Volume 5, Issue 9, September 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Survey: Inter-App Permission Leakage on Android Devices

Nitesh Arun Patil[*], **Dr. K. V. Kulhalli**
Department of Information Technology,
Dr. D. Y. Patil College of Engineering,
Kolhapur, Maharashtra, India

Abstract— New mobile operating systems supports third party developed apps which might be developed intentionally to grab private information of the user. Currently many scenarios have proved that because of inter app messaging mechanisms in android operating system; the user privacy is at risk. This paper presents a survey on inter app permission leakage on android device. Furthermore android app permission mechanism and permission categories also highlighted.

Keywords: Android Security, Android Permissions, Inter-app permission, User centric risk.

I. INTRODUCTION

All Today the boom of the android phones, the users are using more and more applications in almost all sectors such as health, entertainment, office, college, banking etc. Android device activations are hitting near about 1.5 million per day and unfortunately, privacy leakage issues in android devices are also increasing in same flow. In app store, there are many applications those are free but they are relying on advertisement for their income. These applications can get a user's private information easily and use it for target advertisements. Users also accept this business model, but they are unaware about his private information is being leaked by certain applications without his permission.

Android system provides sharing of data and services between apps using inter-app communication system. Android permission system controls access of resources of the mobile device. Hence permissions can be misused intentionally so enforcing permissions is not enough to prevent from permission violations. Android's enforcement of the permissions is at the level of individual apps, allowing multiple malicious apps to collude and combine their permissions or to trick vulnerable apps to perform actions on their behalf that are beyond their individual privileges [1].

Application components are basic logical building blocks of Android apps. Each component can run individually, either by its embodying application or by system upon permitted requests from other applications. Android apps have four types of components: (1) Activity components provide the basis of the Android user interface. Each Application may have multiple Activities representing different screens of the application to the user. (2) Service components provide background processing capabilities, and do not provide any user interface. Playing music and downloading a file while a user interacts with another application are examples of operations that may run as a Service. (3) Broadcast Receiver components respond asynchronously to system-wide message broadcasts. A receiver component typically acts as a gateway to other components, and passes on messages to Activities or Services to handle them. (4) Content Provider components provide database capabilities to other components. Such databases can be used for both intra-app data persistence as well as sharing data across applications [1].

Privacy violations can occur even when a user grants access to protected data (e.g. contact list, exact location, etc.) to a benign app, i.e. one not trying to violate user's privacy. This holds true, since the app may either be used as a confused deputy [14, 15], i.e. accidentally allowing other malicious apps to use its functionality to access the resources, or be bundled with a malicious advertisement library [16], which misuses the shared permissions to violate user privacy. Also, benign Android apps tend to request more permissions than needed for their intended functionality [13].

II. LITERATURE SURVEY

Hamid Bagheri, Alireza Sadeghi, et. al[1] presents novel approach, called COVERT, for compositional analysis of Android inter-app permission leakage vulnerabilities. COVERT's analysis is modular to enable incremental analysis of applications as they are installed, updated, and removed. It statically analyzes the reverse engineered source code of each individual app and extracts relevant security specifications in a format suitable for formal verification.

Alexios Mylonas, Marianthi Theoharidou, and Dimitris Gritzalis [2] provides taxonomy of user data found on a smartphone, their respective Android permissions and discussed ways to disclose their data. They have identified privacy threats applicable to user data, crawled apps from Google Play and used this sample to list descriptive statistics for permission combinations that may violate user privacy.

Li Li, Alexandre Bartel, et. al, [3]. In this paper, authors have proposed IccTA, a static taint analyzer to detect privacy leaks among components in Android applications. IccTA goes beyond state-of-the-art approaches by supporting inter-component detection. By propagating context information among components, IccTA improves the precision of the analysis. IccTA outperforms existing tools on two benchmarks for ICC-leak detectors: DroidBench and ICC-Bench.

Li Li, Alexandre Bartel, et. al [4], In this paper, author presented potential component leaks (PCLeaks), a tool to exploit potential component leaks and PCLeaksValidator, a tool which automatically generates a correspond malicious apps to validate the results of PCLeaks. Concretely, PCLeaks first builds a precise control-flow graph for the analyzed apps. Then, it performs static taint analysis with a well-defined set of source and sink methods to identify potential active component leaks and also potential passive component leaks.

Drago S, Michael G. Burke, Salvatore Guarnieri, [5], Author has identifed three types of inter-application Intent-based attacks that rely on information flows in applications to obtain unauthorized access to permission-protected information. Two of these attacks are of previously known types: confused deputy and permission collusion attacks. The third attack, private activity invocation, is new and relies on the existence of dificult-to-detect misconfigurations introduced because Intents can be used for both intra-application and inter-application communication. Such misconfigured applications allow protected information meant for intra-application communication to leak into unauthorized applications. This breaks a fundamental security guarantee of permissions systems: that application can only access information if they own the corresponding permission.

Yajin Zhou, Xuxian Jiang [6] focuses on the Android platform and aim to systematize or characterize existing Android malware. Particularly, with more than one year effort, they have managed to collect more than 1,200 malware samples that cover the majority of existing Android malware families, ranging from their debut in August 2010 to recent ones in October 2011. In addition, they systematically characterize them from various aspects, including their installation methods, activation mechanisms as well as the nature of carried malicious payloads.

Franziska Roesner and his team [7] has taken the approach of user-driven access control, where permission granting is built into existing user actions in the context of an application, rather than added via manifests or system prompts. To allow the system to precisely capture permission-granting intent in an application's context, they introduce access control gadgets (ACGs). Each user-owned resource exposes ACGs for applications to embed. The user's authentic UI interactions with an ACG grant the application permission to access the corresponding resource. Their prototyping and evaluation experience indicates that user driven access control enables in-context, non-disruptive and least-privilege permission granting on modern client platforms.

Yi Ying Ng, Hucheng Zhou, et. al [8] presents a comprehensive study on the trustworthy level of top popular Android app stores in China, by discovering the identicalness and content differences between the APK files hosted in the app stores and the corresponding official APK files. First, they have selected 25 top apps that have the highest installations in China and have the corresponding official ones downloaded from their official websites as oracle; and have collected total 506 APK files across 21 top popular app stores (20 top third party stores as well as Google Play). Afterwards, APK identical checking and APK difference analysis are conducted against the corresponding official versions. Next, assessment is applied to rank the severity of APK files. All the apps are classified into 3 severity levels, ranging from safe (identical and higher level), warning (lower version or modifications on resource related files) to critical (modifications on permission file and/or application codes). Finally, the severity levels contribute to the final trustworthy ranking score of the 21 stores.

Yury Zhauniarovich, Olga Gadyatskaya and Bruno Crispo [9] present how to enable the deployment of application certification service, we called TruStores, for the Android platform. In their approach, the TruStore client enabled on the end-user device ensures that only the applications, which have been certified by the TruStore server, are installed on the user smartphone. They envisage trusted markets (TruStore servers, which can be, e.g., corporate application markets) that guarantee security by enabling an application vetting process. The TruStore infrastructure maintains the open nature of the Android ecosystem and requires minor modi_cations to Android stack.

Parvez Faruki, Ammar Bharmal and his team [10] discussed the Android security enforcement mechanisms, threats to the existing security enforcements and related issues, malware growth timeline between 2010 and 2014, and stealth techniques employed by the malware authors, in addition to the existing detection methods. This review gives an insight into the strengths and shortcomings of the known research methodologies and provides a platform, to the researchers and practitioners, toward proposing the next-generation Android security, analysis, and malware detection techniques.

III. PERMISSION CATEGORIES

Most device access in android is controlled by permissions. Applications can define their own extra permissions, but here the permissions defined by Android OS are considered only. There are 134 permissions in Android2.2. Permissions are categorized into following threat levels

Normal: API calls with annoying but not harmful consequences are protected with Normal permissions. Example: accessing information about available Wi-Fi networks, vibrating the phone, and setting the wallpaper.

Dangerous: API calls with potentially harmful consequences. Example: Opening a network socket, recording audio, and using the camera.

Signature: The most sensitive operations are protected with Signature permissions. These permissions are only granted to applications that have been signed with the device manufacturer's certificate. Example: Ability to inject user events.

SignatureOrSystem: This category includes signed applications and applications that are installed into the/system/app folder. Example: Preinstalled applications, applications protecting the ability to turn off the phone. During installation permission prompt is displayed to the user for Dangerous permissions. Warnings are categorized according to functionality. For example, Dangerous location related permissions are included in location related warning. Normal permissions are hidden in a collapsed menu. Signature/System permissions are not shown at all.[11] Following table shows available android permission groups with respective permissions.

Table 1: Android Permissions

Permission Group	Permission
android.permission-group.CALENDAR	android.permission.READ_CALENDAR android.permission.WRITE_CALENDAR
android.permission-group.CAMERA	android.permission.CAMERA
android.permission-group.CONTACTS	android.permission.READ_CONTACTS android.permission.WRITE_CONTACTS android.permission.GET_ACCOUNTS
android.permission-group.LOCATION	android.permission.ACCESS_FINE_LOCATION android.permission.ACCESS_COARSE_LOCATION
android.permission-group.MICROPHONE	android.permission.RECORD_AUDIO
android.permission-group.PHONE	android.permission.READ_PHONE_STATE android.permission.CALL_PHONE android.permission.READ_CALL_LOG android.permission.WRITE_CALL_LOG com.android.voicemail.permission.ADD_VOICEMAIL android.permission.USE_SIP android.permission.PROCESS_OUTGOING_CALLS
android.permission-group.SENSORS	android.permission.BODY_SENSORS
android.permission-group.SMS	android.permission.SEND_SMS android.permission.RECEIVE_SMS android.permission.READ_SMS android.permission.RECEIVE_WAP_PUSH android.permission.RECEIVE_MMS android.permission.READ_CELL_BROADCASTS
android.permission-group.STORAGE	android.permission.READ_EXTERNAL_STORAGE android.permission.WRITE_EXTERNAL_STORAGE
android.permission-group.CALENDAR	android.permission.READ_CALENDAR android.permission.WRITE_CALENDAR

IV. ANDROID PERMISSION MECHANISM

Android application consists of distinct UIDs for all apps and permissions which are back bones of Android's security. Each app of android is executing with its Linux process and it gets unique user and group ID (UID) at the time of installation. This allows for protection of memory and file system resources. Communication and resource sharing are subject to access restrictions enforced via a fine-grained permission mechanism. Applications are allowed access to resources if they are granted the respective permissions by the user. The isolation of applications, called sandboxing, is enforced by the kernel, not the Dalvik VM. Java as well as native apps run within a sandbox and are not allowed to access resources from other processes or execute operations that affect other apps. Applications must declare required permissions for such resources within their manifest file. These permissions are granted or denied by the user during the installation of the application. The user does not deny or grant permissions during the runtime of the application. Permissions are enforced during the execution of the program when a resource or function is accessed, possibly producing an error if the app was not granted the respective permission. The Android system defines a set of permissions to access system resources such as for reading the GPS location, or for inter-application/process communication. Additionally, apps may define their own permissions that may be used by other apps. There is no central point for permission enforcement, it is scattered over many parts of the Android system. At the highest level, if permissions are declared in an application's manifest file for a component, they are enforced at access points to that component.

These are calls to startActivity() or bindService() for activities or services, respectively, that would cause security exceptions to be thrown if the caller is not granted the required permission. Permissions may control the delivery of broadcast messages by restricting who may send broadcasts to a receiver or which receivers may get the broadcast.

In the first case a permission for the protected receiver is declared in the manifest file (or when registering the receiver programmatically, respectively). It gets enforced after a sender's call to sendBroadcast() and will not cause an exception to be thrown. Rather, the message will simply not be delivered to that receiver if the sender does not have the required permission. A sender, on the other hand, may declare a permission within the sendBroadcast() call, which will also be enforced without the sender noticing.

Permissions for granting read or write access to Content Providers are declared within the manifest file. Apart from that, content providers allow for a finer grained access control mechanism, via URI permissions. They control access to subsets of the content provider's data, allowing a content provider's direct clients to temporarily pass on specific data elements (identified by a URI) to other applications. A dedicated flag on an Intent indicates that the recipient of the Intent is granted permission to the URI in the Intent's data field, identifying a specific resource, such as a single address book entry. The granted URI permission is finally enforced once the recipient of the Intent queries the content provider holding the URI by calling on a ContentResolver object. Content Providers declare support for URI permissions in the manifest file. Enforcement of URI permissions results in security exceptions being thrown if the caller does not have the required permissions.[12]

V. CONCLUSION

During this survey, it was observed that there is need of more study on detection of inter-app permission leakage with any dynamic strategy. This paper also focuses on a huge demand of new solution for user centric risks control and give requisite information to the user about app behaviour with user centric risks available into it.

REFERENCES

[1] G. Hamid Bagheri, Alireza Sadeghi, Joshua Garcia and Sam Malek, "COVERT: Compositional Analysis of Android Inter-App Permission Leakage", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2015.

[2] Alexios Mylonas, Marianthi Theoharidou, and Dimitris Gritzalis, "Assessing Privacy Risks in Android: A User-Centric Approach", DOI: 10.1007/978-3-319-07076-6_2, Springer International Publishing Switzerland 2014.

[3] Li Li, Alexandre Bartel, Tegawende F. Bissyand,"IccTA: Detecting Inter-Component Privacy Leaks in Android Apps", 37th International Conference on Software Engineering (ICSE 2015), ITALY,2015.

[4] Li Li, Alexandre Bartel, Jacques Klein, Yves le Traon, "Automatically Exploiting Potential Component Leaks in Android Applications", IEEE Conference, Sept 2014.

[5] Drago S, Michael G. Burke, Salvatore Guarnieri, "Automatic Detection of Inter-application Permission Leaksin Android Applications", IEEE Conference, Nov.2013 .

[6] Yajin Zhou, Xuxian Jiang, "Dissecting Android Malware: Characterization and Evolution", Security and Privacy (SP), IEEE, May 2012.

[7] Franziska Roesner, Tadayoshi Kohno, Alexander Moshchuk, Bryan Parno, "User-Driven Access Control: Rethinking Permission Granting in Modern Operating Systems", IEEE ,2012.

[8] Yi Ying Ng , Hucheng Zhou , Zhiyuan Ji , Huan Luo, "Which Android App Store Can be Trusted in China?", Computer Software and Applications Conference (COMPSAC), IEEE, July 2014.

[9] Yury Zhauniarovich, Olga Gadyatskaya, and Bruno Crispo, "Trustore: Implementing A Trusted Store For Android", DISI - Via Sommarive 5 - 38123 Povo - Trento (Italy), May 2014.

[10] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, "Android Security: A Survey of Issues,Malware Penetration, and Defenses", IEEE Communication Surveys & Tutorials, Vol. 17, No. 2, 2015.

[11] Hari H. Rajai, Prof. Sachin Bojewar, "Study Of Permissions And Risk Communication Mechanisms In Android", International Research Journal of Engineering and Technology (IRJET),2015

[12] Clemens Orthacker, Peter Teufl, Stefan Kraxberger, G¨unther Lackner, Michael Gissing,"Android Security Permissions – Can we trust them?" Springer Berlin Heidelberg,2012

[13] Felt, A., Chin, E., Hanna, S., Song, D., Wagner, D., "Android permissions demystified.", Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 627–638. ACM (2011).

[14] Felt, A., Hanna, S., Chin, E., Wang, H.J., Moshchuk, E. "Permission redelegation: attacks and defenses" In: 20th Usenix Security Symposium (2011).

[15] Grace, M., Zhou, Y., Wang, Z., Jiang, X. "Systematic detection of capability leaks in stock Android smartphones.", Proceedings of the 19th Network and Distributed System Security Symposium (2012).

[16] Pearce, P., Felt, A.P., Nunez, G., Wagner, D. "Android: privilege separation for applications and advertisers in android.",Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, pp. 71–72. ACM (2012)