Journal of Computer Science

Original Research Paper

# An Empirical Analysis of Android Permission System Based on User Activities

**Ankur Rameshbhai Khunt and P. Prabu**

*Christ University, Hosur Road, Bangalore-560029, Karnataka, India*

*Article history*
Received: 09-12-2017
Revised: 13-01-2018
Accepted: 6-02-2018

Corresponding Author:
Ankur Rameshbhai Khunt
Christ University, Hosur Road, Bangalore-560029, Karnataka, India
Email: ankur.khunt@mca.christuniversity.in

Co-Author:
Dr. Prabu. P.
Christ University, Hosur Road, Bangalore-560029, Karnataka, India
prabu.p@christuniversity.in

**Abstract:** In today's world there has been an exponential growth among smart-phone users which has led to the unbridled growth of smart-phone apps available in Google play store, app store etc., In case of android application, there are many free applications for which the user need not shell out a penny to use the services. Here the magic word is "free" which entices millions of pliant people into installing those apps and giving unnecessary access to their data and device control. Current studies have shown that over 70% of the apps in market, request to gather data digressive to the most functions of apps that might cause seeping of personal data or inefficient use of mobile resources. Of late, couple of malignant applications gather unobtrusive information of the user through third-party applications by increasing their permissions to high-level on the Android Operating System. Android permission system provides, the user access to the third party apps and in return based on the permissions granted by the user, an app can access the related resource from the user's mobile. A user is bound to grant or deny permits during the installation of the application. For the most part, users don't focus on the asked permissions, or sometimes users do not understand the meaning of the permission and install the app on their device. They allow a way for attackers to perform the malicious task by demanding for more than expected set of permissions. These extra permissions permit the attacker to exploit the device and also retrieve sensitive information from it. In this research paper we describe how permission system security can create an awareness among the users that would assist them in deciding on permission grants. This improved and responsible user activities in Android OS can help the users in utilizing their device securely.

**Keywords:** Android Security, Permissions, Malicious Application, Data Leakage

## Introduction

Android is simply an operating system that facilitates a user to interact and manage mobile devices through a Graphical User Interface (GUI). Some of the features of an android driven smart-phone are GPS capability, camera functionality, internet accessibility, touch screen interface, provision for application installation which is the main differentiating and important feature in comparison with generic mobile phones. To run these applications, Android devices support Operating System (OS) in the similar way as computer supports the operating system. Some most popular OS are Windows, Android, Linux, iOS, etc.

Android is the widely used open source operating system. This in-turn makes it very difficult in managing as any developer can make application in their own way and user isn't aware of pitfalls in the application, about its background services and activities and the related security threats.

Android operating system is based on Linux kernel. There are four layers in Android architecture. Each layer has different tasks. The base layer is Linux kernel which maintains Android operating system security and other components. This layer contains all device drivers, USB drivers, Bluetooth drivers, Wi-Fi drivers, display drivers and it is also helpful in maintaining power management of Android system. Rooting of the device creates security

vulnerability in the Android operating system. After rooting the device hacker or attacker can have direct access to Linux kernel. Once the device is rooted, then there is no permission required for accessing the device.

Native libraries are the upper layer of Linux kernel which mainly consist of all kind of default libraries, for example, SQLite is for all database related operation, Webkit is used for inbuilt web browser, OpenGL is utilized for 2D and 3D graphics in Android and SSL is giving network access related authentications. The developer can use all libraries in their application, but many times attackers may put some extra permission in their apps and might misuse those libraries.

Android runtime also provides the core libraries and most importantly Dalvik Virtual Machine (DVM) facilities which help to run the application on the device. DVM as compared to JVM optimizes the Android device providing fast performance while consuming less memory.

Android Framework provides us with the lot of APIs like package manager, View Manager, content provider, Activity Manager, Resource Manager and also provides lots of classes and interfaces for Android application development.

The existing android system is a permission based system. For most applications, there are a set of permissions that need to be accepted for successful launching of the application. Most of these permissions concern with user sensitive data that are not needed for that application, for example a gaming app asking for permission to access contacts. So, the proposed system lets the user know how many other users have either accepted or rejected these permissions.

### *Permission System*

In Android, each application has one manifest .xml file and in the absence of this file, task of running the application is nearly impossible. It contains the entire list of activities which are used in the apps and additionally it also includes all the permissions which the application needs.

Android forces apps to declare the permissions during the installation. The app user has to decide to grant or revoke the permission of any android applications before or after the installation. Malicious apps cannot be a treat to device until user allow access to demanded permissions. Most of the time user allow application to access android sub-services unknowingly, which causes improper working of device. To create basic awareness, the user could decide whether to allow to access certain permission or not. By providing information at bottom of the permission box, about how many people liked or disliked the set of permissions asked by application at its first time of use. Based on number of likes it helps the user to make certain decisions. Providing this reference, it helps to reduce android threats, crime and also protect the sensitive data. Given below are some sorts of permissions mostly used in the android application:

- Android.permission.READ_CONTAC
- Android.permission.WRITE_CONTACT
- Android.permission.READ_STORAGE
- Android.permission.WRITE_STORAGE
- Android.permission.RECEIVE_SMS
- Android.permission.WRITE_SMS
- Android.permission.SEND_SMS
- Android.permission.READ_SMS
- Android.permission.INTERNET
- Android.permission.READ_PHONE_STATE

Through the above given set of permissions, the applications can have access over all the resources. This manifest file is written within the XML which contains some kind of tags. Through those tags one can define the usage of application and structure of permission.

In Fig. 1 we can see the architecture of the Access Permission in Android. This is a basic model and it's same for all android devices. There are two applications, Application 1 and Application 2 where application 2 provides access to local data like contacts, sms, etc. and device component control like camera, mic, etc. In Application 1, there is one module “A” and in Application 2 there are modules “B” and “C”. Module A can access the module “B” and “C” if they are assigned permission labels of Application 1.

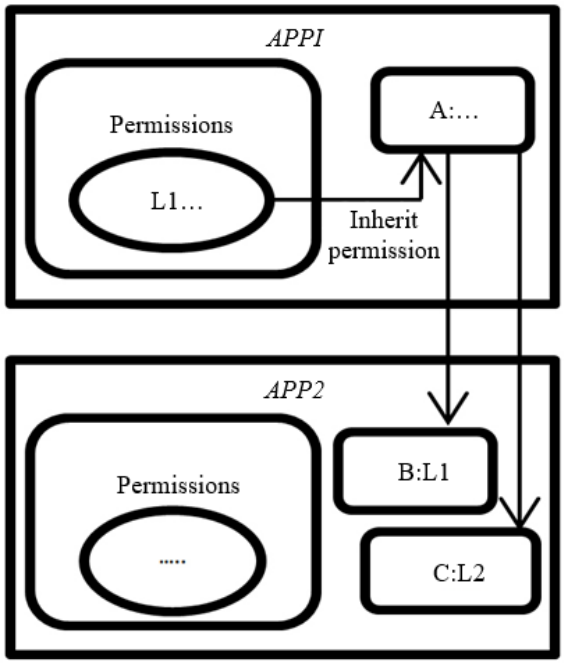


**Fig. 1:** Access permission architecture (Enck *et al*., 2009)

## Literature Review

All applications once installed resides in the sandbox, a secluded zone of the framework that does not have access to device control or device data, unless permissions are explicitly allowed by the user when the apps is being installed. In most cases, permission is requested only when the application is run on the device by the user and not when installed. In either case, permission based mechanism is broadly criticized for its coarse-grained control of application permissions and the inefficient permission management by developers, marketers and end-users. For instance, users can either acknowledge all permission demands from an application to install it or not install the application (Rashidi and Fung, 2009).

Luyi *et al*. (2014) proclaimed that they had developed a new service for automatically identifying the Pileup risks on a mobile OS. They also claim that their approach can continuously scan new emerging Android versions to find out new vulnerabilities and can also conveniently detect related malicious apps on Android devices without undermining its utility.

To be on the safer side always make sure that only apps published in trusted and well-known market place like Google play are download and installed. Disabling application download from poorly trusted sharing channels can protect from accidental installation of poorly trusted or malicious packages (Vecchiato *et al*., 2016).

With all said and done android security depends on equipment and programming along with the portrayal and integration of user interface with the device. An individual smart-phone uses various technologies i.e. music player, telephone, advanced camera and much more, which may contain directions for serious roles to abuse different attack routes (Vecchiato *et al*., 2016).

The developers gain consent permissions through the Manifest file inside an application. Developers characterize the application's security framework. By duping the users into giving consent to the permission requests, the developers gets undetectable access to the device's secured resources and data (Faruki *et al*., 2015).

When approaching users for permission in an Android app, the OS may face proper restrictions which enforced upon by the server. To avoid the forced restriction by the server, the app must be stated as medicinal application while installing. This ensures that the user application is properly interfaced with the trusted server of association (Andow and Wang, 2015).

For introducing an application, the user needs to grant all the asked permission requests for that particular application or else that application won't be installed on the user's device. Accepting the installation implies that user is giving the permission to the application for getting access to all the resources asked for by the application from the device. Along these lines, the user must be vigilant in setting the choice since attacker and malware developer exploit user's lack of regard and develop the malicious application that requests unnecessary permissions (Jain and Prachi, 2016).

In the early years of android, according to William Enck and their team android straightforwardly made use of permission mark task show to limit access to resources and applications. Later on developers brought about several changes in the way android applications were built by which explicit permission setting wasn't needed to access the resources but it can be done just by installing the applications. It can be done by setting the exported attribute to false or by letting the android decide at its discretion based on a distinct attribute in its manifest to create a private component. Security control can be trimmed using such components. By making a component private, the developers do not have to worry about permissions label that has to be assigned to it or about the likelihood that application might get them another label (Enck *et al*., 2009).

A survey was done based on a research paper on the topic of security issues on Android Smartphone's. This research paper was named as Timing Information Stealing Smartphone Application (TISSA). According to this research, TISSA is a structure which is convenient for serving security to the call logs, contacts and so on. The user without being worried can safeguard all of its contacts and call logs by stuffing all of the permissions, by employing TISSA. The users without any doubt can give the unrestricted access to its data in exceptional protection mode and this all happens in the groove of submitting all of the permissions. The private data of the user which is spilled under some circumstances influence a whole lot of android applications which is examined by TISSA. TISSA uses three principle components such as dexterous CPU, memory and vitality which give safety and reliability to the call logs, contacts and other data. These principal components are the privacy setting content provider that is deployed to provide current privacy setting to the already installed apps. Privacy setting manager is very beneficial in enhancing the privacy setting for the installed apps or any execution. Privacy aware components are extended to access the entrance into the user's data which further integrates contacts, call logs and other data. When the user dispatches a request through an installed application to the current provider that is when TISSA start to work.

Third party applications collect user's routine usage, ask for high priority permissions and demand to be a part of Android OS. Chances of getting attacked by the malicious and unsafe third party applications cause risk to Android OS as it changes the various root-level permission and can separate the entire framework security. That will cause malfunctioning of the system (Android device) and android OS will not perform as per

the expectation. Transitive rights utilization is not managed by the android OS which allows applications to bypass the denied access services. This is done by forcing a user to allow access in term of permissions asked during installation of a particular application. Allowing for various permissions is a loop hole in Android OS causing possibilities of various attacks that have been reported (Lee *et al*., 2013).

During installation, any application asks for permissions which are granted by the user, but there are some applications which disturb the performance of android OS and also affect the working of other applications, without asking for permissions. Permission security model is a part of Android which allows the user to decide whether to trust on any application or not. Any applications cannot collect information like ISP unless and until user allows or permit application. Allowing applications by granting permissions could lead to attack scenario which cause malfunctioning of the device as well as Android OS. So during installation process any application, the user must focus on what kind of permission is asked and be careful while granting permissions requested by the particular application (Mohini *et al*., 2013).

Concluding all, the secured and safe behavior of Android device is completely user depended because at the end users decision matters a lot. During installation, the display shows the list of permissions demanded by the application. It could be dismissed or approved at the will of the user (Enck *et al*., 2012).

Android gives authentication to component security prerequisites by using permissions. Android OS has a capability of permitting centrally so that the system software and other application work properly. A similar mechanism is used by the third-party application to set and define new permissions that are brought in work by taking OS permission. Finally, permissions are accorded to apps during installation and stored in Package Manager Service (PMS) (Heuser *et al*., 2014).

Most users do not bother to read or have a comprehensive understanding of the way that the install time permission works in android application. Considering that the users are exploited by malware which uses Android Security Model (ASM) applications benefits (Heuser *et al*., 2014).

The different security mechanism like permission systems, robust detachment between apps, etc., are being executed in android and other portable operating systems. But it was found that this mechanism was insufficient (Fragkaki *et al*., 2012).

## Proposed Work

In our research work, we want to implement small extra module in Android permission system. Occasionally many users might not know or might not understand that how many permissions should be allowed or denied, but after applying this module users will frequently be able to decide whether the permission need to be allowed or denied.

In the module, we are putting one label with permission which is recommended to the user to allow or deny the permissions. This special label shows the how many users had allowed or denied that special permission.

Based on working of application, its requirements must be fulfilled. If user does not grant permit to any one of the permissions, those service will not be accessed by application.. So blindly never grant access to any application to use our device services.

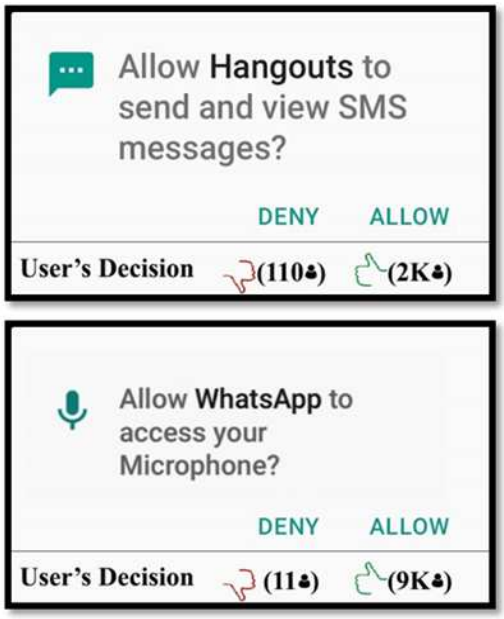


**Fig. 2:** Demo view of permission module in Android

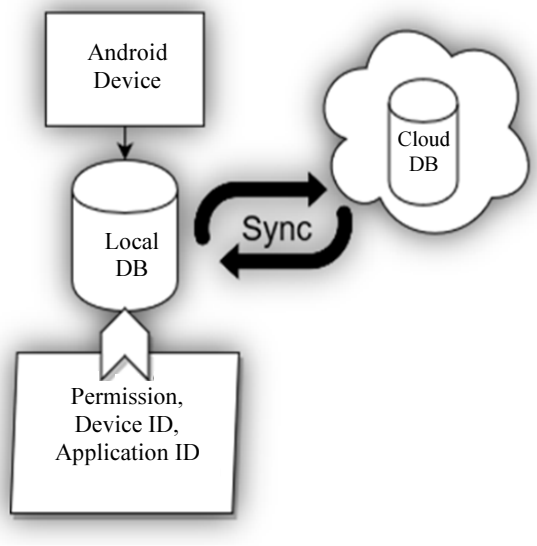


**Fig. 3:** Implementation of module

Due to advancement in android security, user can grant or deny permissions demanded. Further Fig. 2 shows the exact scenario with examples which is outcome of the proposed work. Figure 3 is the overall architecture of proposed work. It would lead to increases the awareness of user and help the user to make the decision whether to allow or deny the permission before or after installation of the applications. Providing a bottom line showing likes and dislike help user as well as developer.

Hangouts application is asking permission for "send and view SMS messages?" which was permitted by 2K users and denied by 110 users. In the same way WhatsApp application is asking permission for "access your Microphone?" that was permitted by 9K users and denied by 11 users. Now the new user installing the app can check these permissions statistics to decide whether or not to allow that permission request.

The statistical information will come from the Internet when user's internet is ON or OFF and it will display the label. Through this tag, the user can quickly decide the permission should be allowed or denied. Based on this simple tag we can improve the Android permission related security than the existing.

### *Implementation*

When the user gets his device connected to the internet the Android local database syncs with server database given the application is installed on the device. All devices are recognized through a unique device id similar to the MAC address of PC and other computing devices. Through the given device id, we can get the information about installed application like the application ID. After getting application id, one can easily view all the permissions utilized by particular application and can also insert entries in a database.

In the database, four initial entries do appear Device id, Application id, Permission Label and State. State demonstrates that permission accepted or denied by the user. The value of the state is 0 or 1. Zero means Denied and One means Allowed appropriate permission.

Figure 4 shows the working of getting the permission data whether the user has denied or allowed the permission request of the application. When any application is installed by the user in their device, the application might ask for the set of permissions to access the device components like camera, microphone, storage or user's data like location, contacts, SMS etc. The applications have different types of permissions with different functionalities. In this algorithm I'm trying to get the data from the user's device to the server which is the choice of the user about to allow or deny the request of the permission.

In the server database the information which is to be stored is DeviceID, AppID, PermissionLabel, A = allow and D = deny. The local database information will be stored likewise. Depending on the device's internet connectivity if the device is connected to the internet it will store the data in both the databases at the same instance, otherwise it will store the data in the local database and will update the server when the internet will be connected.

When the user allows the permission, the allow variable (A) will be incremented by 1 and the whole record with full information like AppID, DeviceID, PermissionLabel, A = 1 and D = 0. When the user allows the permission, the deny variable (D) will be 0. Same will be vice versa when the user denies the permission (A = 0, D = 1). Once the choices are made by the user, it will check for the internet connectivity and if connected, the data will also be stored in the server database, otherwise only in the local database and will update in the server on establishing the connection.

### *Algorithms and Results of the Proposed System*

There are two different algorithms that have been used in the proposed system. These algorithms are used to fetch the data from user's device and displays it. The algorithms and the results are as follows:

### *Algorithm 1: Get Data from user's Device*

Now, the application permission data from the user device has been fetched using the above algorithm. For displaying how many users have allowed or how many has denied, the data will be displayed from the server with the different algorithm.

So, here we can see an example of the database and how data is stored. The database comprises of five columns. First is device id, which will store the all users device id. Second is App id, which will store the app id of install applications by the users and third is Permission, It stores permission asked by the installed app. The 3rd and 4th column is Allow and Deny. It will store the operation performed by the user on particular app permissions.

There a two ways or scenarios in which the database in the server gets updated.

During first time installation and running of the software, based on the customer preferences on the permission request, the device ID, App ID and different permissions and its status would be added as a new line item in the database.

During second time or repeated installation and running of the software, only the status of each permission request would be updated as per the user choice. A new line entry for the same device ID won't be created to avoid duplicate entries that can impact the total count statistics adversely.

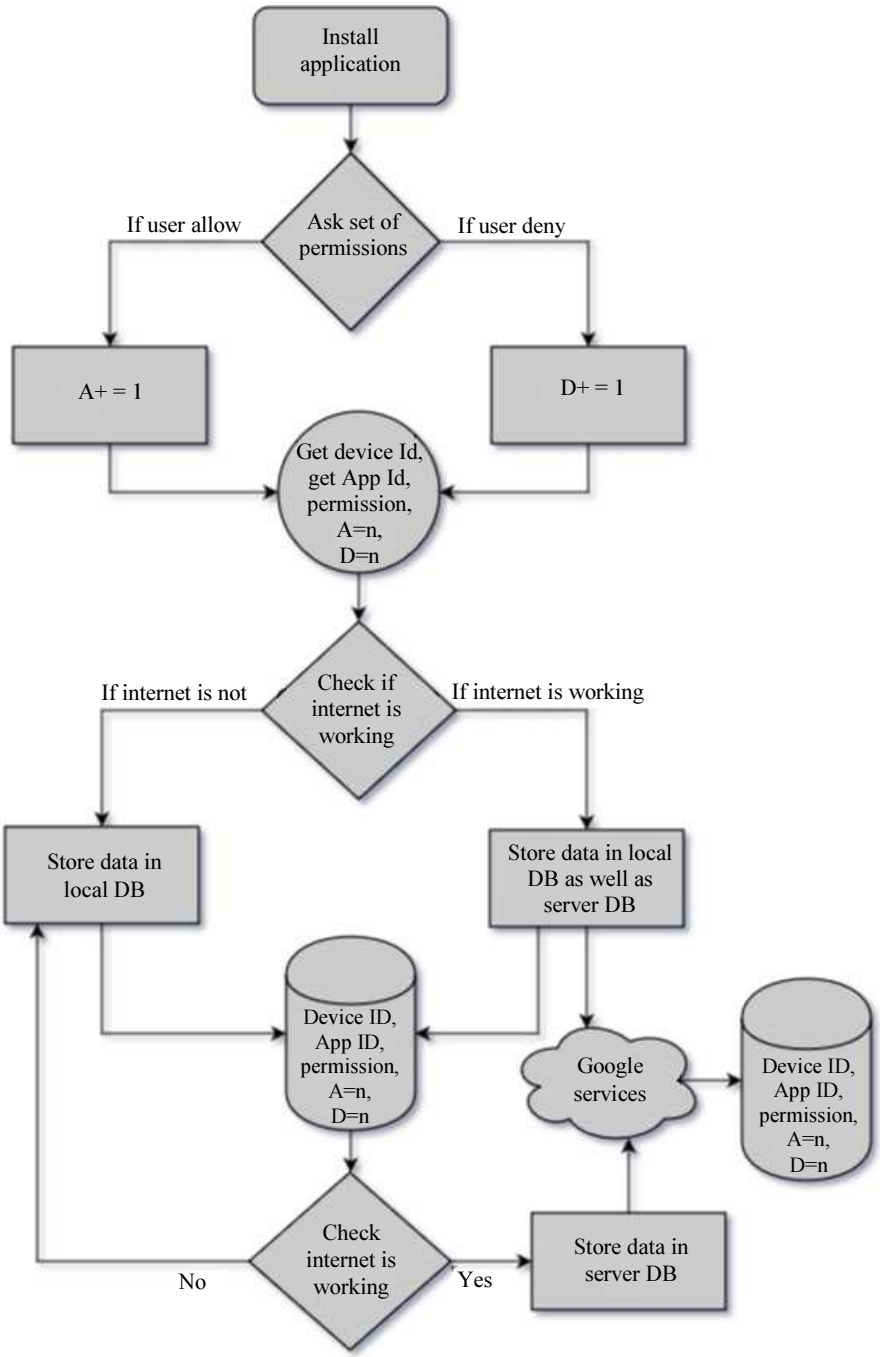


**Fig. 4:** How to get data from user's device

In this example the chatApp has been installed on six devices. The chatApp has many permissions but here we mention only two namely "READ_CONTACT" and "BLUETOOTH" permissions. 1st, 2nd and 4th users allow "READ_CONTACTS" permission and 3rd user denies that permission. 5th user allows "BLUETOOTH" permission and 6th user denies that permission. This example shows how server database stores all data.

In Fig. 6, the data to be provided to the user from the server is been described. In this algorithm, globally allowed and denied permissions statistics for the application is provided to the user at the time of application download. The user can leverage on this information to make the right decision in addressing each permission request. Now let us understand the working of the algorithm.

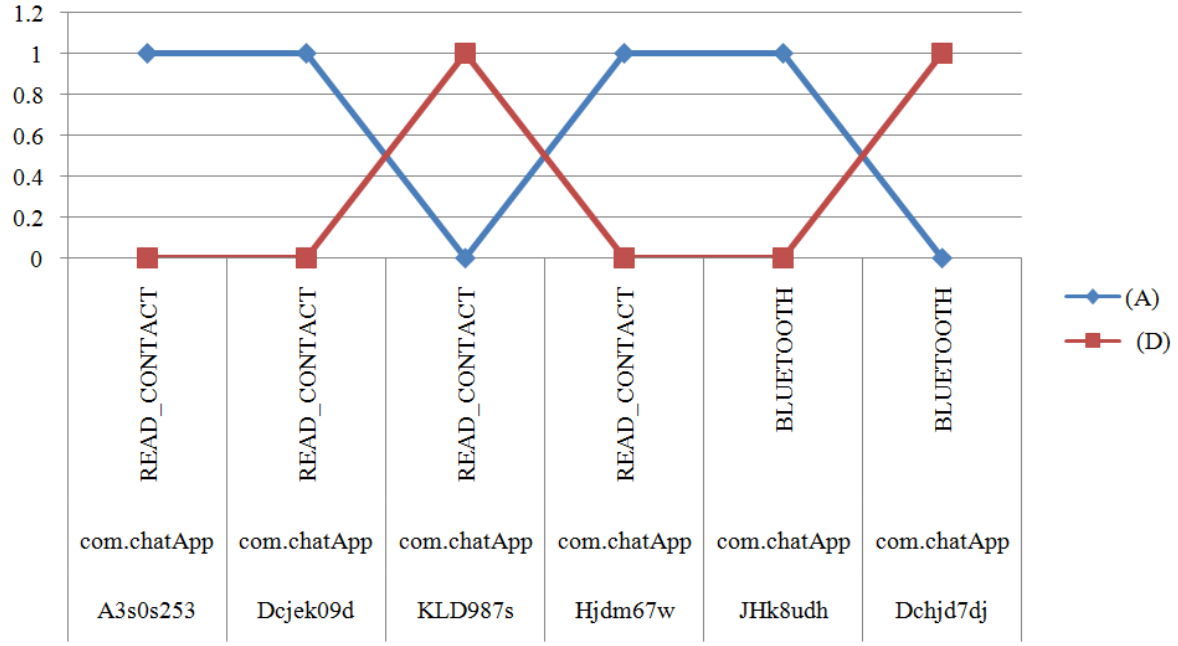


**Fig. 5:** Graph of Table 1

When a user initiates an application installation process from any App store, the OS initially checks whether the app is already installed in the device or not. If not, then it will immediately stop the process. Otherwise, during the installation process it will get the App ID and Device ID for that particular device and store it in the local database. After that it will fetch the count of Allowed (A) and Denied (D) based on the App ID and Device ID for the particular permission from the server to the device. This permission statistics data from the server helps the user in deciding which permission request he should honor or not as per the public opinion. Based on the choices made for each permission request, a line record would be created for each permission request with its status in the local device.

*Algorithm 2: Fetch Data from Server to User's Device*

Now after fetching the data from the server, it will check whether the App ID is there in the local database or not. If not, then data will be fetched from the server with information like Device ID, App ID, Permission, A = n, D = n (n = number of counts). But if found, then it will compare the permission and its global status count. If there is no difference then, it will stop as it contains all the data updated as in the server. In case, there are any changes in the two databases, it will update the local database with specific records with the data fetched from the server and will store the final content of data in the local database.

**Table 1:** Example of how data store in server

| Device ID | App ID | Permission | (A) | (D) |
|---|---|---|---|---|
| A3s0s253 | com.chatApp | READ_CONTACT | 1 | 0 |
| Dcjek09d | com.chatApp | READ_CONTACT | 1 | 0 |
| KLD987s | com.chatApp | READ_CONTACT | 0 | 1 |
| Hjdm67w | com.chatApp | READ_CONTACT | 1 | 0 |
| JHk8udh | com.chatApp | BLUETOOTH | 1 | 0 |
| Dchjd7dj | com.chatApp | BLUETOOTH | 0 | 1 |

**Table 2:** Example of how data store in local database

| Device ID | App ID | Permission | (A) | (D) |
|---|---|---|---|---|
| Abcde1 | com.chatApp | READ_CONTACT | 3 | 1 |
| Abcde1 | com.chatApp | BLUETOOTH | 1 | 1 |

The following database table shows the information transferred from the server.

Now, when the user opens the application after installation, it will ask for certain permissions to the user. As shown in Fig. 2 it will show a dialog box with two options namely Allow and Deny for any particular permission like using Camera, accessing Contacts etc. It will additionally show the count of how many people allowed the permission for that particular application and how many denied for the same.

In Fig. 7 example server data is shown in Table 1, In that table total three users allowed "READ_CONTACT" permission and one user denied. For "BLUETOOTH" permission, one user allowed and 1 user denied that permission. Same count summary of appropriate permissions is shown in Table 2.

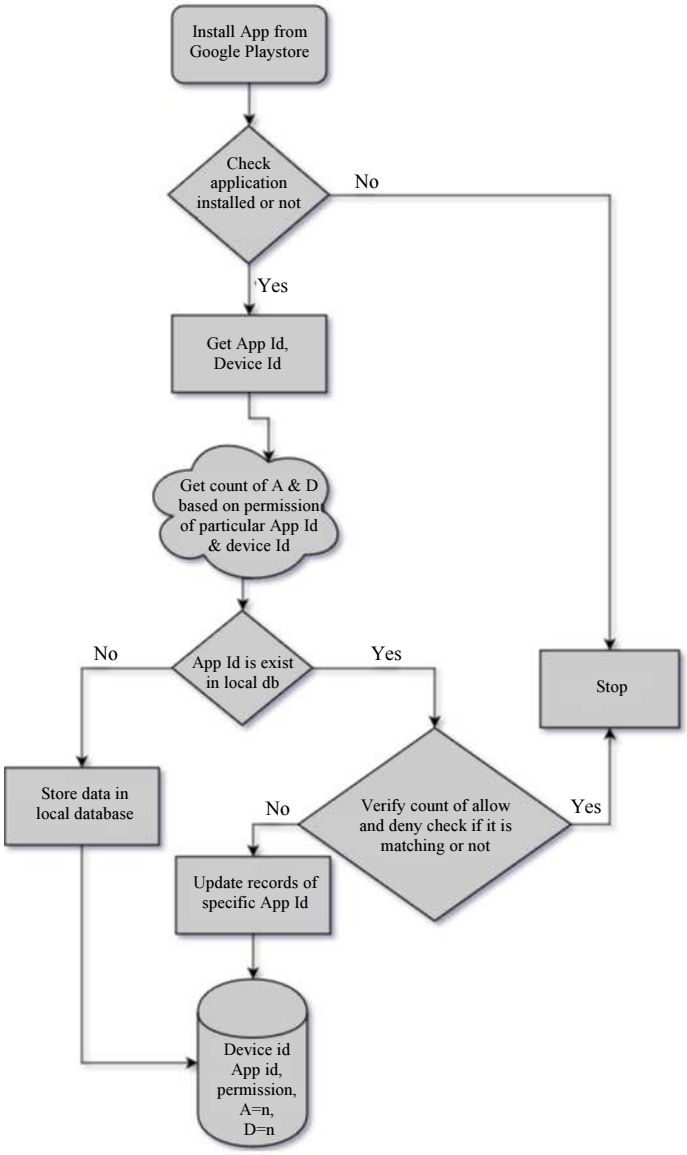


**Fig. 6:** How to fetch data from server to user's device

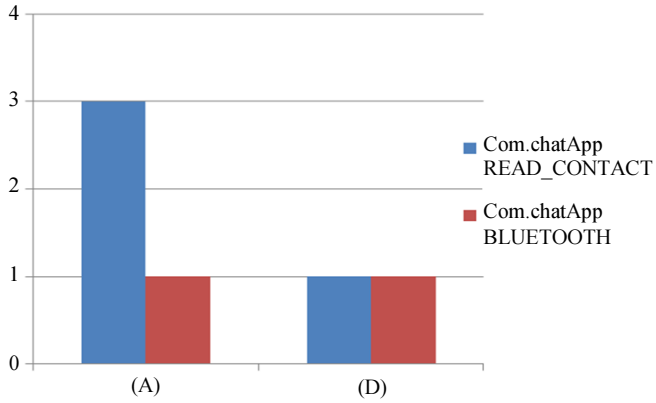


**Fig. 7:** Graph of Table 2

## Results and Discussion

As we see in algorithm 1 the individual data of each user is fetched. The permission name along with its status are extracted for each user. This data has been processed by algorithm 2 to generate the consolidated report. This report consists of each permission name and the number of users who have either accepted or denied the demand permission (Fig. 2).

This number of acceptances and rejections are very useful for a any new user to decide whether to allow or deny the permission for the application. If more number of users have denied the permission, then the new user will know that it may not be safe to allow the permission.

## Conclusion

A wide range of applications incorporate permission excursive to the application's utility. These permissions allow access to assets which are delicate in nature. This may result in the spillover of user data or utilized by the third party identified by the application. The user is unaware of this and agrees to the permissions because the user is not warned about these threats in any possible ways. So the proposed system permits the user to see that in reality how many people have genuinely agreed to this permission and how many have not. This increases the security of information and permits the user to choose which information he/she needs to share. In future, I expect up-gradation of security by evacuating those permissions which are dismissed by the greatest number of users. The application will make a request to the developer to avoid those permissions which most users have rejected in order to continue in the application market. This initiative would guarantee general safety and security of user information and counteractive action of third party applications utilizing private information.

## Acknowledgment

We thank our colleagues from Christ University, Bangalore who provided insight and expertise that greatly assisted the research.

We would also like to show our gratitude to Joy Paulose, H.O.D of Department of Computer Science, Christ University, for sharing their pearls of wisdom with us during this research and we thank 3 "anonymous" reviewers for their so-called insights.

## Authors Contributions

**Ankur Rameshbhai Khunt:** Android extra module for permission system.

**P. Prabu:** Allow and deny services.

## Ethics

All information provided in this paper is confidential and unique. This paper has neither been published nor is under review elsewhere. In this article, we propose a new method and algorithm for the Android security. Any implementation or adaptation of this idea is subjected to the user's own result and the idea and result in this paper no way guarantees safety, security or some pre defied result.

## References

Andow, B. and H. Wang, 2015. A distributed android security framework. Proceedings of the IEEE International Conference on Smart City, Dec. 19-21, IEEE Xplore Press, pp: 1045-52. DOI: 10.1109/SmartCity.2015.207

Enck, W., D. Octeau, P. McDaniel and S. Chaudhuri, 2012. A study of android application security. Systems and Internet Infrastructure Security Laboratory.

Enck, W., M. Ongtang and P. McDaniel, 2009. Understanding android security. IEEE Security Privacy Magazine, 7: 50-57. DOI: 10.1109/MSP.2009.26

Faruki, P., A. Bharmal, V. Laxmi, V. Ganmoor and M. Gaur *et al*., 2015. Android security: A survey of issues, malware penetration and defenses. IEEE Commun. Surveys Tutorials, 17: 998-1022. DOI: 10.1109/COMST.2014.2386139

Fragkaki, E., L. Bauer, L. Jia and D. Swasey, 2012. Modeling and enhancing android's permission system. CyLab at Carnegie Mellon University.

Heuser, S., A. Nadkarni, W. Enck and Ahmad-Reza Sadeghi, 2014. ASM: A programmable interface for extending android security. Proceedings of the 23rd USENIX Conference on Security Symposium, Aug. 20-22, USENIX Association, San Diego, CA, pp: 1005-19.

Jain, A. and Prachi, 2016. Android security: Permission based attacks. Proceedings of the 3rd International Conference on Computing for Sustainable Global Development, Mar. 16-18, IEEE Xplore Press, New Delhi, India, pp: 2754-59.

Lee, C., J. Kim, S. Cho, J. Choi and Y. Park, 2013. Unified security enhancement framework for the Android operating system. J. Supercomput., 67: 738-756. DOI: 10.1007/s11227-013-0991-y

Luyi, X., P. Xiaorui, W. Rui, K. Yuan and W. XiaoFeng, 2014. Upgrading your android, elevating my malware: Privilege escalation through mobile OS updating. Proceedings of the IEEE Symposium on Security and Privacy, May 18-21, IEEE Xplore Press, San Jose, CA, USA, pp: 393-408. DOI: 10.1109/SP.2014.32

Mohini, T., S. Ashish Kumar and G. Nitesh, 2013. Review on android and Smartphone security. J. Comput. Inform. Technol. Sci., 1: 12-19.

Rashidi, B. and C. Fung, 2009. A survey of android security threats and defenses. J. Wireless Mobile Netw. Ubiquitous Comput. Dependable Applic., 6: 3-35.

Vecchiato, D., M. Vieira and E. Martins, 2016. The perils of android security configuration. Computer, 49: 15-21. DOI: 10.1109/MC.2016.184