Preventing Collusion Attack in Android

Iman Kashefi[1], Maryam Kassiri[2], and Mazleena Salleh[1]
[1]Department of Computer Science, Universiti Teknologi Malaysia, Malaysia
[2]Department of Computer and Information Technology, Islamic Azad University, Iran

Abstract: *Globally, the number of Smartphone users has risen above a billion, and most of users use them to do their day-to-day activities. Therefore, the security of smartphones turns to a great concern. Recently, Android as the most popular smartphone platform has been targeted by the attackers. Many severe attacks to Android are caused by malicious applications which acquire excessive privileges at install time. Moreover some applications are able to collude together in order to increase their privileges by sharing their permissions. This paper proposes a mechanism for preventing this kind of collusion attack on Android by detecting the applications which are able to share their acquired permissions. By applying the proposed mechanism on a set of 290 applications downloaded from the Android official market, Google Play, the number of detected applications which potentially are able to conduct malicious activities increased by 12.90% in compare to the existing detection mechanism. Results showed that there were 4 applications among the detected applications which were able to collude together in order to acquire excessive privileges and were totally ignored by the existing method.*

Keywords: *Android security, collusion attacks, colluding applications, over-privileged applications.*

Received March 10, 2013; accepted May 6, 2014; published online August 16, 2015

1. Introduction

Today smartphones are ubiquitous; by providing various services and different functionalities, smartphones have become the inseparable part of people's life. Google Android is the most famous smartphone platform and could possess the first rank in mass-production of application development [18]. Android has dominated smartphone market by attracting most phone manufacturers, carriers, and developers to produce their services and applications [24]. The number of Android users grows tremendously [19]. According to a recent report released by mobile security firm Lookout, the Android Market is growing at three times the rate of Apple's App store [4]. There are currently near 1.5 million Android apps in the market with a huge number of downloads each day [10]. Dissimilar to Apple, Google has no mechanism in auditing applications published in market [10]. Therefore, from time to time, it may need to remove malicious applications from the market after they are proved to contain malware [4] or when enough people registered complaints for an application [16]. Moreover, since everyone who has registered as an Android developer has permission to upload his/her application to Android market, it has been changed to a potential place for attackers to fulfill their malicious intentions [3].

The occurrence of attacks reveals that Android's permission framework has some vulnerability which is targeted by attackers. Currently, Android Operating System (OS) is limited to promote users to review and approve the permissions requested by an application at install time [13]. The most common attacks are perpetrated by applications which misuse critical permissions that are approved by users. Unfortunately most of the users are unaware of critical security issues, while the major responsibility of maintaining the security of the device in right level is left to end-users. Disregarding the security practices may cause leaking out the user sensitive information and misusing the device and user's properties in different ways.

For instance, if a user installs an application that have access to user's location information, he/she is not sure whether the data is being used in a proper way or the application sends it to a remote server for advertising reasons or even malicious purposes. In other words, users blindly trust that application and suppose that the application use them properly.

Unfortunately recent researches [2, 6, 7, 10, 13, 15, 16, 23] showed that currently there are various applications with different malicious purposes uploaded in market and users are attracted by their splendid advertisements. These applications have been developed with malicious purposes such as leaking user sensitive information [25], calling to per-minute telephone numbers to overcharge users, and disturbing the normal function of the device. Each of these malicious behaviors may impose severe harms to users.

Attacks resulted from granting excessive privileges to the applications and lack of effective auditing on application development in Google Play, along with the significant role of the end-users with no or little security knowledge have imposed serious harms to user's privacy, data, and properties. As the result, this area has been changed to a point of concern for security experts. In order to detect these attacks and mitigate the consequent malicious actions, several researches have been conducted, and different approaches have been proposed.

One of the famous existing works, entitled Kirin , is an extension to Android installer and has been proposed by Enck *et al*.[3, 8, 22].It addresses the problem by comparing the required permissions of an application with a set of predefined policy rules and makes users aware of excessive privileges requested by the application which may be used to conduct malicious activities.

Although, Kirin could principally prevent these kinds of attacks, is not able to detect applications which are capable of sharing their permissions in order to acquire excessive privileges; in other words, Kirin checks the granted permissions to a single application rather than a sandbox [3]. Android uses sandboxing as a mechanism to isolate apps' process in order to restrict the interference of the applications [10]. By checking the required permissions declared in the manifest file of each application, it is possible to detect over-privileged applications, but the problem emerges when two or more malicious applications developed by the same author try to pass this checking system separately and then collude together in order to share their acquired permissions. These applications individually request for few critical permissions which are not enough for conducting a malicious activity, therefore they are not considered as over-privileged applications. However, after installation, they are able to increase their privileges through using an Android security mechanism which facilitates the interaction between applications with the same author.

This paper introduces a comprehensive mechanism for detecting over-privileged applications. This mechanism not only detects individual over-privileged applications, but also a group of applications which acquire excessive privileges through sharing their permissions. Moreover, in order to address different kinds of attacks mentioned above, some new rules has been defined and added to this mechanism. The results of applying the proposed mechanism on downloaded applications from the Android official market, Google Play, has been discussed based on the different categories of applications in the market. This paper makes the following contributions:

- It proposes a method for detecting over-privileged applications which acquire excessive privileges through sharing their permissions by using a same shared-user-id.
- Moreover, it enhances the coverage of the existing detection mechanism; Kirin, by adding some more rules in order to detect different kinds of malicious activities.

The remainder of this paper is organized as follows: Section 2 describes the Android architecture, security mechanisms and permission framework, section 3 elaborates the proposed mechanism for detecting applications which are potentially capable of conducting colluding attack and also proposes the new rules in order to enhance the detection of over-privileged applications. Results are presented and discussed in section 4 and finally the work is concluded in section 5.

2. Android

Android is an open source software stack for portable devices that includes an OS, middleware, and key applications [12]. Android is based on Linux and presents critical system functionalities like security management, memory management, process management, and network stack. In the Android conceptual model, the kernel layer is supposed to be between the hardware and the rest of the software layers to provide core functionalities for Android services [12]. The middleware layer includes native Android libraries (written in C/C++), Android runtime module and an application framework. Accordingly, the application framework encompasses applications written in C/C++ or Java that exclusively serve for system purposes.

Android applications are developed as integration of four primary components. The components of one application may or may not be able to communicate with one, some, or all of the components in another application. These components are: Activities, services, content providers, and broadcast receivers. "Activities" present the user interfaces (or screens) of an application; "Services" control backbone processing and they are hidden to the user; "Content Provider" components are the preferred method of sharing data between applications; and "Broadcast Receivers" are implemented in the form of mailboxes to receive messages from other applications. Applications are enabled to broadcast messages to an implicit or explicit destination. In explicit broadcasting the message is sent to a specific component while, in implicit broadcasting, "Broadcast Receivers" which subscribe to receive such messages are able to receive them [3, 17, 21].

There is a communicating mechanism called Inter-Component Communication (ICC) mechanism which facilitates the interaction of one component with other components of the application or with components of other applications. This mechanism is fully provided through the middleware. Applications commence ICC channels by sending a specific message entitled Intent. Intents are responsible for encapsulating the information relevant to the ICC call [17, 18].

Intents may be sent explicitly to named components or implicitly using a named action string [11]. In the action string, the required action, relevant data as argument for the action, component category that should manage the intent and some extra fields to define different required data are precisely specified [18]. Android will redirect implicit intents to appropriate components automatically through

checking the intents with the intent filters of the components. Components use intent filters to subscribe to specific action strings. Intent filters associated with individual components of applications should be included in the manifest file. The so called manifest file introduces crucial details of the application to the Android OS. These details are very critical to the system and are evaluated by the Android installer. The manifest file declares the set of permissions that an application requests along with other useful details about the permissions and the way of accessing to the components of an application form other apps.

In the following of this section, an overview of the core security mechanisms of Android which are sandboxing, application signing, and permission framework will be presented [3, 5, 21].

2.1. Sandboxing

Android is a privilege-separated OS. Each application is isolated from other applications and placed within its own distinct system identity and its own Dalvik Virtual Machine (DVM). System files are accessible by either the "system" or "root" user. Accordingly, an application can only access its own files or files of other applications that are unprotected and publicly available. This provides a sandbox for each application which isolates it from other applications and from the system [4].

2.2. Application Signing

The Android security mechanism obliges all developers to digitally sign their applications with a certificate and the private key should be held by them. There is no necessity to acquire these certificates from an authority. The Android only employs the certificate as a means of identifying the author of the application so that, it will be able to launch reliable connections between applications of the same author [14].

2.3. Permission Framework

Not only Android provides security measures in kernel layer but also it considers application level security in the permission framework and it mainly limits special actions that an application is allowed to perform [14]. This mechanism ensures that an application has no permission to perform operations that adversely impact other applications, the OS, or the user [4]. To use Android resources and share data, an application needs to declare the permissions in its manifest file. A permission is somehow a plain text that can be defined by Android or application developers [9]. There are about 100 built-in permissions in Android [22] which restrict access to the Android components and manage operations such as making phone calls, using internet, writing SMS, and so forth. To acquire permissions, the user will be prompted at install time to approve the application's requested permissions. Moreover, each component of an application can be protected by permission. In this way, such components are only available to other components of the same application, or the components of other applications which have already acquired the related permission.

3. Proposed Mechanism

In order to access Android components and consequently use Android core services, applications need to acquire related permissions. Many of the Android components provide critical services for applications in order to perform their purposes properly, however misusing this protected services may impose serious risks on user's privacy and sensitive information. Applications must declare the needed permissions for performing their functionalities between <uses-permission> tags in the manifest file. These permissions are granted to the applications after user approval at install time and cannot be revoked once they are donated. As it is mentioned in section 1, in android official market, Google Play, there is no effective audit on publishing applications and consequently no constraint on requesting permissions by applications. Many applications request permissions more than what they basically require [1]. Therefore, in this way malicious applications can be published in the market and acquire several critical permissions through user approval and conduct dangerous attacks on victim's smartphone.

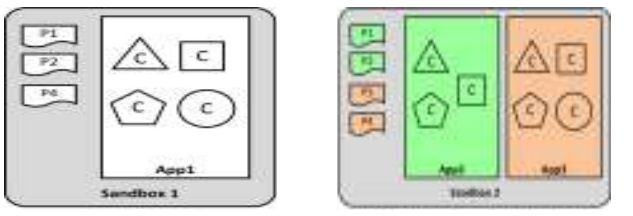
In order to make users aware of the possibility of conducting malicious activities by the installed app, this mechanism checks all the requested permissions of the application. This checking process retrieves the declared permissions in the manifest file and compares them with the predefined set of rules based on the mechanism proposed by Enck *et al*.[8]. If the requested permissions of an application matches to one of the rules, the mechanism determines that the examined app has excessive privileges and gives an alert to the users in order to help them make up their mind whether to use the application or not. The rules are combinations of different critical Android standard permissions that together can give a malicious application the opportunity of conducting dangerous activities.

Despite the fact that the detection mechanism proposed by Enck *et al*. [8] can principally prevent attacks resulted from excessive privileges acquired by a single application [3], still it is not able to detect excessive privileges that may be acquired by two or more applications through sharing their permissions. These permissions can be used in order to conduct a collusion attack. The flaw of this mechanism is that each of these applications can pass the permission test separately, while the union of their obtained permissions may match one of the security rules.

As it was explained in section 2, Android is based on the Linux and benefits from a privilege separation mechanism by giving each application a unique User ID, separate memory space and resources and giving them a virtual isolated environment called sandbox. After approving the requested permissions of an application by the user, android assigns these permissions to the application's sandbox. All components of the application inherit the permissions granted to the sandbox in order to access the Android components and use its core services.

Applications can also ask Android to place them in a common sandbox to share same resources, same User ID and consequently same permissions. Sharing resources and permissions facilitates the functionality of applications which are developed by the same author and need communicating with each other. In order to use this facility, applications should declare the same "sharedUserId" in their manifest file and should be signed by the same author.

Although, this Android mechanism eases the interaction of applications of the same origin, it can be misused by malicious applications. In this way, colluding applications can bypass the permission checking mechanism individually and gain more privileges by permission sharing in order to conduct malicious activities. Figure 1 illustrates 3 applications which 2 of them have shared a User ID and are placed in a common sandbox. Each application has its own components which enable it to conduct different activities and provide various services. The components of an application can also be protected by a permission. So, in order to access to Android core services and protected components of other apps, each application must acquire the related permissions which are shown in the left side of each sandbox. In Figure 1, sandbox is denoted by S, app by A, component by C, and permission by P. If $P(S_i)$ represents the set of permissions granted to the sandbox S_i; $P(A_i)$, the set of permissions acquired by the application A_i; and $P(C_{Ai})$, permission set of the components of application A_i which is inherited form the relevant sandbox, then $P(A_1)$={$P1$, $P2$, $P4$}, $P(A_2)$={$P1$, $P2$}, and $P(A_3)$= {$P3$, $P4$}.



a) App1 and its sandbox.

b) App2 and App3 share their permissions in a common sandbox.

P1: Permission
C: Component

Figure 1. Applications and sandboxes.

Components of App2 and App3 inherit the permissions granted to their common sandbox which is the union of the permissions obtained by App2 and App3:

$$P(C_{A2}) = P(C_{A3}) = P(S_2) = P(A_2) \cup P(A_3) = \{P1, P2, P3, P4\} \quad (1)$$

Both of the applications could increase their privileges by sharing the permissions acquired at install time. In order to depict the functionality of the proposed mechanism, it is assumed that rule R is one of the predefined rules for detecting over-privileged apps. With regard to the scenario presented in Figure 1, rule R is defined as: R={$P1$, $P2$, $P4$} which states: "if an application could acquire permission $P1$, $P2$, and $P4$, it is considered as an over-privileged application and potentially is able to perform malicious actions", in other words:

$$if \ [R \cap P(A_i) = R] \quad (2)$$

Then A_i is over-privileged.

In order to compare the effectiveness of the proposed mechanism with the existing one, first the results of applying each mechanism on the presented scenario are discussed, then the checking process and techniques used in the proposed mechanism are explained in detail. The result of evaluating the three applications illustrated in Figure 1 with the Kirin mechanism which checks the acquired permissions of an individual application, will be:

$$[R \cap P(A_1) = \{P1, P2, P4\} = R] \quad (3)$$

Therefore A_i is over-privileged.

$$[R \cap P(A_2) = \{P1, P2\} \neq R] \quad (4)$$

Therefore A_2 is not over-privileged.

$$[R \cap P(A_3) = \{P4\} \neq R] \quad (5)$$

Where A_i is not over-privileged.

By applying the proposed method and considering the granted permissions to each sandbox, different results are obtained:

$$[R \cap P(A_1) = \{P1, P2, P4\} = R] \quad (6)$$

Therefore A_i is over-privileged.

$$[R \cap P(A_2) = R \cap P(S_2) = \{P1, P2, P4\} = R] \quad (7)$$

Therefore A_2 is over-privileged.

$$[R \cap P(A_3) = R \cap P(S_2) = \{P1, P2, P4\} = R] \quad (8)$$

Therefore A_3 is over-privileged.

As it is observed, App2 and App3 which could bypass the first mechanism, were detected as over-privileged applications by the proposed mechanism.

Figure 2 demonstrates the flowchart of the detection process used in this mechanism.

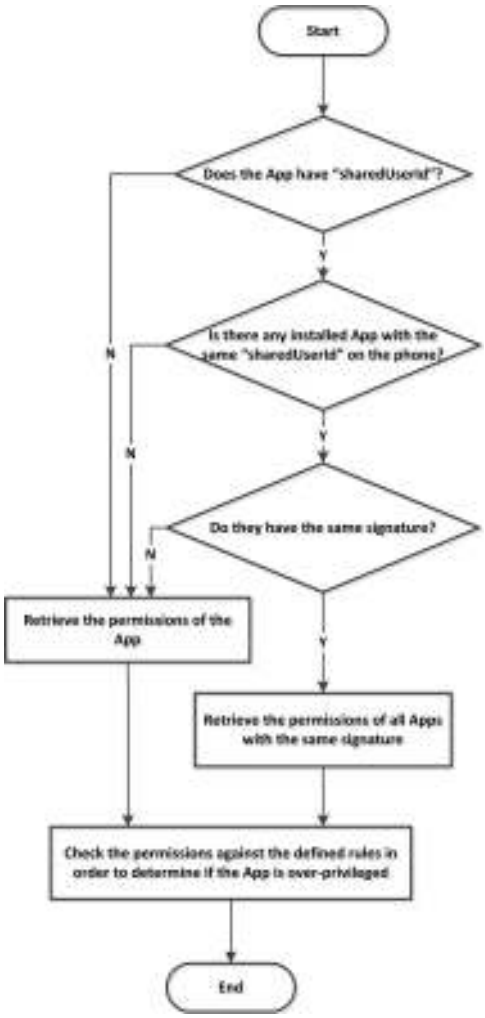


Figure 2. The algorithm of detecting applications which are able to collude together.

The first step to prevent colluding attack perpetrated by applications with the same author is to detect the applications that are able to be run in a common sandbox. In order to share a sandbox, applications need to declare the same 'sharedUserId' in their manifest file. Therefore, upon installing the application, this mechanism checks whether the application has declared the 'sharedUserId' in its manifest file or not, if it is so, it searches for any other applications installed on the phone with the same 'sharedUserId'. As it was mentioned in section 2, only applications that are signed by the same author can use this facility. Therefore, as the second step, the similarity of the signatures of the applications with the same 'sharedUserId' are checked.

In case of the similarity of the signatures, requested permissions of these applications which are declared as uses-permissions in the manifest file are retrieved. Then, the union of the permissions acquired by the applications is compared against the rules in order to detect the applications with excessive privileges. These over-privileged applications are potentially capable of conducting malicious activities by misusing their acquired permissions.

In addition to the rules proposed by Kirin, stated in Table 1, five more rules were defined in order to address different attacks which are not covered by the existing policies.

Existing policies can be classified in a set of rules which address malicious activities like location tracking, voice call eavesdropping, tampering with the incoming SMS or sending SMS spam, and the ability of debugging other applications, while new rules address spying through video and audio recording, accessing to the personal data, and sniffing the received data through MMS. Description of the new rules is stated in Table 2.

Table 1. Rules proposed by Kirin.

Rule Number	Rule Description
1	This rule protects against app debugging by third-party applications.
2	This rule protects against eavesdropping. Malicious applications may determine the phone number and device IDs, record a call and send them to a remote server by acquiring a set of permissions including recording audio and access to internet.
3	This rule protects against eavesdropping and calls intercepting. By acquiring a set of permissions including processing outgoing call, recording audio, and access to internet, malicious applications may eavesdrop a call or monitor, redirect and prevent outgoing calls.
4	This rule protects against location tracking. Malicious Apps may determine the user's exact location and send the acquired information to a remote server if they obtain a set of permissions including access to precise location and internet.
5	This rule protects against location tracking. By acquiring a set of permissions including access to approximate location and internet, malicious applications may determine the user's approximate location and send it to a remote server.
6	This rule protects against interacting with SMS. Malicious apps may monitor, edit or delete messages sent to user's device without showing them to user by obtaining a set of permissions including receiving and writing SMS.
7	This rule protects against interacting with SMS. By obtaining a set of permissions including sending and writing SMS, malicious applications may monitor, edit or delete messages sent to user's device without showing them to user and may send multiple messages to premium numbers and charge the user.

Table 2. Rule proposed by this work.

Rule Number	Rule Description
8	This rule protects against interacting with MMS. Malicious apps may monitor multimedia messages sent to user's device and send them to a third party for malicious purposes by acquiring a set of permissions including receiving and sending MMS.
9	This rule protects against misusing the calendar data. By obtaining a set of permissions including reading calendar data and access to internet, malicious applications can access the calendar sensitive data and send them to a remote server.
10	This rule protects against misusing the contact list data. By acquiring a set of permissions including reading the contact list data and access to internet, malicious applications may misuse the contact list sensitive data and send them to a third party.
11	This rule protects against spying. Malicious applications may take pictures and video without the user awareness and send them to a remote server for malicious purposes by obtaining a set of permissions including access to camera and internet.
12	This rule protects against spying and eavesdropping. Malicious apps may record audio without the user awareness and send it to a third party for malicious purposes if they acquire a set of permissions including recording audio and access to internet.

The new rules have been constructed based on the methodology proposed by Enck *et al*. [8]. Moreover, the lists of critical permissions introduced by Sarma *et al*. [20] have been considered in defining these rules. The statistics provided by them represents the level of the criticality of each component and the related permission based on the incidence of using these permissions by benign and malicious applications. The new defined rules enhance the detection of over-privileged applications by addressing different kinds of attacks. Though, there are still more attacks that are not

addressed by these policies and should be considered in the same way in future.

4. Results and Discussions

In order to evaluate the proposed method, a sample set of 290 applications from different categories of the Android official market, Google Play, was downloaded. With the purpose of obtaining more accurate results, the sample set of apps was selected randomly from the most popular apps of the different categories in the Android market. Both existing and proposed mechanisms were implemented and installed on the android emulator along with the downloaded apps. Android APIs were used in order to retrieve the required information of the installed apps such as the information declared in the manifest file and also the signatures. The set of rules stated in Table 1 and 2 were employed in order to detect the applications that are potentially able to perform malicious activities. First, the existing method proposed by Enck *et al*. [8] was applied on the installed apps by the application implemented for this purpose and then, the same set of applications were examined by the proposed method exactly in the same way. Results showed 12.90% increase in the number of applications detected by the proposed method. It proves that among the tested apps, there are applications with normal privileges which are able to share permissions and become over-privileged. As it is shown in the Figure 3, the number of applications with the capability of voice call eavesdropping has been increased by 27.27%. These applications which were detected based on the rule number 2 and 3, were not able to conduct voice call eavesdropping individually, while they became able to do so after sharing their permissions. A similar result obtained when the proposed mechanism checked the application set by applying rule number 4 and 5. Therefore, 10.53% increase in the number of applications which are potentially able to perform location tracking was observed. The results also showed 25.00% increase in the number of applications which are able to tamper with incoming SMS or send SMS spam (rule number 6 and 7), 8.00% in the number of applications capable of sharing users personal data with the third party (rule number 8, 9, and 10) and 12.5% in the number of applications capable of spying through capturing video or audio (rule number 11 and 12). All these applications which were detected by the proposed mechanism as potentially dangerous apps, could easily pass the existing checking mechanism. The acquired permissions of each of them did not match to any of the mentioned rules in the Tables 1 and 2, while the overall permissions obtained by these applications after installation, matched to the defined rules. Figure 3 depicts the growth of the number of detected applications based on the types of the attacks.

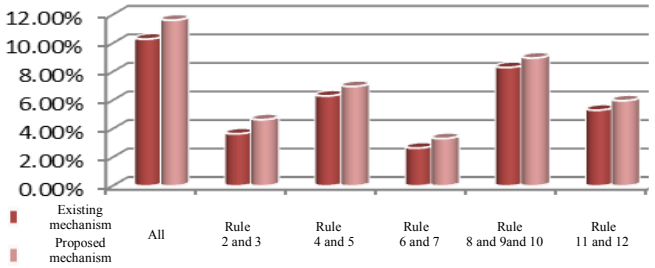


Figure 3. Detected apps before and after applying the proposed method

The applications detected by the proposed mechanism has been totally ignored by Kirin while they are potentially able of conducting serious attacks to user's privacy and sensitive information in Android devices. The significant increase in the number of detected applications shows that considering the applications which are able to collude together in detecting over-privileged applications in Android is highly important.

Moreover, among the 290 examined applications, 45 of them were matched the new rules which showed that the detection of over-privileged applications has been expanded to a wide variety of possible attacks.

As it is observed in the obtained results, comparing the signatures of the applications with the same 'sharedUserId' which is used in this mechanism is a definite method to determine the applications of the same origin which are able to collude together in order to acquire excessive privileges. However, misusing the unprotected components of applications with different authors through sending explicit intents is not addressed by this method. In other words, if it is assumed that the author of an application becomes aware of the detailed functionality of another application's components, then there would be a probability of misusing an application with different author through sending explicit intents to its unprotected components.

During the evaluation process and applying the proposed detection method on different categories of the Google Play, it was observed that the percentage of detected applications varies in each categories. Investigation in the results proved that applications in some categories usually require more permissions to function properly than the other categories and also in some cases they follow the same permission patterns which conforms with the hypothesis that applications in one category probably have similar requirements in order to fulfill their purposes. Figure 4 demonstrates the number of detected over-privileged applications among 20 categories of the Google Play. Results shows that categories like "Communications", "Music and Audio" and "Social" usually are more sensitive to the defined rules.

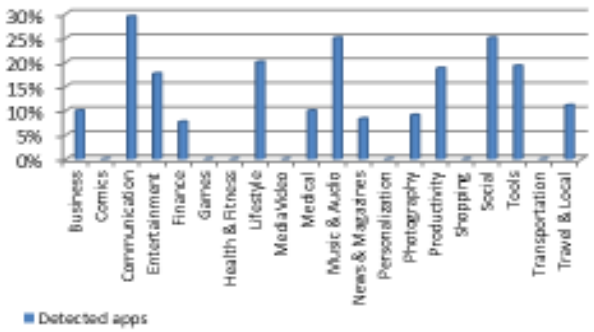


Figure 4. Detected applications according to the different categories of the Google play.

The obtained results along with the statistical analysis done by Sarma *et al*. [20] on the permissions of each category in the Google play will be used in future work in order to define more fine grained rules that conforms with the functionality of the applications in each category. This can have a significant impact on reducing the false positive rate in detecting over-privileged applications.

5. Conclusions

This paper addressed the colluding attack in Android by checking the applications which are potentially able to conduct malicious activities. These applications are able to collude in order to acquire extra privileges through permission sharing. The considerable number of detected applications that were able to share their permissions which was 12.90% of all detected applications showed the impact of employing the proposed mechanism in detecting over-privileged applications. Moreover, by employing the new defined rules, the proposed mechanism could address new kinds of attacks, as the result, a number of applications with the capability of conducting these kinds of attacks were detected. Finally, analyzing the results showed that defining more fine grained rules by taking the functionality of each application into consideration, can result in reducing the false positive rate in detecting over-privileged applications which will be the next step for improving this mechanism in the future work.

References

[1] Bartel A., Klein J., Le Y., and Monperrus M., "Automatically Securing Permission-Based Software by Reducing the Attack Surface: An Application to Android," *in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, USA, pp. 274-277, 2012.

[2] Bradley T., "DroidDream Becomes Android Market Nightmare," available at: http://www.pcworld.com/businesscenter/article/221247/droiddream_2015becomes_android_market_nightmare.html, last visited 2015.

[3] Bugiel S., Davi L., Dmitrienko A., Fischer T., and Sadeghi A., "Xmandroid: A New Android Evolution to Mitigate Privilege Escalation Attacks," *Technical Report*, Technische Universität Darmstadt, Germany, 2011

[4] Chan P., Hui L., and Yiu S., "A Privilege Escalation Vulnerability Checking System for Android Applications," *in Proceedings of the 13th International Conference on Communication Technology*, Jinan, pp. 681-686, 2011.

[5] Davi L., Dmitrienko A., Sadeghi A., and Winandy M., "Privilege Escalation Attacks on Android," *in Proceedings of the 13th International Conference on Information Security*, Boca Raton, USA, pp. 346-360, 2010.

[6] Egele M., Kruegel C., Kirda E., and Vigna G., "PiOS: Detecting Privacy Leaks in iOS Applications," *in Proceedings of the 18th Annual Network and Distributed System Security Symposium*, San Diego, USA, pp. 1-15, 2011.

[7] Enck W., Gilbert P., Chun B., Cox L., Jung J., McDaniel P., and Sheth A., "TaintDroid: An Information-Flow Tracking System for Real-time Privacy Monitoring on Smartphones," *in Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, Vancouver*, Canada, pp. 99-106, 2010.

[8] Enck W., Ongtang M., and McDaniel P., "On Lightweight Mobile Phone Application," *in Proceedings of the 16th ACM Conference on Computer and Communications Security*, Chicago, USA, pp. 235-245, 2009.

[9] Fang Z., Han W., and Li Y., "Permission Based Android Security: Issues and Countermeasures," *Computer and Security*, vol. 43, pp. 205-218, 2014.

[10] Faruki P., Bharmal A., Laxmi V., Ganmoor V., Gaur M., Conti M., and Rajarajan M., "Android Security: A Survey of Issues, Malware Penetration and Defenses," *Communications Surveys and Tutorials*, vol. 17, no. 2, pp. 998-1022, 2014.

[11] Fragkaki E., Bauer L., Jia L., and Swasey D., "Modeling and Enhancing Android's Permission System," *in Proceedings of the 17th European Symposium on Research in Computer Security*, pp. 1-18, 2012.

[12] Google Inc., "Android Security Overview, Security and Permissions," available at: http://source.android.com/tech/security/#android-application-security, last visited 2015.

[13] Hsiao Sh-W., Hung S-H, Chien R., and Yeh C-W., "PasDroid: Real-time Security Enhancement for Android," *in Proceedings of the 8th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, Birmingham, England, pp. 229-235, 2014.

[14] Kashefi I. and Salleh M., "A Survey on Mitigating Attacks Related to Shortcomings of Andoid Permission Framework," *Journal of Theoretical and Applied Information Technology*, vol. 55, no. 2, pp. 1-9, 2013.

[15] Mahaffey K. and Hering J, "App Attack: Surviving the Explosive Growth of Mobile Apps," pp. 1-93, 2010.

[16] Mittal P., Dhruv B., Kumar P., and Rawat S., "Analysis of Security Trends and Control Methods in Android Platform," *in Proceedings of International Conference on Innovative Applications of Computational Intelligence on Power, Energy and Controls with their Impact on Humanity*, Ghaziabad, pp. 75-79, 2014.

[17] Nauman M. and Khan S., "Design and Implementation of a Fine-grained Resource Usage Model for the Android Platform," *the International Arab Journal of Information Technology*, vol. 8, no. 4, pp. 440-448, 2011.

[18] Nauman M., Khan S., and Zhang X., "Apex: Extending Android Permission Model and Enforcement with User-Defined Runtime Constraints," *in Proceedings of the 5th ACM Symposium on Information*, New York, USA, pp. 328-332, 2010.

[19] Pettey C., "Gartner Says 428 Million Mobile Communication Devices Sold Worldwide in First Quarter 2011, a 19 Percent Increase Year-on-Year," available at: http://www.gartner.com/it/page.jsp?id=1689814, last visited 2011.

[20] Sarma B., Li N., Gates C., Potharaju R., Nita-Rotaru C., and Molloy I, "Android Permissions: A Perspective Combining Risks and Benefits," *in Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, New York, USA, pp. 13-22, 2012.

[21] Shabtai A., Fledel Y., Kanonov U., Elovici Y., and Dolev S., "Google Android: A state-of-the-Art Review of Security Mechanisms," available at: http://arxiv.org/abs/0912.5101, last visited 2009.

[22] Shabtai A., Fledel Y., Kanonov U., Elovici Y., Dolev S., and Glezer C, "Google Android: A Comprehensive Security Assessment," *IEEE Security and Privacy*, vol. 8, no. 2, pp. 35-44, 2010.

[23] Thurm S. and Kane Y., "Your Apps Are Watching You," *The Wall Street Journal*, available at: http://online.wsj.com/article/SB10001424052748704694004576020083703574602.html, last visited 2015.

[24] Zhou X., Lee Y., Zhang N., Naveed M., and Wang X., "The Peril of Fragmentation: Security Hazards in Android Device Driver Customizations," *in Proceedings of IEEE Symposium on Security and Privacy*, Washington, USA, pp. 409-423, 2014.

[25] Zhou Y., Zhang X., Jiang X., and Freeh V., "Taming Information-Stealing Smartphone Applications (on Android)," *in Proceedings of the 4th International Conference on Trust and Trustworthy Computing*, Pittsburgh, PA, USA, pp. 93-107, 2011.

Iman Kashefi received his MS degree in computer science at Universiti Teknologi Malaysia (UTM) in the field of information security in connection with the years of related work experience in IT Development Center of Iran. The Best Student award of the UTM was granted to him and he was honored to receive the Pro-Chancellor award among the eight best PhD and MS graduates of the UTM. He received his Bachelor's degree in the field of computer engineering from Islamic Azad University of Tehran and has published Journal papers in the field of network security and Android security. Along with conducting research on smartphones security, currently he works as Solution Manager in Mobile Communication Company of Iran (MCCI).

Maryam Kassiri is a lecturerat Islamic Azad University (IAU), lecturing under the Department of Computer and Information Technology. She received her MS degree in Management of Information Technology from Payam-e-Noor University, and her BS degree in Information Technology Engineering from Islamic Azad University. She also serves as an IT expert in the eLearning sector of IT Development Center of Iran, affiliated to Industrial Development and Renovation Organization of Iran. She has published some Journal and Conference papers related to her research works including eLearning and Network Security.

Mazleena Salleh is an associate professor at Universiti Teknologi Malaysia (UTM), lecturing under the Department of Computer Science, Faculty of Computing. She has taught several courses in the area of computer hardware system, cryptography and computer security. She received her PhD in Computer Science at UTM in the field of computer networking while her Master's degree from Virginia Polytechnic State University in the field of electrical engineering. She has published several journal and conference papers related to her research works that include watermarking, steganography, chaos image encryption, network analysis, e-learning and knowledge management. Her current research is on computer security related issues namely data survivability and availability in cloud, elliptic curve cryptography, body sensor network and detection of misuse in computer forensic.