Zero Permission Android Applications - Attacks and Defenses

Veelasha Moonsamy, Lynn Batten
School of Information Technology
Deakin University
Melbourne, Australia
v.moonsamy@research.deakin.edu.au, lynn.batten@deakin.edu.au

Abstract

Google advertises the Android permission framework as one of the core security features present on its innovative and flexible mobile platform. The permissions are a means to control access to restricted APIs and system resources. However, there are Android applications which do not request permissions at all.

In this paper, we analyze the repercussions of installing an Android application that does not include any permission and the types of sensitive information that can be accessed by such an application. We found that even applications with no permissions are able to access sensitive information (such the device ID) and transmit it to third-parties.

Keywords: android, application, permission.

1. Introduction

Permission systems were introduced in the early days of computing when desktop computers were regarded as an emerging technology [18]. Traditionally, permissions were used to assign file access rights (for example: read, write) to users and also to regulate access to lower-levels of the operating system (for example as superuser). The operating system keeps a list – *Access Control List* (ACL) – in order to document the permission rights for each user who has access to the machine.

Google implemented a similar idea in its Android mobile operating system. Android includes a Linux kernel (and its libraries) which serves as its base operating system, a Dalvik virtual machine, an application middleware layer and lastly, a set of system applications. Each application is assigned its own user ID (UID) and is executed in individual sandboxes. Even though the applications' executions are separated, Android allows inter-application communication, provided that the correct levels of permissions have been assigned. Permissions are also required to access restricted system resources, such as the contact list. Whilst most applications do contain permissions, some might not necessarily make use of them; it depends on their functionalities.

This work investigates the consequences of installing an application that does not request any permission and determines the sensitive information such an application can have access to. The rest of the paper is organized as follows: In Section 2, we describe Android permissions in further detail and also present some relevant work in the Android permission area; we describe the potential attacks by a zero permission application on a device and possible defenses against such attacks in Section 3. In Section 4, we provide a discussion around zero permission applications and lastly, we give our conclusions and ideas for future work in Section 5.

2. Android Permissions

Android applications are available for download/purchase from Google's official market, Google Play [12], as well as from multiple third-party markets. The applications are available in .APK (application package file) format, which is essentially an optimized version of Java code. Each APK file includes the application's code (.dex files), multimedia resources, certificates, and the AndroidManifest.xml where the permissions are defined. The .xml file includes <*uses-permission*> and <*permission*> tags to allow developers to request permissions. The <*uses-permission*> tag is used if the developer needs to request any permission that has been predefined by Google. Currently, there are 130 official Android permissions running on Android 4.0. The other <*permission*> tag allows developers to define customized permissions in their applications.

Permissions can be classified into four types [11]: *Normal*, *Dangerous*, *Signature* and *SignatureorSystem*. Normal permissions do not require the user's approval but they can be viewed after the application has been installed. Dangerous permissions are displayed to the user and require confirmation in order to proceed with the installation process; these permissions have access to restricted resources and can have a negative impact if used incorrectly. Permissions classified under the

Signature category are granted automatically, provided that the requesting application is signed with the same certificate as the application that declared the permission. Finally, SignatureorSystem permissions are granted to those applications that have the same certificates as the Android system image.

Android adopts an 'install-time' permission granting policy [2]. In this case, once the application is downloaded the user will have to acknowledge the permission request in order to be able to use the application. Once the permissions are granted, they cannot be revoked unless the application is completely uninstalled. The authors in [1, 6, 15, 20] have developed methodologies that can provide users with more flexibility over the permissions once they are granted.

In [7], Enck et al. provided some insight into the Android security model by demonstrating the use of permissions during install-time and through inter-component communication. In their work, the authors explained that whilst application developers are allowed to control access to restricted resources by defining permissions in the AndroidManifest.xml file, they can also regulate inter-component communications. In regard to the latter, Enck et al. mentioned that developers can prevent other applications from accessing an application's components by either explicitly assigning permissions to those components or declaring the components to be private instead of public.

The work of Chaudhuri et al. [4] focused on designing semantics that can be used to formulate abstract representation of a particular application. The authors argued that their work can reveal the integrity of an application before the user installs it. This is useful in cases where third-party applications are required to interact with the components of those applications that come pre-installed on an Android phone.

The Android Permission framework allows developers to use Google's predefined set of permissions or generate their own, depending on the requirements of the applications being developed. Shin et al. [21] found a flaw in the customized permission scheme: since developers are not required to follow any naming conventions while defining their own permissions, the authors pointed out that this can introduce conflicted permissions. They implemented a legitimate banking application and a malicious application, both sharing a customized permission of the same name, to demonstrate how the privileges for the rogue application escalated by exploiting the vulnerability.

Felt et al. [8] and Bartel et al. [3] developed tools that can assist both users and developers to assess the integrity and reliability of applications before they are installed or uploaded on the market.

Nevertheless, it should be noted that if an application does not require access to any of the restricted system resources, the developer can choose not to include any permission in the AndroidManifest.xml file. In the next section, we will elaborate further on this particular case.

3. Potential Attacks and Defenses

In this section, we investigate the possible negative ramifications of installing an application with no permission by conducting a manual investigation of the application and also the defense mechanisms that are present in the literature to counter this issue.

3.1 Attacks

In one of his online posts, Brodeur [16] pointed out that an application that does not request any permission can scan the external storage directory and return a list of applications and files that are stored on the SDcard. This information can be accessed at /sdcard/. This vulnerability is rather predictable as Google grants read-only access to the SDcard on any device irrespective of the OS version that it is running on.

Additionally, a similar exploit can be carried out on the internal memory by scanning the /data/system/packages.list folder to retrieve the list of package names of those applications that are installed on the device. Furthermore, in the same folder, the file packages.xml stores the shared permissions of all applications on the device along with their corresponding UIDs.

The aforementioned vulnerabilities can be exploited by enticing the user to download a decoy application which can give the attacker access to a backdoor on the victim's device, as explained by Cannon in [22]. Cannon used an active remote shell to interact with the device and therefore had access to the data on the device. Furthermore, Brodeur demonstrated that the absence of INTERNET permission does not stop an attacker from exporting the captured data onto an external server. In fact, the URI_ACTION_VIEW Intent can initiate a Web browser call and thus allow the attacker to transfer data via the Internet. This is a common technique used by advertising libraries designed to leak device ID or subscriber ID in order to carry out targeted advertising, as observed in [14].

3.2 Analysis of the Brodeur Application

We downloaded and installed Brodeur's zero permission application on two versions of Android – 2.3 and 4.0. We set up an Android emulator, which is provided by the Android SDK and executed the application in question. At first glance, the installed application behaves as any other clean applications. We then start a logcat filter, running in parallel with the emulator, to monitor the execution behaviors of the zero permission application.

The application itself includes three buttons – which can be customized depending on the nature of the attack. Each button carries out an attack and sends the information to an external party via the URI_ACTION_VIEW Intent, as explained in sub-section 3.1. When a user clicks on the first button, the application will read any information stored on the SD card and transfer it to the attacker. The second button allows the attacker to retrieve the application package names that are installed on the user's device. Finally, the third button returns two important unique identifiers, which are the device's SIM and GSM operator identifier.

Once the aforementioned information is gathered, an attacker can then carry out a targeted attack to exploit a particular user's device. Moreover, a zero permission application can be used to identify vulnerable Android users which will ensure the longevity of the attack, and afford a lower risk of being discovered or reported to the authorities.

3.3 Defenses

It is well-known [9] that the best defense against any security attack is achieved by raising user awareness. In the case of Android permissions, users need to be educated on how to interpret them. A study conducted by Felt et al. [9] showed Android users do not fully comprehend the permissions requests presented to them during install-time. Even more alarming is the fact that some users do not even take the trouble to read the permission descriptions and simply proceed to install the application. (This type of user behavior is very common when dealing with End-User License Agreements). Therefore, since smartphone users do not value the importance of permission requests, their absence is hardly noticed.

In addition to training, users may rely on metrics before proceeding to download and install applications. User ratings, reviews, number of downloads are some of the available resources that can help users to assess whether or not the application is popular or has caused other users any problems.

The solution proposed by Chin et al. [5] is also an effective way to quickly assess the integrity of an application before it is installed on the device. The authors developed a tool, ComDroid, that can detect potential vulnerabilities in Android applications. In order to do so, ComDroid first disassembles the APK file and then parses through the code to track the communication flow between components. However, it should be noted that this method is valid only for applications that are downloaded from third-party markets and not the ones from Google Play.

4. Discussion

In this section, we give our viewpoints on zero permission Android applications.

In its official documentation [13], Google states that *"a basic Android application has no permissions associated with it, meaning it cannot do anything that would adversely impact the user experience or any data on the device"*. For instance, an example of such an application could be a customized calculator application to calculate tips to be given for good service at an eatery. Based on the description provided by Google, the aforementioned application should not be able to access sensitive/restricted information on a device. However, as demonstrated in Section 3, the current security vulnerabilities within the Android platform can allow a zero permission application to access restricted information.

Furthermore, we believe that imposing rules stipulating that all applications must request at least one permission will not necessarily solve the problem. The literature shows that application that request unnecessary permissions cause far more damage than a zero permission application [17, 19]. Moreover, an over-privileged application can also compromise the functionalities of other applications stored on the device; the impact of this attack is far beyond the scope than the ones mentioned in Section 3.

Additionally, even though Google considers the Android permission system as one of the key security features of the platform, it should be noted that users' responses to understanding and approving permission requests have compromised the security standards of Android. The work by Felt et al. [10] demonstrates the lack of comprehension from Android users when it comes to understanding and interpreting permission requests upon installing an application. In fact, a majority of users do not even read the list of permissions and simply proceed to download and install the application. This shows that users regard the permission system as a hassle and therefore will not be concerned even if an application does not request any permission.

5. Future Work

In this paper, we have given an overview on Android permissions and analyzed and elaborated on the implications of zero permission applications. We also presented some measures that users can apply to identify vulnerable applications. In summary, user understanding and user behavior are the key aspects that can mitigate the propagation of rogue applications.

As future work, we will provide a third-party service where Android users can perform a quick scan of applications that have been downloaded from Google Play and return the user a report, assessing the integrity of the applications before installation.

6. References

[1] H. Banuri, M. Alam, S. Khan, J. Manzoor, B. Ali, Y. Khan, M. Yaseen, M. Tahir, T. Ali, Q. Alam, and X. Zhang, "An android runtime security policy enforcement framework," *Personal and Ubiquitous Computing*, pp. 1–11, 2010, 10.1007/s00779-011-0437-6. [Online]. Available: http://dx.doi.org/10.1007/s00779-011-0437-6

[2] D. Barrera, J. Clark, D. McCarney, and P. van Oorschot, "Understanding and improving app installation security mechanisms through empirical analysis of android," in *Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*. ACM, 2012.

[3] A. Bartel, J. Klein, M. Monperrus, and Y. Traon, "Automatically securing permission-based software by reducing the attack surface - an application to android," *Arxiv preprint arXiv:1206.5829*, 2012.

[4] A. Chaudhuri, "Language-based security on android," in *Proceedings of the ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security*, ser. PLAS '09. New York, NY, USA: ACM, 2009, pp. 1–7. [Online]. Available: http://doi.acm.org/-10.1145/1554339.1554341

[5] E. Chin, A. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Procs. of the 9th Annual International Conference on Mobile Systems, Applications, and Services*, 2011.

[6] M. Conti, V. Nguyen, and B. Crispo, "Crepe: Context-related policy enforcement for android," in *Information Security*, ser. Lecture Notes in Computer Science, M. Burmester, G. Tsudik, S. Magliveras, and I. Ilic, Eds. Springer Berlin / Heidelberg, 2011, vol. 6531, pp. 331–345, 10.1007/978-3-642-18178-8_29. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-18178-8_29

[7] W. Enck, M. Ongtang, and P. McDaniel, "Understanding android security," *Security Privacy, IEEE*, vol. 7, no. 1, pp. 50 –57, jan.-feb. 2009.

[8] 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*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 627–638. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046779

[9] A. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," UC Berkeley, Tech. Rep. UCB/EECS-2012-26, 2012.

[10] A. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," 2012.

[11] Google. Android Permissions, available at http://developer.android.com/guide/topics/manifest/permission-element.html.

[12] Google. Google play, available at https://play.google.com/.

[13] Google. (2012, Oct) Android Security - Using Permissions, http://developer.android.com/guide/topics/security/permissions.html.

[14] V. Moonsamy, M. Alazab, and L. Batten, "Towards an understanding of the impact of advertising on data leaks," *to appear in 'International Journal of Security and Networks (IJSN)'*, 2012.

[15] M. Nauman and S. Khan, "Design and implementation of a fine-grained resource usage model for the android platform," 2010.

[16] Paul Brodeur (Leviathan Security Group), "Zero-permission android applications, available at http://leviathansecurity.com/blog/archives/17-zero-permission-android-applications.html," April 2012.

[17] P. Pearce, A. Felt, G. Nunez, and D. Wagner, "Addroid: Privilege separation for applications and advertisers in android," in *Proceedings of AsiaCCS*, 2012.

[18] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38 –47, Feb 1996.

[19] S. Shekhar, M. Dietz, and D. Wallach, "Adsplit: Separating smartphone advertising from applications," *Arxiv preprint arXiv:1202.4030*, 2012.

[20] W. Shin, S. Kiyomoto, K. Fukushima, and T. Tanaka, “A formal model to analyze the permission authorization and enforcement in the android framework,” in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, aug. 2010, pp. 944 –951.

[21] W. Shin, S. Kwak, S. Kiyomoto, K. Fukushima, and T. Tanaka, “A small but non-negligible flaw in the android permission scheme,” in *Policies for Distributed Systems and Networks (POLICY), 2010 IEEE International Symposium on*, july 2010, pp. 107 –110.

[22] T. Cannon (from Via Forensics), “No permission android app gives remote shell, available at https://viaforensics.com/security/nopermission-android-app-remote-shell.html,” December 2011.