Global Scientific JOURNALS

GSJ: Volume 5, Issue 9, September 2017, Online: ISSN 2320-9186
www.globalscientificjournal.com

A SURVEY OF ANDROID PERMISSION PROBLEM

Omid Movaghar

Abstract

Android security has been a hot spot recently in both academic research and public concerns due to numerous instances of security attacks and privacy leakage on Android platform. Mobile devices get smarter and increasingly provide access to sensitive data. Smart phones and tablet computers present detailed contact information, e-mail messages, appointments, and much more. Users often install apps on their devices to get additional functionality like games, or access to social networks. Too often, such apps access sensitive data and take privacy less serious than expected by users. In this survey, we discuss the existing Android security permission and existing security enforcements solutions between 2010_2015 and try to classify works and review their functionalities. As a result, mobile users are left to decide for themselves whether an app is safe to use.

Keywords: Android, Security, Permission, Attack

1. INTRODUCTION

Android is a Linux-based operating system for mobile devices like smart phones and tablet computers. It is developed by the Open Handset Alliance led by Google. Android apps can be downloaded from online stores like Google's app store Google Play (formerly Android Market) and also from third-party sites. People use smartphones for many of the same purposes as desktop computers: web browsing, social networking, online banking, and more. Smartphones also provide features that are unique to mobile phones, like SMS messaging, constantly-updated location data, and ubiquitous access. As a result of their popularity and functionality, smartphones are a burgeoning target for malicious activities. Current mobile devices offer a large amount of services and applications than those offered by personal computers. At the same time, the increasing number of security threats that target mobile devices has emerged. In fact, malicious users and hackers are taking advantage of both the limited capabilities of mobile devices and the lack of standard security mechanisms to design mobile-specific malware that access sensitive data. Android smartphones are protected by a permission based framework which restricts third-party applications' accesses to sensitive resources such as SMS database and external storage in Android smartphones. The accesses to sensitive resources may lead to money loss. For example, Android malware may send premium rate messages, make premium rate calls, and generate large amount of network data without users' acknowledgment. Such permission based framework is criticized as coarse-grained. Many applications tend to request much more per- missions than necessary. In most cases, a user has to either grant all permissions an application requests or abort the installation process, instead of granting the permissions one by one. At install time, a user is shown with a list of permissions which an application requests. The user must either grant or deny all of these permissions together. After the user approves the permission request and installs the application, the application owns its permissions throughout its lifetime and it does not need to request them again at run-time. Android controls Inter-Component Communication (ICC) through a reference monitor. The reference monitor provides a Mandatory Access Control (MAC) enforcement on how applications access components by evaluating whether the applications are granted with necessary permissions. In this paper, we Survey the behavior of current mobile permission, and discuss about the feature of them.

2. Android Applications Architecture

In this section we describe the architecture of the Android and its applications. Android is being developed and maintained by Google and promoted by the Open Handset Alliance (OHA). Android is placed on top of the Linux kernel and it includes the middleware, libraries and APIs written in c language, and application software running on an application framework which includes Java-compatible libraries. Android's source code is released by Google under open source licenses. Android operating system is a stack of software components, which is roughly divided into five sections and four main layers. Application layer is located at the top of the Android software stack. These comprise both the pre- installed apps provided with a particular Android implementation and third-party apps developed by individuals (unofficial) app developers. Examples of such apps are Browser, Contacts Manager, and Email apps. More examples of such applications can be found from many official and unofficial app markets.

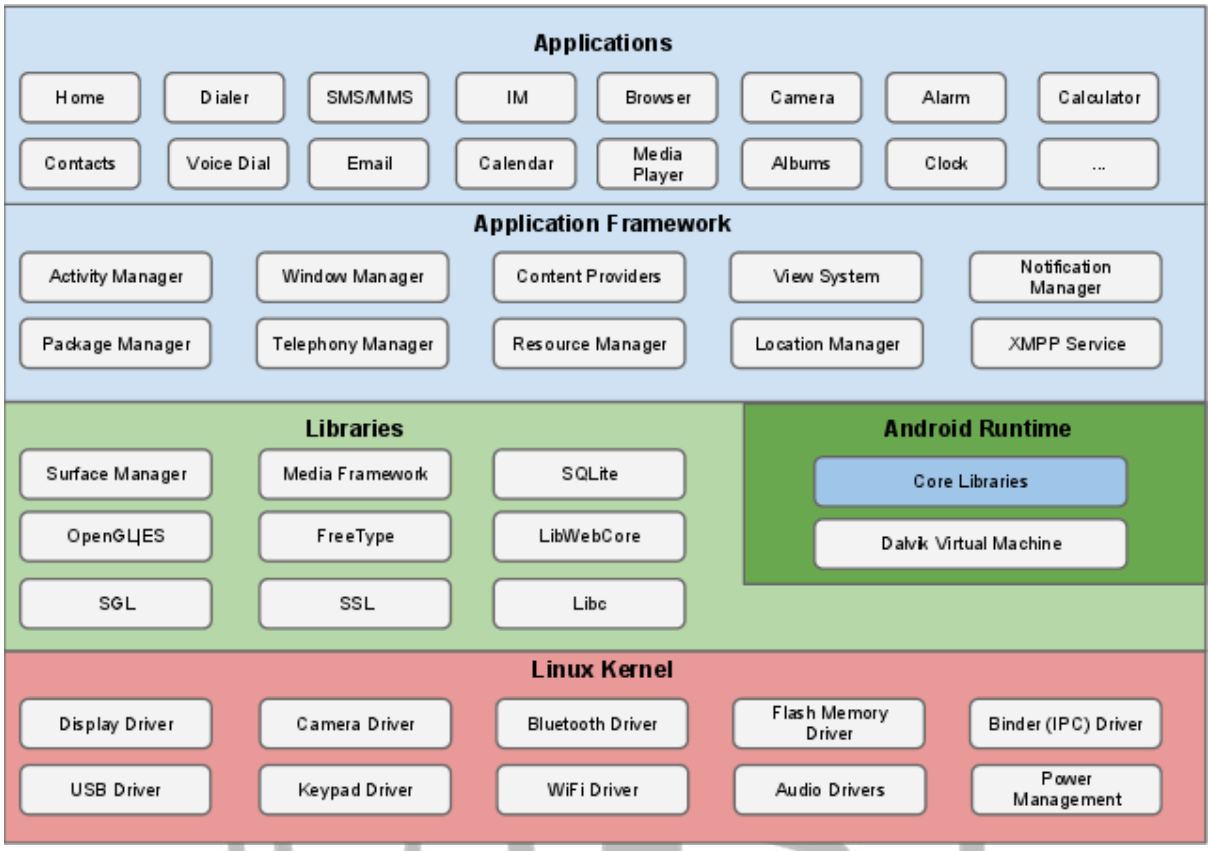

Figure 1: Android operating system architecture

Linux kernel

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access.

Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android. The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of

the Dalvik virtual machine. The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

3. Security Measures

A game may, for example, need to activate vibration but should not need to read messages or access contact information. After reviewing the permissions, users can decide whether to install an application. Protected resources include camera, location data (GPS), Bluetooth, telephony, SMS/MMS, and network/data connections. Granted permissions are applied to applications as long as they are installed. Android's permissions are some form of Mandatory Access Control, or MAC for short. In contrast to DAC which stands for Discretionary Access Control, access is not controlled by users or by user ids, but rather by permission labels that are assigned system functions. Accessing a resource requires the call of system functions. If an application wants access to a resource, it needs the permissions required by the appropriate system functions .Smartphone operating system vendors use curated markets and/or application permissions to protect users. Smartphone operating systems may also protect users by requiring user consent before an application can access sensitive information or dangerous capabilities. User-approved permissions can alert users to the activities of gray ware or malware, although malware may seek to circumvent permission systems. Permissions do not prevent the installation of personal spyware because the attacker can grant all necessary permissions during installation.

Analysis of Android security issues

Android has been designed with security in mind from the very inception with the aim to protect user data, apps, the device and the network [30]. However, overall security depends on the developers' willingness and capability to employ best practices. Also, user must be aware of the effect that some app can have after installation, on its data and device's security. Anti-malware solutions on Android cannot handle malware aggressively due to security model enforced on apps. For example, anti-malware apps have limited scanning and/or monitoring capability for other apps or file-system in the device.

Coarse granularity of permissions

Although Android defines 130 permissions, most of them are of coarse granularity. Especially, the INTERNET permission the READ_PHONE_STATE permission and the WRITE_SETTINGS permission are coarse-grained as they give an application arbitrary accesses to certain resources [1]. Taking the INTERNET permission as an example, the INTERNET permission allows an application to send HTTP(S) requests to all domains, and connect to arbitrary destinations and ports [2].As a result, the INTERNET permission provides insufficient expressiveness to enforce control over the Internet accesses of the application.

Incompetent permission administrators

Unfortunately, both developers and end-users usually lack professional knowledge. In addition, the developers and end- users may have conflict of interest (Han et al., 2013). When a developer writes a manifest file requesting permissions for his or her application, the developer may not know in detail what is at risk for end-users if the application is granted with these permissions [3]. While some enthusiastic developers might take time to learn what each of the 130 permissions does and request them appropriately, other developers might choose to simply over-claim permissions so as to make sure their applications work anyway.

Insufficient permission documentation

Google Inc. provides a great deal of documentation for Android application developers, but the content on how to use permissions on Android platform is limited. As investigated by Felt et al., the lack of permission usage information may lead to developers' errors. In Android 2.2 documentation, permission requirements are provided for 78 methods; however,[2].'s test reveals permission requirements for 1259 methods, which is a sixteen-fold improvement over the documentation. The documentation lists additional permissions in several class descriptions, but it is not clear which methods of the classes require the stated permissions. Moreover, six errors are identified in Android permission's documentation. The insufficient and imprecise permission information confuses Android application developers, who may write applications with guesses, assumptions and repeated tries. Consequently, this leads to defective applications which become threats with respect to security and privacy of Android users[4].

Over-claim of permissions

Over-claim of permissions is probably the most severe threat to Android security. It directly breaks the principle of least privilege (PLP)[5]. This violation of PLP exposes users to potential privacy leakage and financial losses. For example, if a standalone game application requests for the SEND_SMS permission which is unnecessary, the permission can be exploited to send premium rate messages without users' acknowledgment[2]. identified that 56% of the over-privileged applications have only one extra unnecessary permission and 94% have 4 or fewer extra permissions.

Permission escalation attack

In contrary to the general belief that the damage imposed by an Android malware is limited to an application's sandbox, the permission escalation attack allow a malicious application to collaborate with other applications so as to access critical resources without requesting for corresponding

permissions explicitly[6]. In addition, the collusion attack can be carried out by multiple applications in generating a joint set of permissions which enables them to perform an unauthorized or malicious actions[7]. The collusion attack can be further classified by the way applications communicate with each other, into direct collusion attack, where applications communicate directly, and indirect collusion attack where applications communicate via a third application or component in be- tween. The indirect collusion attack usually involves another application or component as a mediation which can provide either overt channels or covert channels. Overt channels, such as buffers, files and I/O devices, use a data object as the entity to hold certain information. In other words, the entity is an object that is normally viewed as a data container[8].

TOCTOU attack

The vulnerability of TOCTOU (Time of Check to Time of Use) exists in Android mainly due to naming collusion. No naming rule or constraint is applied to a new permission declaration. Malicious application developers may exploit this flaw[9]. Suppose a malicious developer manages to trick a user into installing malicious application A which declares permission P', and another malicious application B which requests permission P'. The name of permission P' is the same as permission P which protects accesses to a critical resource. Afterward the user uninstalls application A and installs benign application C which declares permission P. Now the malicious application B would be able to use permission P0 to access the critical resource. According to Shin et al., this TOCTOU flaw exists in Android 1.5, 1.6, 2.0 and 2.1, on both emulators and actual devices.

Important feature	
Stowaway	Detect over-claim of permission
PScout	Detect over-claim of permission
Apex	Allow revoking of over-claimed
MockDroid	Allow revoking of over-claimed permissions
AppFence	Allow revoking of over-claimed permissions
TISSA	Allow revoking of over-claimed permissions
Flex-P	Allow revoking of over-claimed permissions

Figure 2: Comparison among the countermeasures to over-claim of permissions**.**

4. Enhanced designs and implementations

Some important work has been done to deal with these problems is referenced below.

Countermeasures to over-claim of permissions

Detection of over-claim of permissions. As for detecting over- claim of permissions in Android applications [2]. manually reviewed the top free and top paid applications from 18 Google Play categories in 2011 [2]. For each of the applications, Felt et al. compared its functionalities with the permissions it requested by exercising its user interface. Four out of 36 applications were over-privileged, while the INTERNET permission accounted for three of the over- privileged applications [11].

Finer-grained permissions

Finer-grained permissions can be used to address the issues of coarse-grained permissions. Finer-grained permission implementations can be categorized into install-time policy enforcement and run-time policy enforcement[12]. The install- time policy enforcement aims to stop malicious applications from being installed on user's devices. It provides finer policy enforcement at install-time rather than asking users to decide whether to grant all permissions or nothing. More factors, such as permission combinations, permission and action string combinations, and signatures of requesting applications, are taken into consideration. If an application violates one of the preset policies, the installation of the application would be aborted. On the other hand, the run-time policy enforcement allows users to set finer-grained restrictions at run-time[12]. proposed Secure Application INTeraction (Saint) in 2009 whose install-time policies regulate the granting of application defined permissions[11]. In addition to Android's protection level-based permission granting policy, an application can define the conditions under which the permissions defined by the application can be granted to other applications at install- time.

Countermeasures to permission escalation attack

Demonstration of permission escalation attack. In 2011[13]. presented Soundcomber, a Trojan with few and innocuous permissions that can extract targeted private information from the audio sensor of a phone. Defense mechanisms. Along with presenting Soundcomber, [13]. proposed a defense mechanism[13] which maintains a list of critical numbers and blocks all applications from accessing the audio data during a sensitive phone call[12].

Facilitating permission administration

Three roles are usually involved in the Android permission administration: developers declare which permissions the application will request application marketers verify whether the application is legitimate or not by an automatic tool or manual review users decide whether to approve the permission requests[14]. These three roles are usually performed by those who are not well-trained in policy based management. In 2012, Google announced that a service named Bouncer had been deployed on Google Play Store (Google, 2012). Bouncer scans Google Play automatically for potentially malicious applications without disrupting the user experience of Google Play Store or requiring developers to go through an application approval process.

5. Permission-Granting Mechanisms

Permission systems can grant permissions to applications using four basic mechanisms: automatic granting, trusted UI, runtime consent dialogs, and install-time warnings. We discuss these permission-granting mechanisms in order of preference, based on the amount of user attention that they consume.

Automatic Grant

An automatically-granted permission must be requested by the developer, but it is granted without user involvement. We propose that permissions should be automatically granted if they protect easily-revertible or low-severity permissions. For example, changes to the global audio settings are easily revertible, and vibrating the phone is merely annoying. Currently, any web site can play audio or generate pop-up alerts without re- questing permission from the user; although this can lead to annoyance, the web is still usable. Web users simply exit web sites that have unwanted music or alerts.

Trusted UI

Trusted UI elements appear as part of an application's workflow, but clicking on them imbues the application with a new permission. To ensure that applications cannot trick users, trusted UI elements can be controlled only by the platform. Trusted UI has been incarnated in many forms, including CapDesk [15], access control gadgets , and sensor-access widgets[16]. Trusted UI elements require effort to design because they need to fit applications' workflows and accommodate as much functionality as possible. They also con- strain the appearance of applications, so their design needs to be neutral enough to fit most applications. To help trusted UI blend into applications, the platform could allow some degree of customization or allow developers to choose between multiple designs. For ex- ample, developers could choose the size, placement, or color scheme of an element. Ideally, their design would involve input from application developers.

Install-Time Warnings

Install-time permission warnings integrate permission granting into the installation flow. Installation screens list the application's requested permissions. In some platforms (e.g., Facebook), the user can reject some install-time permissions. In other platforms (e.g., Android and Windows 8 Metro), the user must approve all requested permissions or abort installation[17].

6. Some drawbacks of the Android permission system

There are some drawbacks of the Android permission system.

Static permissions

Android's permission system is rather rigid and lacks flexibility. Users can only install applications by granting all permissions requested by that application. It is not possible to withdraw any permission, neither during installation nor after the installation process. The only option users have is to uninstall an application[18].

Missing control

Users have no control over their resources. Once an application has been installed, it can access resources with the permissions that have been granted during installation. Users cannot neither watch which resources an application accesses, nor can they permit or deny any such access[18].

Over-privileged applications

Applications sometimes are over-privileged, which means they require access to resources they do not need to function. Over-privileged applications increase the impact of vulnerabilities[18].

Permission granularity

Some standard permissions are defined at a coarse granularity, e.g., INTERNET, WRITE_EXTERNAL_STORAGE. Applications with the permission INTERNET have arbitrary access to the Internet. There is no way to restrict access for example to specific domains or services[18].

7. RELATEDWORK

Previous studies of Android applications have been limited in their understanding of permission usage. Our permission map can be used to greatly increase the scope of application analysis[12]. apply Fortify 's Java static analysis tool to decompiled applications; they study their API use. However, they are limited to studying applications' use of a small number of permissions and API calls. In a recent study[2]. manually classify a small set of Android applications as over privileged or not, but they were limited by the Android documentation. provide a tool that performs an over privilege analysis on application source code. Their tool could be improved by using our permission map; theirs is based on the limited Android documentation. Our static analysis tool also performs a more sophisticated application analysis. Unlike their Eclipse plugin, Stowaway attempts to handle reflective calls, Content Providers, and Intents.

Sarma et al. proposed to better inform policy administrators whether the risks of installing an application was commensurate with its expected benefit. Specially, Sarma et al. proposed to capture the benefit of an application by using the category and sub-category of the application, and capture the risk by using the usage percent- age of the permissions among the applications in the same category. When a user sees a warning triggered by an application, he or she should be more cautious about the risk of the application being installed. When a developer sees a warning triggered by his or her application, he or she should consider how to make the application avoid triggering the signal.

In 2012, Fragkaki designed and implemented Sorbet, which allows developers to define policies to mitigate undesired in- formation flows and confused deputy attack. Sorbet tracks the permissions of all components on a call stack. When a component A is called, and A is protected by the permission P, Sorbet evaluates if every component on the call stack has P.

In 2012, Wei et al. applied Stowaway to a set of 237 evolving third party applications covering 1703 versions. The result showed that the overall tendency was towards over-claim of permission (Wei et al., 2012). In particular, 19.6% of updated versions of applications were over-privileged due to added permissions, and 25.2% of applications were initially over- privileged and stayed over-privileged during their evolutions. On the other hand, 11.6% of applications resorted from over-privileged to legitimate.

8. CONCLUSION AND FUTURE WORK

The security issues and countermeasures of Android systems have been rigorously studied since the first Android device was shipped to the market. In recent years, the problem of Android security has become even more severe, partially due to the vulnerabilities in the design of Android systems, and partially due to the huge success of Android devices in the market. Motivated to provide a systematic overview of the current research on Android security, we identify six issues in Android security, including coarse granularity of permissions, incompetent permission administrators, insufficient permission documentation, over-claim of permissions, permission escalation attack, and TOCTOU attack. The former three are indirect issues, while the latter three are direct issues, which may lead to financial loss or privacy leakage directly. In order to empower all end-users with flexible permission control, it will be necessary to include more flexible permission control into the regular Android system. Based on comprehensive analysis on the issues and countermeasures, we argued that Android security can be improved by developing data-driven tools to strengthen Android framework, and maintaining consistency between application intentions and implementations.

As future work, we can raise users awareness of permissions and a summary of our relationship with permissions. Another option is to write applications that will encrypt and decrypt all private data as they are accessed in order to assure its confidentiality, also we will examine Android application family classification more extensively and also investigate the implications of data leakages in benign applications. We will also extend DroidBox to provide more detailed API monitoring. Increasing ways to improve “permission gap” Can be an important contribution to malware detection. Security API enables users to install the apps and if the built-in security of Android is not able to prevent the unauthorized access of critical data, then this enhanced security framework will provide necessary safeguards. We identified a set of issues that are impeding awareness and comprehension. In particular, category headings are confusing, some users cannot connect resource-based warnings to risks, some users cannot reason about the absence of permissions, and some users are experiencing warning fatigue. We provide a set of recommendations to address these issues.

REFERENCES

[1] Barrera D, Kayacik HG, van Oorschot PC, Somayaji A. A methodology for empirical analysis of permission-based security models and its application to android. In: Proc. of ACM CCS. ACM; 2010. pp. 73e84.

[2] Felt AP, Ha E, Egelman S, Haney A, Chin E, Wagner D. Android permissions: user attention, comprehension, and behavior. In: Proc. of SOUPS. ACM; 2012. p. 3.

[3] Han W, Lei C. A survey on policy languages in network and security management. Comput Networks 2012;56(1):477e89.

[4] Vidas T, Christin N, Cranor L. Curbing android permission creep. In: Proc. of the Web, vol 2; 2011.

[5] Saltzer JH. Protection and the control of information sharing in multics. Commun ACM 1974;17(7):388e402.

[6] Bugiel S, Davi L, Dmitrienko A, Fischer T, Sadeghi A, Shastry B. Towards taming privilege-escalation attacks on android. In: Proc. of NDSS; 2012.

[7] Dietz M, Shekhar S, Pisetsky Y, Shu A, Wallach D. Quire: lightweight provenance for smart phone operating systems.

[8] Schmidt A,Schmidt H, ClausenJ,YukselK,Kiraz O, Camtepe A, et al. Enhancing security of linux-based android devices. In: Proc. of 15th International Linux Kongress.Lehmann;2008.

[9] Shin W, Kwak S, Kiyomoto S, Fukushima K, Tanaka T. A small but non-negligible flaw in the android permission scheme. In: IEEE POLICY. IEEE; 2010. pp. 107e10.

[10] Fragkaki E, Bauer L, Jia L, Swasey D. Modeling and enhancing android's permission system. In: Computer SecurityeESORICS 2012. Springer; 2012. pp. 1e18.

[11] Nauman M, Khan S, Zhang X. Apex: extending android permission model and enforcement with user-defined runtime constraints. In: Proc. of ACM ASIACCS. ACM; 2010.

[12] Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification. In: Proc. of ACM CCS. ACM; 2009. pp.

[13] Schlegel R, Zhang K, Zhou X, Intwala M, Kapadia A, Wang X. Soundcomber: a stealthy and context-aware sound trojan for smartphones. In: Proc. of NDSS; 2011. pp. 17e33.

[14] Han W, Fang Z, Yang LT, Pan G, Wu Z. Collaborative policy administration. IEEE TPDS 2013;24(1):1.

[15] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan. User-driven access control: Rethinking permission granting in modern operating systems. In Proceedings of the IEEE Symposium on Security and Privacy, 2012.

[16] J. Howell and S. Schechter. What you see is what they get: Pro- tecting users from unwanted use of microphones, cameras, and other sensors. In Proceedings of the Web 2.0 Security & Privacy Workshop (W2SP), 2010.

[17] R. B¨ ohme and S. K¨ opsell. Trained to accept? A field experiment

on consent dialogs. In Proceedings of the ACm Conference on Human Factors in Computing Systems (CHI), 2010.

[18] M.KEN MICRO FOCUS Borland Software Corporation Linz, Austria