and Magazine	1419	63
Personalization	1342	54
Photography	1165	283
Productivity	1319	54
Racing	216	76
Shopping	1155	46
Social	1296	41
Sports	1433	23
Sports Games	365	71
Tools	690	27
Transportation	454	8
Travel and Local	1473	35
Weather	342	4
Widgets	1395	64

TABLE I
APPLICATIONS WITH DUPLICATE PERMISSIONS BY MARKET CATEGORY

Permission	Count
INTERNET	620
ACCESS_NETWORK_STATE	438
READ_PHONE_STATE	153
RECEIVE_BOOT_COMPLETED	147
WRITE_EXTERNAL_STORAGE	59
READ_CONTACTS	49
ACCESS_FINE_LOCATION	48

TABLE II
TOP TEN DUPLICATE PERMISSIONS REQUESTED

IDE plugin that can be used alongside the Android-specific development environment provided in the SDK. The tool evaluates an Android application for platform permission access and informs the developer on minimum controls required for proper execution. In this section we discuss the creation of a permission-API database and the Eclipse implementation.

A. Permission-API database

We created one-to-many permission-API mappings by manually parsing the API documentation[4] and creating a database of functions and permissions upon which they depend. Some permission mappings are more complex than others. For example, instantiating and using a `BluetoothSocket` requiring the `BLUETOOTH` permission, is a fairly straightforward example, but the `LocationManager` class cannot be instantiated directly and the permission varies based on constants used in the instantiation: when using `GPS_PROVIDER` with `LocationManager` the required permission is `ACCESS_FINE_LOCATION`, when using `NETWORK_PROVIDER` with `LocationManager` the permission is `ACCESS_COURSE_LOCATION`.

Permission-API databases can, and should, be created a priori and simply loaded by the tool the first time the plugin is executed. Once created, the databases should require little maintenance since each database is particular to an Android API revision which is static. The only maintenance would be the result of an error or omission in the database itself.

Inconsistent nomenclature in the documentation further complicates the creation of a Permission-API database. Figure 2 shows several examples taken directly from the documentation. The class overview for `BluetoothSocket` shows the requirement for the `BLUETOOTH` permission in a "Note": "Requires the `BLUETOOTH` permission." Similarly the documentation for the `disable[]` function we see "Requires the `BLUETOOTH` permission" but not as a "Note." Other instances demonstrate more variation as is the case with `KILL_BACKGROUND_PROCESSES` which is called out with "You must hold the permission...to be able to call this method" in lieu of the more common "required." In Figure 2 we see that a permission required for `restartPackage` is actually noted in the text associated with `killBackgroundProcess`.

As a result of these complexities, the version of the database developed as a proof of concept for our tool handles all method (function) and class level permission enforcement denoted in the SDK documentation for Android 2.2. The database consists of two sets of one-to-many mappings: permission associated with one or more methods and permission associated with one or more classes. Permissions that have no associated method (or class) have no entry in the set.

B. Static Analysis

When the user invokes the plugin, the plugin parses application requested permissions from `AndroidManifest.xml` into a map. Each map entry stores meta information about permission, such as source code line number, and is initially marked as *unused*. The plugin then inspects all java source files (Eclipse IProject "members") using Eclipse's built in API functions for Java, in particular the Abstract Syntax Tree (AST). Each API reference is checked against the Permission-API databases, if a permission is found to be required for a reference the associated map entry is marked as *used*. Once all source has been inspected, permission entries in the map still marked as *unused* are known to be extraneous. As an additional aid to the developer, references that are found to require a permission that was not specified in the `AndroidManfiest.xml` are also tracked in order to

[4]While no longer automatically distributed, it can be locally installed using the SDK Manager tool.

suggest additional permissions for inclusion in the manifest and prevent error conditions during execution.

C. User Interface Notification

After the static analysis completes, both permissions that have been specified by the developer but are not required and permissions that are required by the application but have not been specified are known. The plugin again utilizes familiar Eclipse features to notify the developer of any omitted and/or extraneous items. An extraneous permission is shown in Figure 3 with a red "error" mark and associated tooltip text. The "error" condition is recognized by Eclipse and will prevent the build (unless the developer corrects the condition and reruns the tool, or manually clears the mark). Similarly, the tool will notify the developer, using a yellow "warning" mark, in the case where a function requiring a permission is use yet the developer hasn't specified the appropriate permission in the manifest. By using the existing Eclipse interfaces, other views, such as the Problems View, can be used to obtain a list of permissions related errors for the entire Eclipse Workspace.

D. Tool Results

Of the applications obtained directly from the Android market, only a very small fraction of the developers have

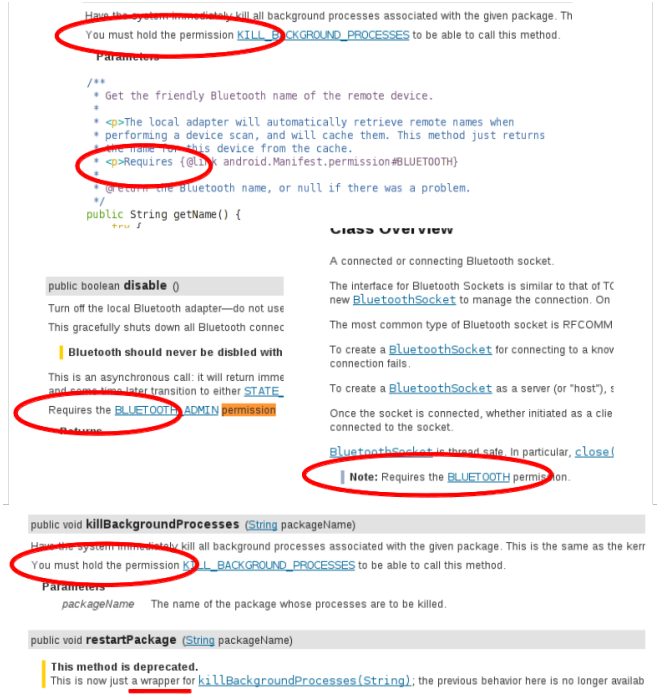

Fig. 2. **Android Documentation examples of calling out required permissions**

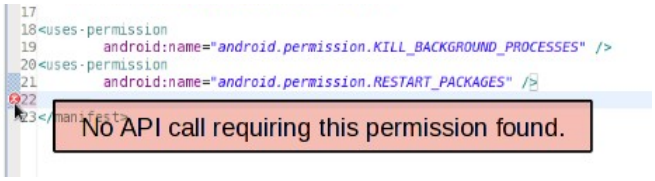

Fig. 3. **Eclipse plugin Highlighting Extraneous Permission**

Application	Extra Permissions
droidcon	1
meshapp	2
posit mobile	7
selenium	1
wifi-tether	1
YouTube Direct	3

TABLE III
EXTRANEOUS PERMISSIONS FOUND IN OPEN ANDROID APPLICATION SOURCE

elected to make source code available to the public, making empirical analysis of our source code oriented tool difficult. Many mobile applications are developed by novice individuals or small groups that have minimal quality assurance or code auditing procedures. One might predict that, due to the open audit ability, applications that have available sources are more likely to have minimal permissions specified than the closed source applications. Even so, cursory searching revealed that some of these open sourced applications indeed have or have had extraneous permissions specified. As shown in Table III, six applications were identified to have extraneous permissions via source analysis. Not only did applications include extra permissions, but some specified the same permission multiple times [4], [6] or specified fictitious permissions [3].

IV. RELATED WORK

Several groups have explored Android security and attempted to formalize the security model [8], [22], apply security enhancements found on modern computers to Android [13], [17], [20], [21], and adapt or extend Android's current models [9], [11], [14], [16]. Some of the general reviews of Android security also take into account characteristics specific to the mobile market, such as battery life or billing based on throughput [16], [21]. Related research spans across the underlying Linux base, the Android middleware and Android applications. Android applications have been studied in the context of a permission model and as case studies. Case studies often involve the collection and various analysis of a percentage of the Android market. This percentage varies widely in collection manner and number of applications used in the study (from 30 [9] to 48,000 [23]).

Twenty-five Firefox browser extensions were analyzed by Barth et al. [7] who found that 78% request more privilege than required, demonstrating a lack of least privilege in the browser extensions. Enck et al. scrutinize 30 applications demonstrating their dynamic taint system on Android. Of the 30 popular applications, 1/2 to 2/3 were found to exhibit questionable behavior such as reporting location to 3rd parties or disclosing sensitive information [9]. In this case the applications were clearly utilizing the permissions requested of the application[5]. Complementing this work, we have focused on applications that request extraneous permissions neatly addressing both sets of applications mentioned in section II.

[5]Enck et al. also note that the EULA and privacy policies, if provided, omit or are not clear about advertising and 3rd party data collection

Perhaps the most related work is [19] which explores iPhone privacy issues in rooted iPhones, iPhones vulnerable due to software vulnerabilities, and fully patched iPhones that ostensibly only allow data access through the published API. The author releases an open source, proof of concept application: SpyPhone, which collects users data. Some of this collected data would likely be expected by a typical smartphone user: address book, phone call logs, email messages. Other bits of information may cause alarm in many users such as the keyboard cache file.

V. Discussion and Future Work

Market share alone demonstrates the viability of the Android platform. Both the generality and granularity of application-level permissions warrant some concern to users and developers alike. However, even if the Android security and privacy models remain unchanged, developers can create applications that request only the minimum set of permissions required for execution. We have presented a tool that aids developers in just this way by highlighting code that require certain permissions (and thus avoiding errors during execution) and highlighting requested permissions that are not needed (maintaining least privilege, a desirable security feature).

The tool described here is intended as a source analysis tool for developer use. There is no requirement for developers to release the source code of Android applications. Compiled and packaged applications are typically a collection of binary XML and JAVA classes in a zipped format. As such, there is no ready dataset of Android application source code which makes extensive empirical analysis of this plugin difficult.

Several updates and additions could be made to the tool. For example, the tool already integrates with the suggested IDE for Android development, but the process of creating an appropriate permission set could be automated as part of the build process. The permission-function pairs used by the tool need to be generated for all API levels. Additionally, there are several indirect enforcements of permissions through bit-field class data members, or class interfaces that are more difficult to extract from documentation. It may be difficult to enumerate all instances from documentation and API fuzzing may be more appropriate. Eclipse integration could also be improved through the use of an Eclipse *Nature* allowing for dynamic checking of extra and omitted permissions as software is developed. A Nature can also automatically detect when the user had corrected an error mark eliminating the need for the developer to re-run the tool. Using the Eclipse AST to locate API references is a convenient method of source inspection, but more advanced static analysis techniques could be employed.

Corpora of applications and application source code should be created from open sources and then analyzed using future versions of this tool. Such corpora will be useful for many future Android based research projects, both related to this work and not. In this work we assumed a very broad definition of "proper" in regard to the operation of an application. Applications such as a battery monitor need very few permissions, and certainly should not require Internet access, contact list access or location information in order to report battery information to the user. We intend to study the privacy and security impacts of publicly available Android applications that offer similar extended functionality.

The current version of the tool should be considered beta quality. The plugin is available as a jar file available at http://www.ece.cmu.edu/~tvidas/PermCheckTool.jar and can be installed by saving the file into the plugins directory for Eclipse.

References

[1] Android most popular operating system in us among recent smartphone buyers|nielsen wire. http://blog.nielsen.com/nielsenwire/online_mobile/android-most-popular-operating-system-in-u-s-among-recent-smartphone-buyers/, Oct. 2010.

[2] Android.com. http://www.android.com/privacy.html, Oct. 2010.

[3] Posit-mobile issue 100. http://code.google.com/p/posit-mobile/issues/detail?id=100, Nov. 2010.

[4] selenium revision log. http://code.google.com/p/selenium/source/diff?path=/trunk/android/server/AndroidManifest.xml\&format=side\&r=10729\&old_path=/trunk/android/server/AndroidManifest.xml\&old=10639, Dec. 2010.

[5] A. Acquisti and J. Grossklags. Privacy and rationality in individual decision making. *Security & Privacy, IEEE*, 3(1):26–33, 2005.

[6] D. Barrera, H. Kayacik, P. van Oorschot, and A. Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *CCS*, pages 73–84. ACM, 2010.

[7] A. Barth, A. Felt, P. Saxena, and A. Boodman. Protecting browsers from extension vulnerabilities. In *NDSS*. Citeseer, 2010.

[8] A. Chaudhuri. Language-based security on android. In *ACM SIGPLAN Workshop on Prog. Lang. and Analysis for Security*, pages 1–7, 2009.

[9] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth. TaintDroid: an Information-Flow tracking system for realtime privacy monitoring on smartphones. In *OSDI 2010*, Vancouver, BC, Canada.

[10] M. Meeker, S. Devitt, and L. Wu. Ten questions internet execs should ask & answer. San Fransisco, CA, Nov. 2010. Web 2.0 Summit.

[11] M. Nauman, S. Khan, and X. Zhang. Apex: extending android permission model and enforcement with user-defined runtime constraints. In *CCS*, pages 328–332. ACM, 2010.

[12] P. Nickinson. http://m.androidcentral.com/rovio-explains-why-angry-birds-update-needs-sms-permission, Feb. 2011.

[13] M. Ongtang, K. Butler, and P. McDaniel. Porscha: Policy oriented secure content handling in android. pages 221–230, 2010.

[14] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically rich application-centric security in android. pages 340–349. IEEE, 2009.

[15] N. Saint. 50% of android apps with internet access that ask for your location send it to advertisers. http://www.businessinsider.com/50-of-android-apps-that-ask-for-your-location-send-it-to-advertisers-2010-10, Oct. 2010.

[16] A. Schmidt, R. Bye, H. Schmidt, J. Clausen, O. Kiraz, K. Yuksel, S. Camtepe, and S. Albayrak. Static analysis of executables for collaborative malware detection on android. In *ICC*, pages 1–5. IEEE, 2009.

[17] A. Schmidt, H. Schmidt, J. Clausen, K. Yksel, O. Kiraz, A. Camtepe, and S. Albayrak. Enhancing security of linux-based android devices. In *15th International Linux Kongress, Lehmann*, 2008.

[18] M. Schroeder and J. Saltzer. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[19] N. Seriot. iPhone privacy. Arlington, VA, 2010. Blackhat DC.

[20] A. Shabtai, Y. Fledel, and Y. Elovici. Securing Android-Powered mobile devices using SELinux. *IEEE Security and Privacy*, 2009.

[21] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. Google android: A comprehensive security assessment. *Security & Privacy, IEEE*, 8(2):35–44, 2010.

[22] W. Shin, S. Kiyomoto, K. Fukushima, and T. Tanaka. Towards formal analysis of the Permission-Based security model for android. pages 87–92. IEEE, 2009.

[23] T. Vennon and D. Stroop. Threat analysis of the android market. Technical report, Tech. rep., SMobile Systems, 2010, June 2010.