ISSN: 2278 – 7798
International Journal of Science, Engineering and Technology Research (IJSETR)
Volume 2, Issue 7, July 2013

Security Threat Avoidance through Selective Permissions in Android

Swapnil Powar

***Abstract*— The current Android application framework works on an "all or none" permission policy that means application will be installed only if all required permission are granted. and we cannot deny any permission after installation if we do it will lead to miss use of granted permission. There are some mechanisms addressing this issue for Example CyanogenMod which denies unwanted permission of installed app but it will leads to application crash. Others are APEX and WhisperCore trying to address this issue but this models are not publically available. In this paper we trying to address problem of misuse of permission and proposed novel idea of "Shadow Manifest". This shadow manifest system contains following components SelPerProvider to host user permission. In general whenever there is resource request system manifest is checked and resources are allocated in our solution after system manifest check control flows through Shadow Manifest. Query that is given to shadow manifest when returns TRUE then resources are allocated to application. Here we create shadow manifest database and update it according to misused permissions and avoid permission misuse by granting dummy resource for unwanted permission. User can decide unwanted permission of app by using log analysis app. Given app has been implemented and tested by Contact Retrieve.**

***Index Terms*— Android, Selective Permissions, Content Providers, Application Framework, Logcat**

I. INTRODUCTION

Android OS assign separate unique userid to each android application and each application runs as process on its own Dalvik Virtual Machine instance. Application cannot communicate each other directly and they have limited access to operating system. Application A cannot access Contact info of Application B until A has that permission. Sandboxing mechanism based on unix separate the processes and file permissions.

1.1 Application Elements:-

Following are the element of any android application AndroidManifest.xml: It contains definition for each system level operation including all important components like activities, services, broadcast receivers, and InterProcess Communication in an application. It also specifies which permission the app requires.

- **Activities:** It is the entry point of application usually includes displaying UI to the user.
- **Services:** It works in background. It runs in own process or in context of other application. Other components bind with services and access methods using RPC.
- **Broadcast Receiver:** Object of broadcast receiver created when Inter Process Communication initiated by OS by calling Intent.
- **InterProcess Communication:**

Binder: Lightweight capability based RPC performing in process and cross process call

Intent: Simple message object explain intention of do something.

Content Provider: It is a data store house that gives access to data stored on device.

1.2 Permissions:

When we install application form android market that application need permission to install on devices for example Read_Contact, Write_Contact, Read_Calender, Write_Calender, Read_SMS, Access_Coarse_Location, Internet, Send_SMS, etc. Here to install app we have to assign all required permission to app otherwise if we deny application will not be installed because there is no selective permission mechanism.

II. RELATED WORK

Selective permission can be implemented by intercepting function that actual perform system level permission check where we can disable assigned permission (CyanogenMod) but it may lead to application crash. WhisperCore is a private resource allocation mechanism publicly not available. Application permission extension framework (APEX) intercepts call to permission check function and redirect flow to custom defined function which perform user policy checking.

III. MOTIVATIONS

Today Smartphone's are coming with android operating system. Android operating system allows third party application to be installed on devices. It leads rapid growth in android market so there is possibility of security threats. Misuse of grated permission is increasing. The existing mechanism are not publically available and are not efficient to avoid selective permission problem so here we introducing "shadow manifest" mechanism that grant selective permission try to avoid security breach.

Manuscript received June , 2013.
***Swapnil Powar**, M.tech-NIMS Dept. of Comp. Engg. VJTI Mumbai*

IV. PROPOSED SOLUTION

A. Introduction:

1. Existing Permission Checking Android Framework:

Fig1. shows how the control flows in android OS when application components access the resources. First the control flows to contextimpl.java in Android.app class having functions like checkCallingPermissionOrSelfPermission(), and checkPermission().

The ApplicationContext uses IActivityManager interface that works on concept of Binder and Percels for IPC.The ApplicationContext uses parcel to decide whether the calling application has a specific permission.

The ActivityManagerNative class accept this parcel and extracts the Process ID (PID), User ID (UID) and the permission related with the call and sends these extracted data as argument to the checkPermission() method of the ActivityManagerService class. These arguments are then passed to the checkComponentPermission(), which performs multiple checks; for system or root UID permission is always granted and for all other it invokes other UIDs, it invokes the PackageManagerService, which extracts the package names for the calling UID and validates the received permissions against the grantedPermission hashset of the application.

If the received permission does not match any of these contained in the hashset, the security exception will be thrown by Android framework. The android manifest file for each application is written in XML.

2. Content Providers resource accessing existing framework:

Content provider provides centralized access to data. Generally commonly accessed puts in content provider. Following content providers present in android CalendarProvider, ContactsContractProvider, UserDictionaryProvider, MediaProvider, etc.

Fig. 2 shows flow of control when application accesses the resources. The application instantiates an object of the ContentResolver class to insert, delete or query a particular database. It provides the Uniform Resource Identifier (URI) used to uniquely identify each resource. Unique URI is associated with the databases and files in the content provider ContentResolver object map this URI to correct Content Provider. Abstract ContentProvider class overridden by the appropriate Content Provider and it is present in com.android.provider package.

The resource class is defined in the android.provider package which interacts with the appropriate provider through an Application Program Interface. Insert, Delete, and Query method are overridden by each provider in com.android.provider package. To insert and delete application require write permission and to query application require read permission on database in ContentProvider.

B. Proposed Shadow Manifest Solution:

Fig. 3 shows shadow manifest solution for granting selective permission to application. The requested resource should be given to application only if that application should have permission in system manifest to grant that resource. If permission is absent then security exception is thrown and application terminates abnormally. Shadow Manifest positioned after the system manifest check. If the related permission entry for the particular resource that was requested is present in the shadow manifest, the resource is granted, else a null value is returned to the application, so application believe that the user has not recorded any data.

C. Algorithm

Input: System manifest file and Shadow Manifest file.

Output: Deny application permissions as per user requirement.

Process:

1) If system perm(X) and if shadow perm(X) then grant real resource to application X
2) If system perm(X) and if !shadow perm(X) then grant empty resource to application X
3) If !system perm(X) then application X terminate abnormally.

here system perm(X) means granting resource looking at android manifest and shadow perm(X) means granting permission after shadow returns TRUE.

D. Implementation

Android source code need to be installed on Linux machine and to run application we should have emulator.

Installing app in android OS step:-

1) Copy application folder to app package/folder
2) Create Application.mk file in application folder
3) Add name of application folder in core.mk file in build\target\product
4) In cmd type source build
5) Set path using setpaths command
6) Go to application folder and put mm to sync application
7) Execute lunch full-eng command to initiate the emulator, make –j4 to build and run the emulator.

1. Log Creation:

Creation of Log is shown in Figure 4. The checking of permissions during the resource request happens in the PackageManagerService The permissions recently used by applications installed in the phone are logged from this file to the system log buffer called logcat. The user can view the log using the Log_read application to check for malicious applications. Log_Read retrieves the log entry from the system logcat file to get the desired output.

2. Implementation of shadow manifest:

Sel_Perm application developed by us shows list of installed application along with their granted permissions. If user wanted to revoke any permission then particular permission is to be marked as uncheck and changes made through this GUI are stored to database referred as shadow manifest.

Database created in application cannot be referred by Android framework and database created in android OS cannot be accessed by application so database is embedded in Content Provider to give access to many applications.

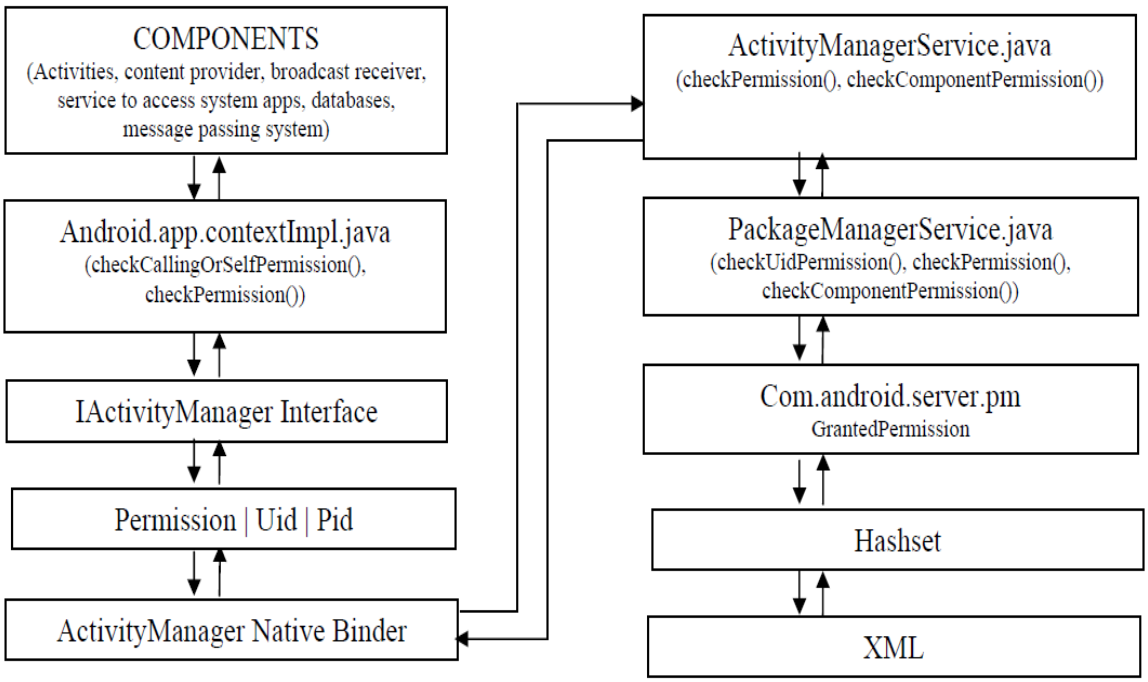

Figure1. Existing Android application framework for checking permissions

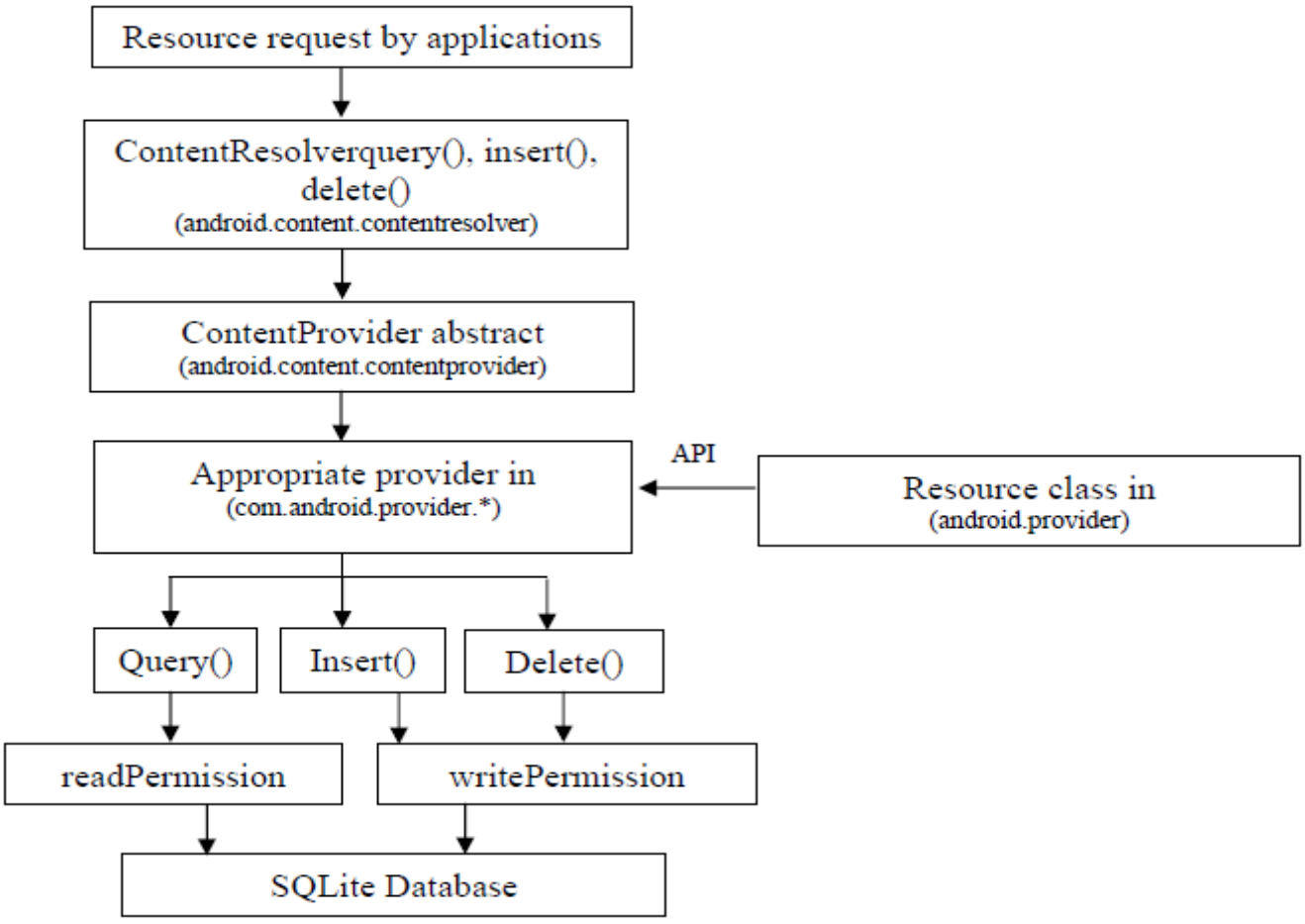

Figure2. Existing framework for accessing resources of Content Providers

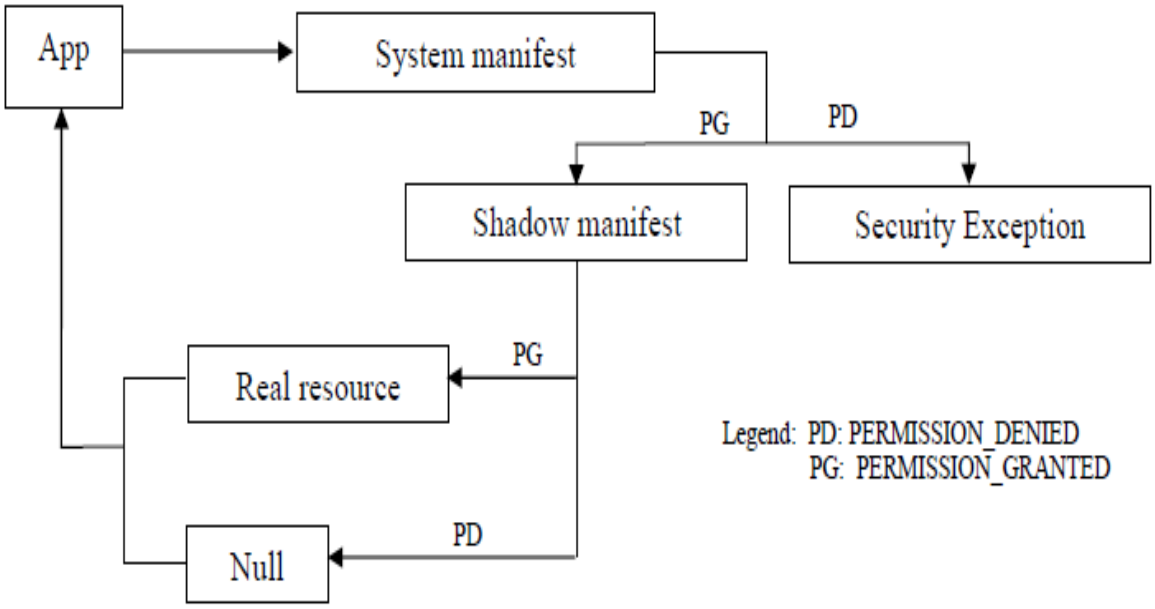

Figure3. Control flow through the proposed Shadow Manifest

Content provider can be created by application but it cannot be accessed by Android OS. It should be created in android OS.

SelPermProvider, created for this purpose, it has database to stores the shadow manifest file. Sel_Perm application is given access to this ContentProvider. Whenever the user unchecks a particular permission, Sel_Perm accesses that database and inserts the permission with state zero in the database. The structure of the database is as follows: {Permisssion_id(PRIMARY KEY), Appname(UID), Permission name(Perm) and State(0 or 1)}.

Figure 5 shows the realization of shadow manifest file. To realize this permission check within the SelPermProvider, the query function of each Content Provider needs to be modified so that it queries the permissions database in the SelPermProvider.

SQLite database is supported by the Android. This is used to write queries to query the database to retrieve the required data. Since Content Providers can be accessed by any application, applications other than Sel_Perm can also write to the permissions database. To avoid this issue, while inserting the entries into the permissions database, it is ensured that the application which is doing the insertion is only Sel_Perm and not any other application by checking the package name of the calling application with the package name of Sel_Perm

E. Experiments and Results

The Log_Read, Sel_Perm applications installed in the Androidemulator and the ASelPermProvider was introduced in android system image used by the emulator. Figure 6 shows the screen shot of the Log_read application. The log shows the date, time, application name, application uid, and the permission name which was logged in the logcat system file from the PackageManagerService when an application useda permission. Figures 7 and 8 are the screen shots of the Sel_Perm application which display the list of applications and the permissions associated with each application. Figure 7 shows the list of installed applications such as Camera, Calculator, Sel_Perm, Broswer, etc. Figure 8 shows the list of permissions associated with the Browser application.

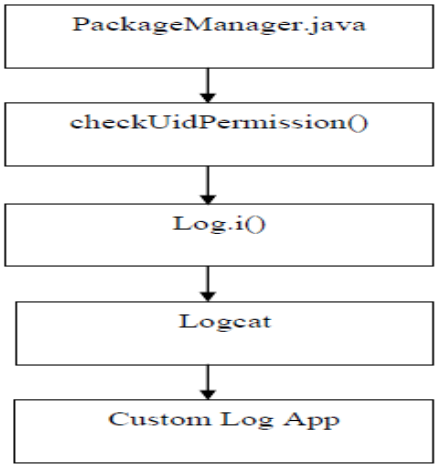

Figure4. Creation of log

V. Conclusion

Android Application framework has problem of detecting malicious applications due to “all or none ” permission policy. The selective permission policy mechanism which is proposed and implemented in this paper helps to detect malicious applications using the permissions unaware of the user and restricts to only a few permissions with the knowledge of the user. This solution has been implemented with minimum changes to the existing Android application framework, leaving the core components the same.
This patch up can be applied to the current Android version-Ice Cream Sandwich or be released in the upcoming version of Android.

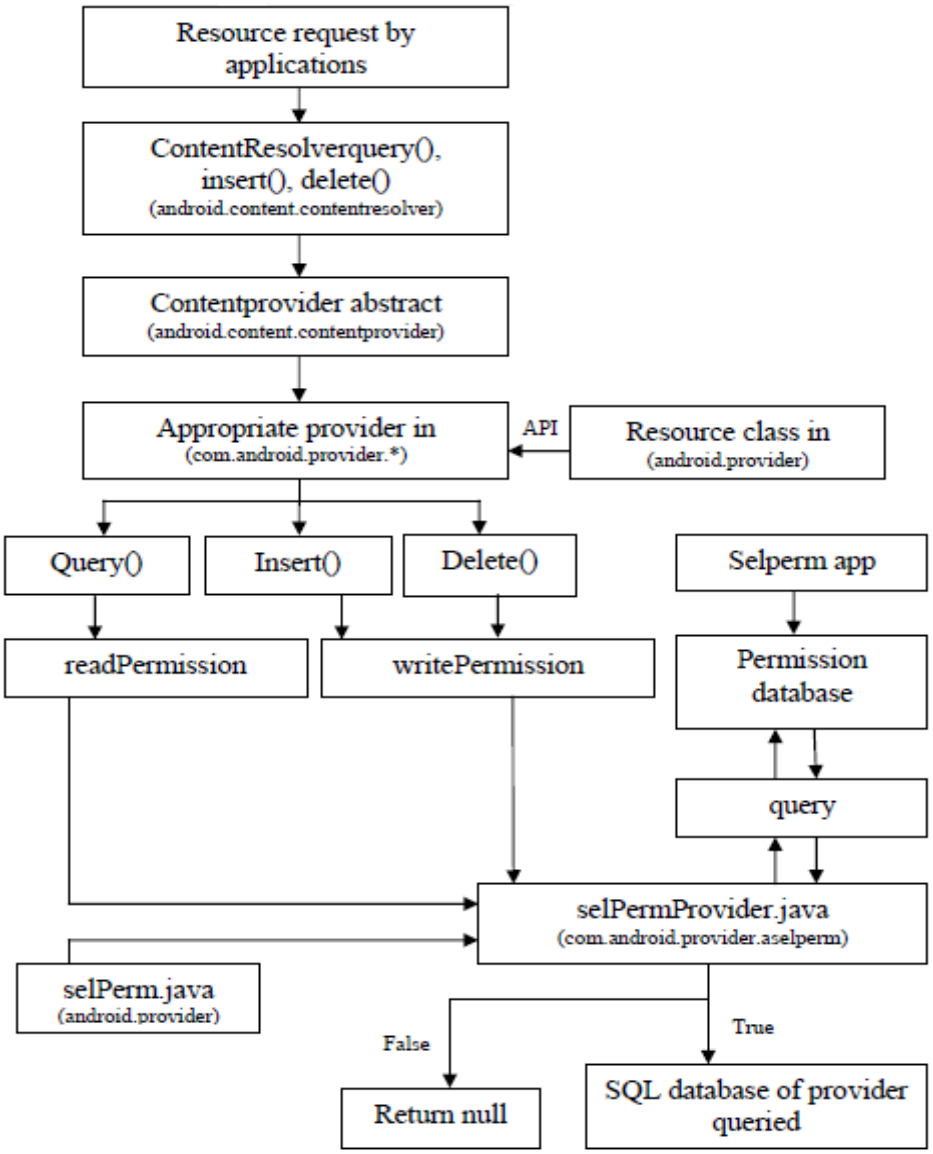

Figure.5 Realization of shadow manifest file

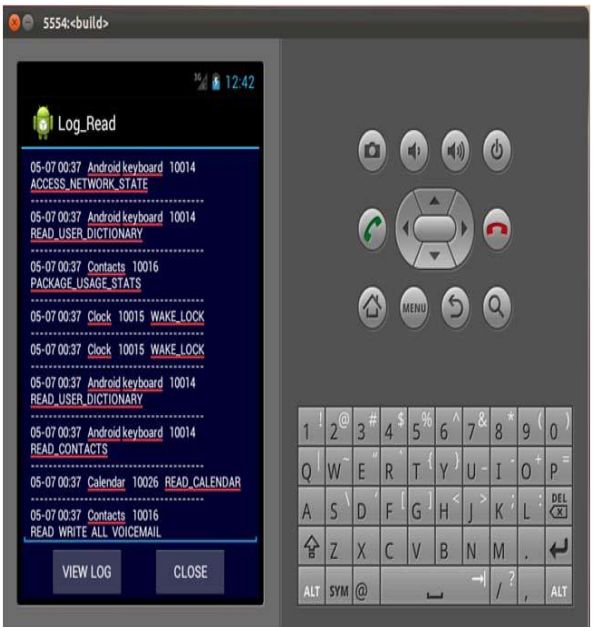

Figure6. **Screenshot of Log_Read application**

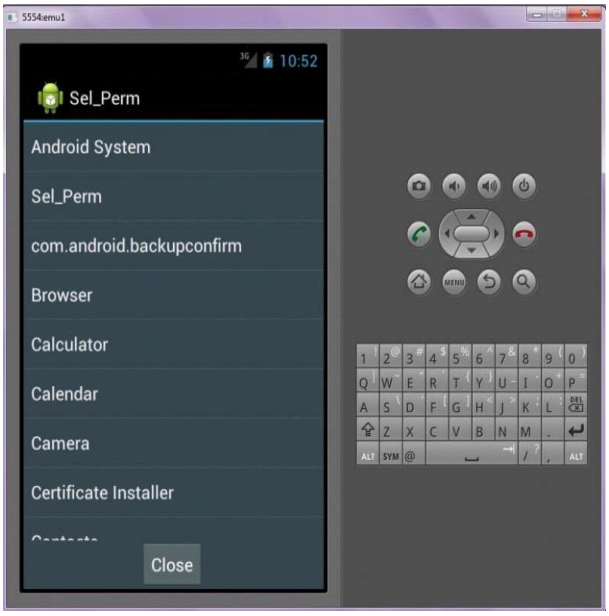

Figure7. **Screenshot of Sel_Perm application-1**

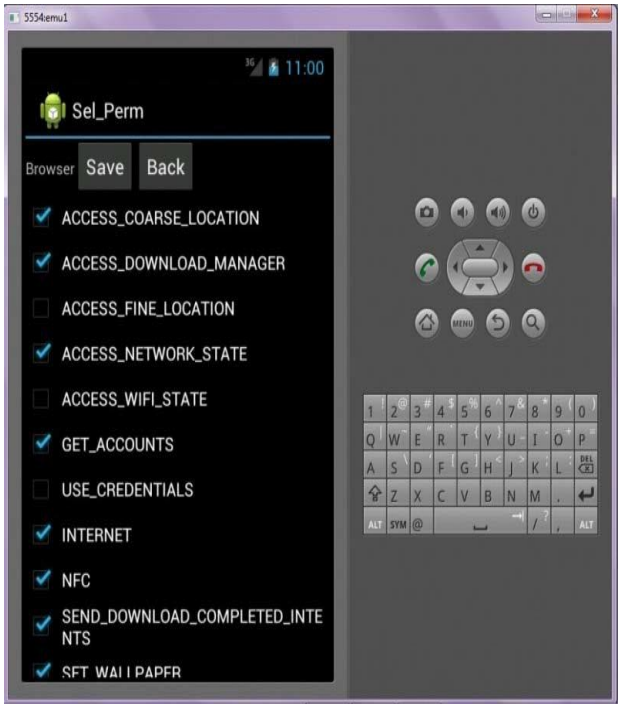

Figure8. **Screenshot of Sel_Perm application-2**

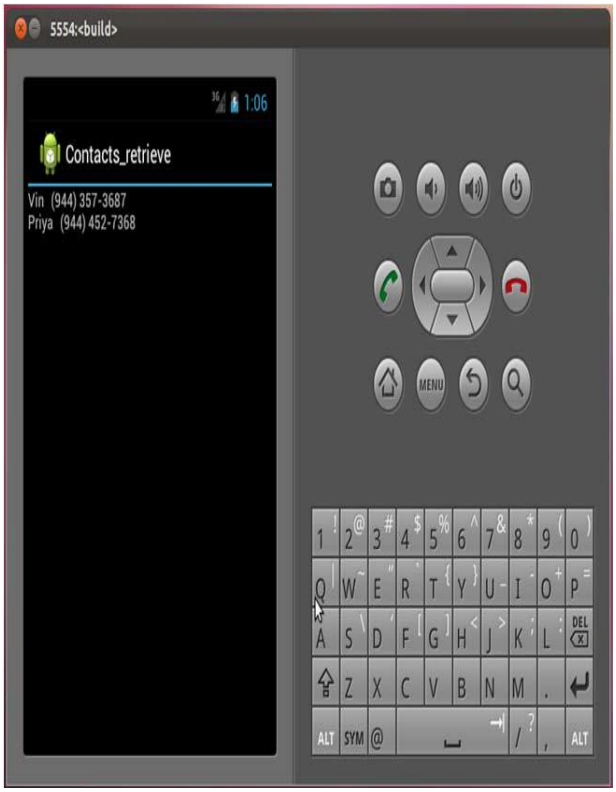

Figure9. **Screenshot of Contacts_retrieve application-1**

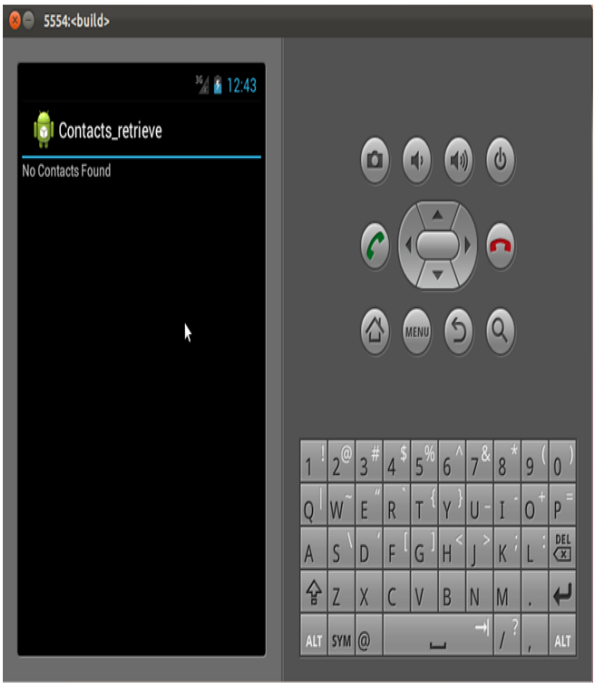

Figure10. Screenshot of Contacts_retrieve application-2

REFERENCES

[1] Android open source project, Application Sandbox, http://source.android.com/tech/security/index.html.
[2] Android open source project, Elements of Applications, InterprocessCommunication, http://source.android.com/tech/security/index.html.
[3] CyanogenMod, Permission Management- CyanogenMod Settings, http://www.cyanogenmod.com/about.
[4] WhisperSystems, Selective permissions for Android, http://www.whispersys.com/permissions.html.
[5] Mohammad Nauman and Sohail Khan. 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, Oct 2011
[6] Stackoverflow, Android "Application Framework" Docs/Tuts, http://stackoverflow.com/questions/8992677/android-applicationframework-docs-tuts.
[7] GrepCode, http://grepcode.com/snapshot/repository.grepcode.com/java/ext/com.google.android/android/4.0.3_r1/
[8]