Analysis of Android Applications' Permissions

Ryan Johnson, Zhaohui Wang
Center for Secure Information Systems
George Mason University
4400 University Dr, Fairfax, Virginia 22030
Email: {rjohnso8, zwange}@gmu.edu

Corey Gagnon
Department of Computer Science
James Madison University
Harrisonburg, Virginia 22807
Email: gagnonca@dukes.jmu.edu

Angelos Stavrou
Center for Secure Information Systems
George Mason University
4400 University Dr, Fairfax, Virginia 22030
Email: astravrou@gmu.edu

Abstract—**We developed an architecture that automatically searches for and downloads Android applications from the Android Market. Furthermore, we created a detailed mapping of Android application programming interface (API) calls to the required permission(s), if any, for each call. We then performed an analysis of 141,372 Android applications to determine if they have the appropriate set of permissions based on the static analysis of the APK bytecode of each application. Our findings indicate that the majority of mobile software developers are not using the correct permission set and that they either over-specify or under-specify their security requirements.**

I. Introduction

The Android Market [1] has experienced tremendous growth since its inception and has democratized application development by allowing anyone to upload applications to the market. Application developers may be compelled to develop applications for monetary gain by charging for their application or by receiving ad revenue. There can be tendency for the developer to be liberal when allocating permissions to an application in order to avoid the application possibly encountering an exception [2]. In addition, the online documentation for the Android API is incomplete which further complicates the process of assigning the strictest permission set to an application. We perform static analysis on Android applications in order to determine the exact permission set required by the application based on its functionality. Our program also looks for the presence of certain Android API calls due to their potential to obfuscate the application's behavior.

II. Permissions

The Android operating system (OS) uses a permission-based model to limit the behavior of an application and to inform the user of the application's potential behavior. An application declares the permissions that it requests in its AndroidManifest.xml file. The user is presented with the list of permissions that an application requests when the application is to be installed. The user gets to make the choice whether or not to install the application based on the list of permissions it says that it requires. The user cannot selectively allow or disallow individual permissions that the application requests since installation is done on an all or nothing basis. Once an application is installed, the permissions that it has remains static.

The use of the important resources on the phone is controlled by the use of permissions. For example, to use the GPS resource on an Android phone, the application must possess the ACCESS_COARSE_LOCATION permission or the ACCESS_FINE_LOCATION permission. If an application requests the GPS resource without having the appropriate permission, then the Android OS may throw a Security Exception or simply not grant the requested resource. These permission-protected resources are accessed through the Android API and other classes resident on the phone. There is a mapping from each Android API call to the associated permission(s), if any, that it requires for proper operation. Having the ACCESS_FINE_LOCATION permission in an application's AndroidManifest.xml file will give the application access to a number of Android API calls that use the GPS resource. Over-provisions of permissions increases the attack surface of an application offering an attacker more capabilities if the application becomes compromised.

Researchers at Berkeley [3] compiled an extensive list of intents, content providers, and Android API calls which require permissions for Android 2.2.1. Although their approach is thorough, they did not adequately map instances of back-end server calls that many of the public API classes interface with in order to obtain access to permission-protected resources. Many of these back-end server objects cannot actually be instantiated unless the application is a system process.

All of the new permission-protected calls we identified are from the com.android.server package. We searched for permission names in the Android source code and tested the calls that contained permission names. We identified an additional 146 Android API calls that require at least one permission. These calls either need to be performed using Java reflection or by modifying the Android.jar file used by the Eclipse Integrated Development Environment to include these classes. We have also identified 217 more Android API calls that perform a permission check in the source code, but require a permission only available to system processes. We combined our mapping with Berkeley's mapping to yield a more complete mapping. We have written a program that examines an application and uses the mapping to determine if a program has more permissions than it needs and/or less permissions than it requires.

III. Analysis Methodology

During the period of March 2011 to November 2011, we downloaded 141,372 applications from the official Android

Market. We developed a program to automate the searching and downloading of free applications on the Android Market. We utilized the android-market-api [4] for its ability to interact with the market. We modified our program to download the searched applications, and ran various instances of the program concurrently. We used different methods for searching such as using a dictionary file, downloading recently uploaded applications, downloading by application category, and using all permutations of a custom character set using fixed length strings.

An apk file is a compressed file that contains the various resources that comprise an Android application. apktool [5] is a free open-source utility that unpacks an apk file into its constituent resources and then disassembles the Android application's classes.dex file into a format called smali. Each application has a single classes.dex file, and apktool disassembles it into a directory tree of smali files. The smali files contain a human-readable representation of the Dalvik bytecode which shows the instructions being used as well as the registers, objects, and literal values that are used as arguments. The smali files also contain information about methods, constructors, instance variables, data types, and the assignment of variable names.

Sensitive behavior on the phone is protected by the use of permissions. We used the mapping of Android API calls to permissions to determine if the set of permissions requested by an application is correct. There are numerous permission-protected method calls that are not part of the public Android API, but are in classes that are resident on the phone. Our program examines all the application's smali files to obtain a list of method calls used in the application. Each method call found in the application is compared to each method call in our list of permission-protected Android API calls to obtain the associated permissions.

The program determines the strictest permission set that application should have according to its functionality to be consistent with the principle of least privilege [6]. The generated strictest permission set is then compared to the permission set that is declared in the application's AndroidManifest.xml file. The program determines if the application has extra permissions, lacks permissions, or has exactly the permission set that it requires based on its functionality. In addition to the analysis of permissions, our program detects the use of Java reflection, and dynamic class-loading. These behaviors can help to obfuscate the behavior of the program [7].

IV. Analysis Results

The results are based on running our static analysis program on 141,372 Android applications as indicated in Table 1. We consider an application to have extra permission(s) if it has at least one permission that it does not require. Lacks permission(s) is for applications that should have one or more permissions that they currently do not request. Using this classification, an application may concurrently have extra permission(s) and lack permission(s). Exact permission(s) is for applications that have the exact permission set that they require. Reflective method invocation is for applications that use the java.lang.reflect.Method.invoke() method. ClassLoader usage is for applications that use the java.lang.ClassLoader class or any of the other 5 classes used for dynamic class loading in the Android API.

	Number of applications	Percent
Extra permission(s)	76,366	54.01%
Lacks permission(s)	70,618	49.95%
Exact permission(s)	28,199	19.95%
Reflective method invocation	67,958	48.07%
ClassLoader usage	15,580	11.02%
Total	141,372	100%

TABLE I
Absolute Numbers and Proportions for Characteristics and Behaviors of Android Applications

V. Limitations

The static analysis code mapping can assist software developers to understand what permission(s), if any, are required for each Android API call. Our tool does not perform a reachability analysis to ensure that any of the permission-protected API calls are actually reachable in the code. Therefore, it may introduce false positives in terms of permissions required when unreachable code is present in an application. The static code analysis does not cover dynamic and foreign bytecode including Java Native Interface calls which can generate false negatives in terms of permissions required.

VI. Conclusion

We downloaded more than 141,000 applications, and we performed analysis on a sizeable portion of the Android market. Although the Android Market is constantly adding new applications, we have provided a significant snapshot of the Android Market which details the prevalence of certain characteristics and behaviors of applications. We also identified hundreds of additional permission-protected API calls to further the capability of accurately assessing the appropriate permission set for an application. Our results show that the majority of the developers are not using the strictest permission set based on their code functionality but rather over-specify their permission requirements which increases the attack surface and causes additional security risks from an application breach.

References

[1] Android Market. https://market.android.com

[2] Au, K., Zhou, B., Huang, Z., Gill, P., Lie, D. Short Paper: A Look at SmartPhone Permission Models. In Proceedings of the ACM Workshop on Security and Privacy in Mobile Devices (SPSM), 2011.

[3] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In Proceedings of the 18th ACM conference on Computer and Communications Security, CCS 11, pages 627-638, New York, NY, USA, 2011.

[4] android-market-api - Android Market for all Developers! http://code.google.com/p/android-market-api/

[5] android-apktool - A tool for reengineering Android apk files. http://code.google.com/p/android-apktool/

[6] M. Schroeder and J. Saltzer. The protection of information in computer systems. Proceedings of the IEEE, 63(9), pages 1278-1308, 1975.

[7] Chan, J.T. and Yang, W., Advanced obfuscation techniques for Java bytecode, Journal of Systems and Software 71, No. 2. pages 1-11, 2004.