Avoidance of Security Breach through Selective Permissions in Android Operating System

Lakshmi Priya Sekar
National Institute of Technology, Tiruchirappalli,India
Department of Computer Science and Engineering
lpriya.ms@gmail.com

Vinitha Reddy Gankidi
National Institute of Technology, Tiruchirappalli,India
Department of Computer Science and Engineering
vinitha210@gmail.com

Selvakumar Subramanian
National Institute of Technology, Tiruchirappalli,India
Department of Computer Science and Engineering
ssk@nitt.edu

ABSTRACT

The current Android application framework has an “all or none” permission policy, viz., an application can be installed if and only if all the permissions are granted. Also, no provision exists to deny granted permissions after installation. Therefore, any application can misuse the granted permission. CyanogenMod addresses this issue by denying the unwanted permissions which might cause the application to crash. WhisperCore, has worked on this problem but the working model is unavailable at the moment. APEX, also is currently doing some research on this issue but there is no publicly available document.

In this paper, the problem of misusing the granted permission n in Android is addressed and a novel idea of ‘shadow manifest’ is proposed. The proposed shadow manifest is implemented by creating a novel Content Provider, viz., SelPermProvider, which hosts the user permissions. In general, during the resource request phase, the system manifest is checked and the resources are allocated. But, in our implementation, the control is altered to flow through the shadow manifest after the system manifest is checked. If the query to shadow manifest is TRUE, then the resource is granted else a dummy or null value is returned. This facilitates the user to identify the malware and block the malware from achieving the intended task. Thus, the application is unaware of the indirect permission denial and continues to run normally. The user can decide which permission to restrict by checking a log of all recent permission requests, a facility provided in our app.

The proposed shadow manifest has been implemented and tested using an application called ‘Contacts_Retrieve’. It was found to successfully complete the application if the shadow manifest returned TRUE, and unsuccessfully complete otherwise.

Categories and Subject Descriptors

C.5.3 [**Iphone**]: Android open source – *permissions, application development.*

General Terms

Algorithms, Design, Security

Keywords

Android, Selective Permissions, Content Providers, Application Framework, Logcat

1. INTRODUCTION

The Android system assigns a unique user ID (UID) to each Android application and runs it as that user in a separate process in its own instance of a Dalvik Virtual Machine [1]. This approach is different from other operating systems (including the traditional Linux configuration), where multiple applications run with the same user permissions.

By default, applications cannot interact with each other and applications have limited access to the operating system. If application A tries to do something malicious, viz., read application B's data or dial the phone without permission (which is a separate application), then the operating system protects against this malicious intention because application A does not have the appropriate user privileges. The sandbox is simple, auditable, and based on decades-old UNIX-style user separation of processes and file permissions.

1.1 Elements of Applications

The following are the main building blocks of Android application [2]:

- **AndroidManifest.xml**: The AndroidManifest.xml file is the control file that contains definition for system operation with all the top-level components such as activities, services, broadcast receivers, and InterProcess Communication in an application. This also specifies which permissions are required.
- **Activities**: An Activity is, generally, the code for a single, user-focused task. It usually includes displaying a User Interface (UI) to the user. Typically, one of the application's Activities is the entry point to an application.
- **Services**: A Service is a body of code that runs in the background. It can run in its own process, or in the context of another application's process. Other components "bind" to a Service and invoke methods on it via remote procedure calls.
- **Broadcast Receiver**: A Broadcast Receiver is an object that is instantiated when an IPC mechanism known as an Intent is issued by the operating system or another application.
- **InterProcess Communication**
  Processes can communicate using any of the traditional UNIX-type mechanisms. Android also provides the following IPC mechanisms:
  - **Binder**: A lightweight capability-based remote procedure call mechanism designed for high performance when performing in-process and cross-process calls. Binder is implemented using a custom Linux driver.
  - **Intents**: An Intent is a simple message object that represents an "intention" to do something.
  - **ContentProviders**: A ContentProvider is a data storehouse that provides access to data on the device; the classic example is the ContactsProvider that is used to access the user's list of contacts. An application can access data that other applications have exposed via a ContentProvider, and an application can also define its own ContentProviders to expose data of its own.

1.2 Permissions

When the user installs an application on an Android phone from the Android Market, a list of permissions that the app needs in order to function appears. The list of permissions in Android are: READ_CONTACTS, WRITE_CONTACTS, READ_CALENDAR, WRITE_CALENDAR, ACCESS_COARSE_LOCATION, READ_SMS, ADD_VOICEMAIL, INTERNET, SEND_SMS, SET_WALLPAPER, and so on. If an application wants to read contacts, then it has to include READ_CONTACTS in its manifest file.

Android apps are not generally implemented with the expectation that they could be denied access to a resource that has been requested. One can only grant or deny all permissions that the application is requesting. There is no scope for selective permission.

The unavailability of selective permissions leads to availability of extra permissions, which could be later used to exploit the Android OS. For example, a weather app in an Android phone gives the weather conditions at a particular place. It might need GPS and internet connection but it would not need access to the messaging service. Suppose this weather application, which prompts for messaging service, is available in the market and the user installs it, then the application can send messages to random people, costing the user as well as draining the resources. This is a very dangerous flaw that needs to be addressed. An attempt has been made in this paper to solve this issue by creating a new application which grants selective permissions by maintaining a shadow manifest and altering the flow through this manifest. The permission entries to be created in the shadow manifest could be obtained with the prior knowledge of executing the application and investigating the logs. To revoke the unnecessary permissions, such permissions are maintained in the shadow manifest with a mask. Henceforth, the application would get an empty resource instead of the real resource for the revoked permissions so that it would not crash due to lack of the resource and the security of the user is also guaranteed.

The rest of the paper is organized as follows: In Section 2 Related Works are discussed. In Section 3 Motivation is given. In Section 4 the proposed shadow manifest, an illustration of permission checking, the implementation details, and the results of execution are discussed in detail. Finally Section 5 concludes the paper.

2. RELATED WORK

Selective permissions feature can be implemented by intercepting the flow to the function that performs the actual permission check, or by disabling the permission in the manifest file of the application. CyanogenMod [3], an open source Android project has addressed this issue by denying the permission, which may cause the application to crash. WhisperCore [4], a software from WhisperSystems, attempts to solve the problem by granting a “private resource” for each permission revoked by the user for an application. A working model of the WhisperCore is currently not available. Application permission extension framework (APEX) [5] intercepts the call to the permission check function and redirects the flow to a custom defined function which performs user policy checking.

3. MOTIVATION

Current smartphones come with Android Operating System. Android allows third party developers to create applications and put them on the Android market. While this gives access to a wide variety of applications tailored to the requirements of the user, the all or nothing policy of the application permission poses serious security issues. Naturally, the user has to grant all permissions in order to be able to install the application. There is no way of restricting the extent to which an application may use the granted permissions. The only way of revoking permissions once they are granted to an application is to uninstall the application.

There is no practical solution to this problem till date. This motivated us to propose the novel idea of ‘shadow manifest’ in this paper. The solution proposed in this paper implements selective permissions by making changes to the Android Application framework at the higher level so that it can be incorporated in the existing source code directly. The advantage of our proposal is that the number of checks involved in granting selective permissions to the application is minimal.

4. PROPOSED SOLUTION

4.1 Introduction

In Android [6], applications are composed of components which can be instantiated and executed on their own. There is no main() function as such. The instantiation of these components is handled by different methods of the ApplicationContext class in the Android application framework layer.

4.1.1Existing Android application framework for checking permissions

Figure 1 shows the flow of control in the Android OS when the components of applications access a resource. The OS checks if the application has the permission to do so. First the control flows to contextimpl.java in Android.app class which has functions like checkCallingPermissionOrSelfPermission(), and checkPermission().The ApplicationContext implements the IActivityManager interface that uses the concept of Binders and Parcels, the specialized Inter Process Communciation mechanism for Android.

The ApplicationContext creates a parcel aimed at deciding whether the calling application has a specific permission. The ActivityManagerNative class receives this parcel and extracts the PID, UID, and the permission associated with the call and sends these arguments to the checkPermission() method of the ActivityManagerService class. These arguments are passed to the checkComponentPermission(),which performs multiple checks; if the UID is a system or root UID, it always grants the permission. For all 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 Android framework throws a security exception signifying a denial of the permission. The android manifest file which is present for each application is written in XML.

4.1.2 Existing framework for accessing resources of Content Providers

Content provider manages access to central repository of data. If some data has to be available to all applications, then the best way is to put it in a content provider. The content providers present in the Android source code are CalendarProvider, ContactsContractProvider, UserDictionaryProvider, MediaProvider, etc. Content Providers can also be created from an application, but such a ContentProvider will be present in the work space of the application.

Figure 2 shows the flow of control when an application wants to access a resource [7]. The application instantiates an object of the Content Resolver class, through which it can insert, delete or query a particular database and provide the Uniform Resource Identifier (URI) which is used to uniquely identify each resource. The databases and files in the content providers are associated with a unique URI. ContentResolver object uses this URI to map to the correct Content Provider. This call traces back to the Content Provider abstract class, the functions of which are overridden by the appropriate Content Provider in the 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. Each provider in the com.android.provider package overrides the insert, delete, and query methods. The permissions required for each of these operations are as follows: to insert and delete, the application should have a writePermission on the databases in the ContentProvider, to query the application should have a readPermission on the databases in the Content Provider.

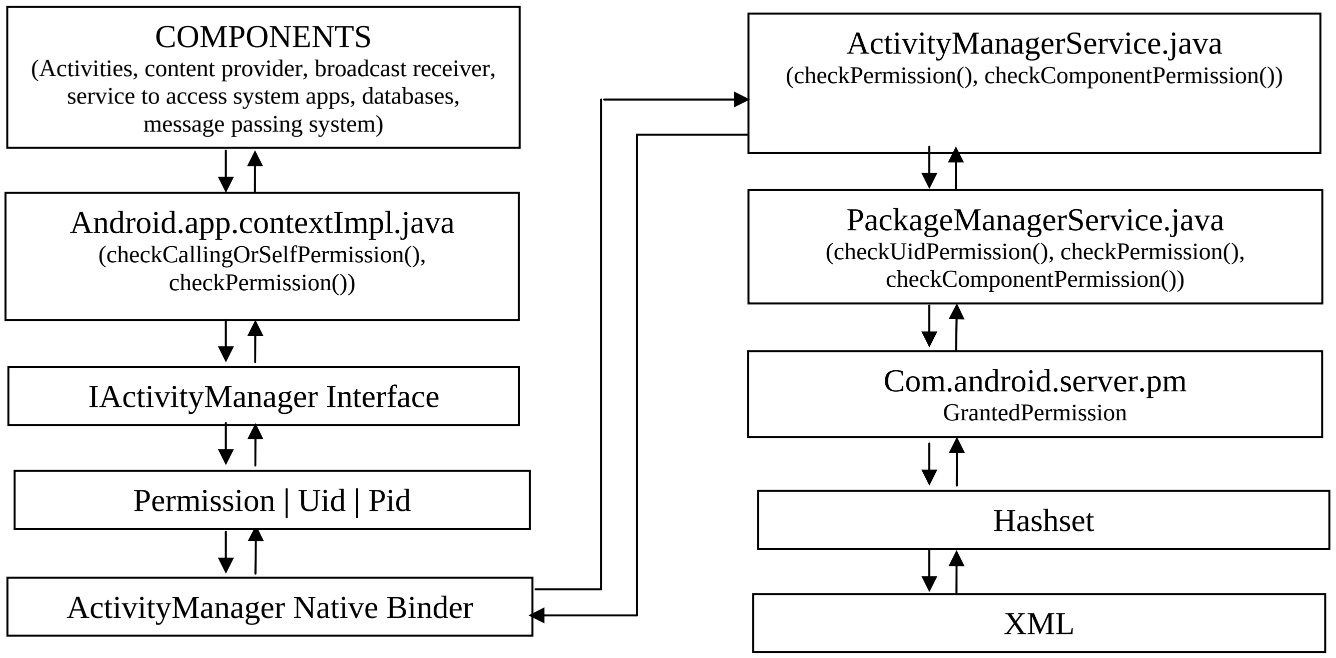

COMPONENTS
(Activities, content provider, broadcast receiver, service to access system apps, databases, message passing system)
Android.app.contextImpl.java
(checkCallingOrSelfPermission(), checkPermission())
IActivityManager Interface
Permission | Uid | Pid
ActivityManager Native Binder
ActivityManagerService.java
(checkPermission(), checkComponentPermission())
PackageManagerService.java
(checkUidPermission(), checkPermission(), checkComponentPermission())
Com.android.server.pm
GrantedPermission
Hashset
XML

Figure 1: Existing Android application framework for checking permissions

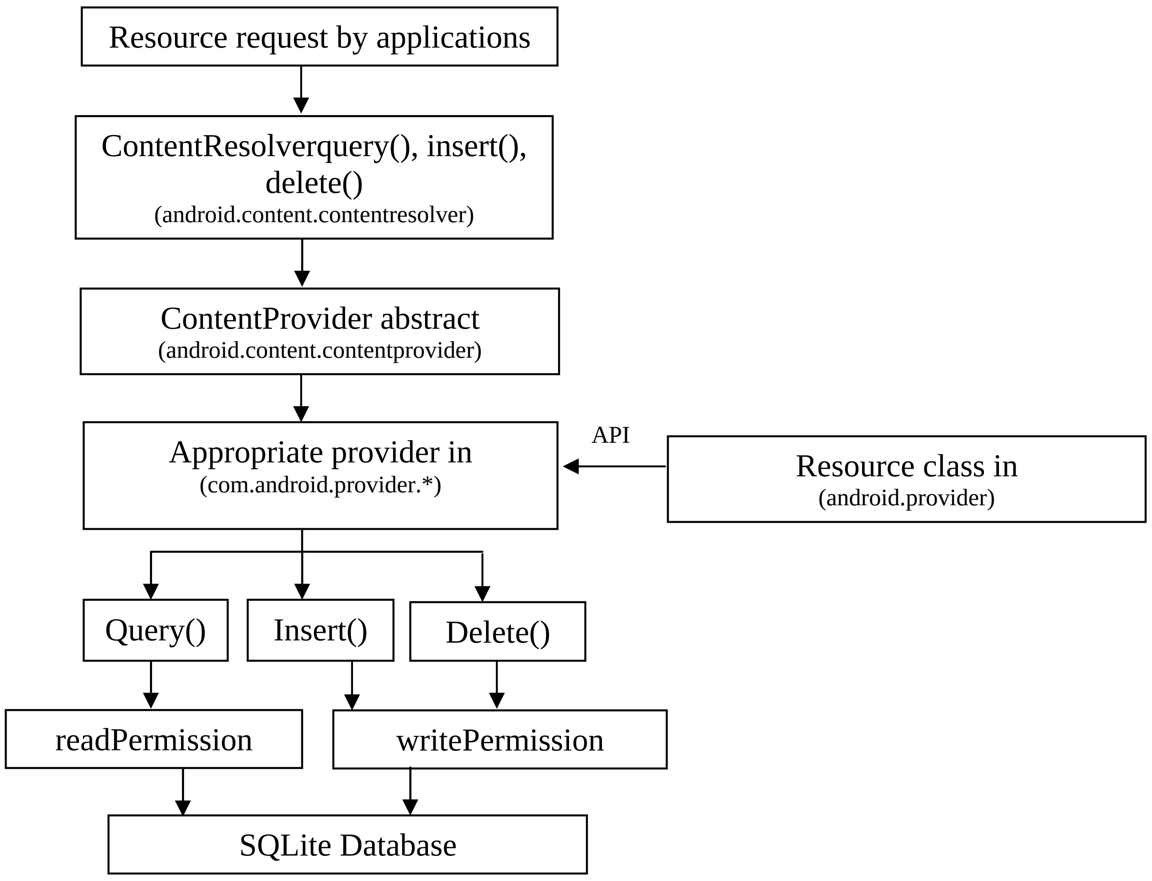


Figure 2: Existing framework for accessing resources of Content Providers

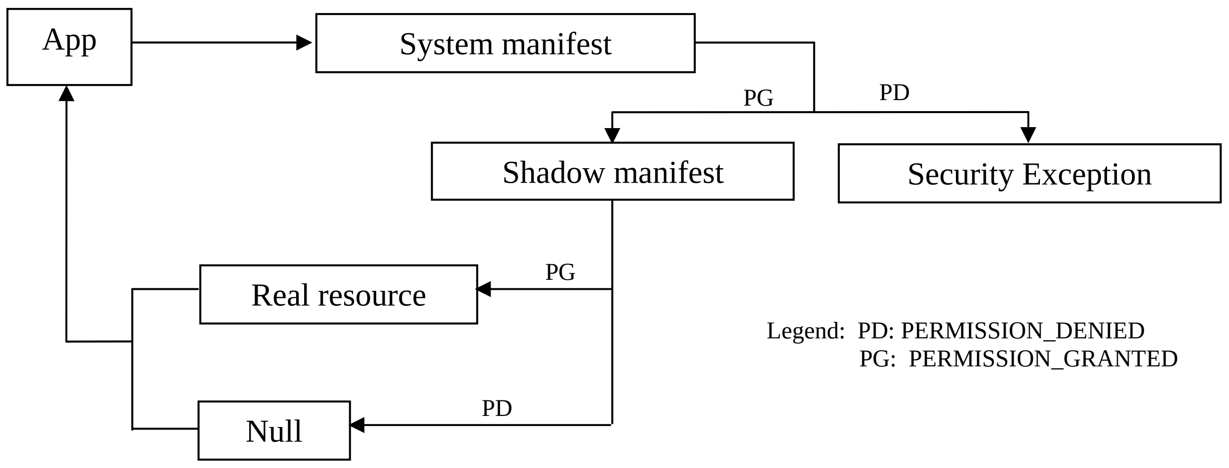


Figure 3: Control flow through the proposed Shadow Manifest

4.2 Proposed Shadow Manifest Solution

In this paper, Selective Permissions is achieved by the proposed shadow manifest solution. The control flow through the proposed shadow manifest is given in Figure 3.

An application in Android is given the resource it has requested, only if the system manifest contains the corresponding permission entry for that particular resource. If the entry is absent then a security exception is thrown and the application terminates abnormally. The proposed shadow manifest is positioned in the framework after the system manifest check. The control is made to flow through the shadow manifest after the system manifest check. Only if the corresponding 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, making the application believe that the user has not recorded any data.

4.3 Algorithm

Given: Shadow manifest file and the system manifest file.

Aim: To deny permissions to the application as per the user requirement.

Revoke: Set the state of permissions to be revoked to 0 in the shadow manifest.

Check:

- If system (p(x)) and if (shadow (p(x)) then grant real resource to application x.
- If system (p(x)) and if !(shadow(p(x)) then grant the empty resource to application x.
- If system (p(x)) is false then the application x terminates abnormally.

Note: System (p(x)) denotes a function which tests if the application x has a given permission p by looking at the system manifest file. In other words, p does the permission checking in the existing Android framework. Shadow (p(x)) denote a function which tests the permission p by looking at the shadow manifest file, which is present in the content provider database.

4.4 Implementation

The Android source code needs to be installed on a Linux machine and be built to obtain the system image which is pushed to the emulator.

For installing an application in Android OS, the steps to be done are the following:

1) Copy the application folder into packages/apps folder.

2) Create Android.mk file in the application folder.

3) Add the name of the application in the core.mk file in build/target/product.

4) In the command line, type source build/envsetup.sh.

5) To set the paths execute the setpaths command.

6) Go to the application folder and put mm to sync the application.

7) Execute lunch full-eng command to initiate the emulator, make –j4 to build and run the emulator.

4.4.1 Creation of Log

Creation of Log is shown in Figure 4. The checking of permissions during the resource request happens in the PackageManagerService (frameworks/base/services/java/com/android/server/pm/PackageManagerService.java). 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.

4.4.2 Shadow Manifest Implementation

Sel_Perm application implemented by us displays the list of applications and the permissions granted to the applications. When a user wants to revoke any permission, that particular permission is to be unchecked for the application. The changes made through this GUI are stored in a database which is referred to as the shadow manifest file.

This database, if created from the application, then it cannot be used in the Android Application framework and if it is created from the Android OS then it cannot be referenced by the application. Thus, the database needs to be embedded in a Content Provider for providing the content to many applications. Content Providers can be created from the Android applications or the Android OS. If created from the Android application, then the Content Provider is stored in the work space of the

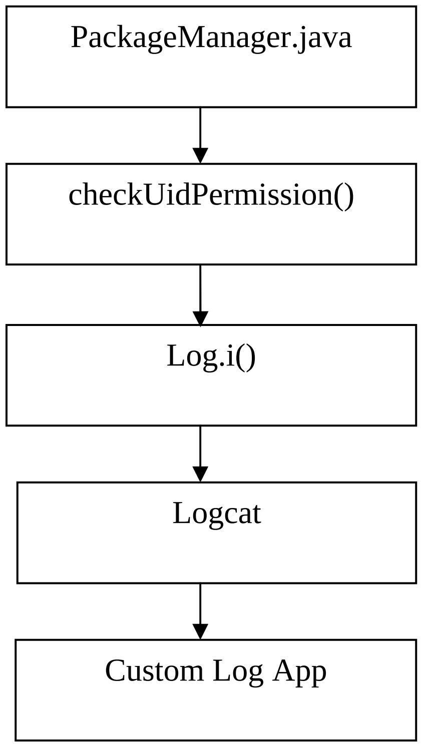


Figure 4: Creation of log

application and hence is inaccessible from the Android OS. Therefore, the Content Provider needs to be created in the OS.

SelPermProvider, created for this purpose, has a database which 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 (The package name of an application is the same and unique in all Android phones).

4.4.3 Illustration

The permissions database accessible through the content provider ASelPermProvider, can be in any one of the three states with respect to a permission of an application. Consider an application with UID X that has been granted a permission P during installation. The three states in the database are as follows:

State a: Permission P entry for application X is not found in the database.

This state happens when the user has not unchecked the permission P for application X in the Sel_Perm application installed in the Android Phone. Thus an entry is not created in the permissions database for that particular permission of application X. When the control flows through the shadow manifest and the check for granting the resource which requires permission P is done, the real resource is granted to the application X. This is because the user has not revoked the permission P for application X.

State b: Permission P entry for application X is found in the database with state zero. The tuple for this state is (id,X,P,0), where id is the position of this record from the beginning of the database.

This state happens when the user has unchecked the permission P for application X in the Sel_Perm application installed in the Android Phone. Thus an entry is created in the permissions database for that particular permission of application X with state zero. When the control flows through the shadow manifest and the check for granting the resource which requires permission P is done, the null value is returned to the application. This is because the user has revoked the permission P for application X. The application X is unaware of this indirect permission denial and it unsuccessfully completes.

State c: Permission P entry for application X is found in the database with state one. The tuple for this state is (id,X,P,1), where id is the position of this record from the beginning of the database.

This state happens when the user has rechecked the permission P for application X in the Sel_Perm application installed in the Android Phone, as in, granting the permission P back to the application X. Thus the already existing entry for permission P of application X in the permissions database is updated with state as one. When the control flows through the shadow manifest and the check for granting the resource which requires permission P is done, the real resource is granted to the application X. This is because the user has re-granted the permission P to the application X.

4.4.4 Installation of Provider

The custom provider ASelPermProvider is installed in the Android OS by the following steps:

i) Create an API which defines the structure of the provider. In our case, ASelPerm.java class defines the API for the ASelPermProvider.

ii) Create the content provider ASelPermProvider by extending the contentprovider abstract class. All the methods in the contentprovider abstract class have been overridden in the custom contentprovider.

iii) Create the androidmanifest.xml file for the ASelPermProvider. This file contains the read permission and write permission definitions for accessing the ASelPermProvider.

iv) Create a Android.mk file, which contains the definitions for the target of the android and final build system commands to generate that target.

v) Register the provider in the build file core.mk (/build/target/products)

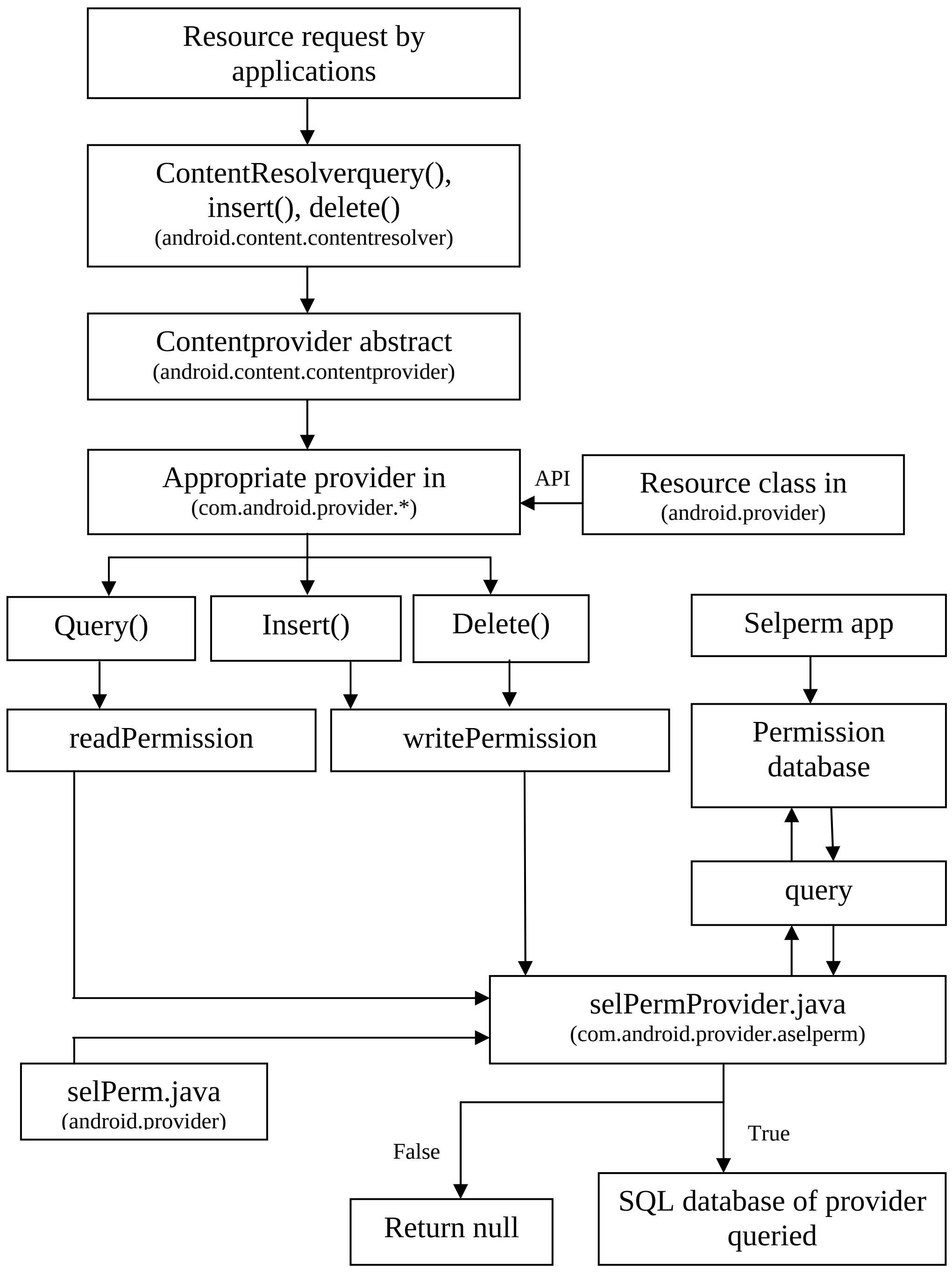
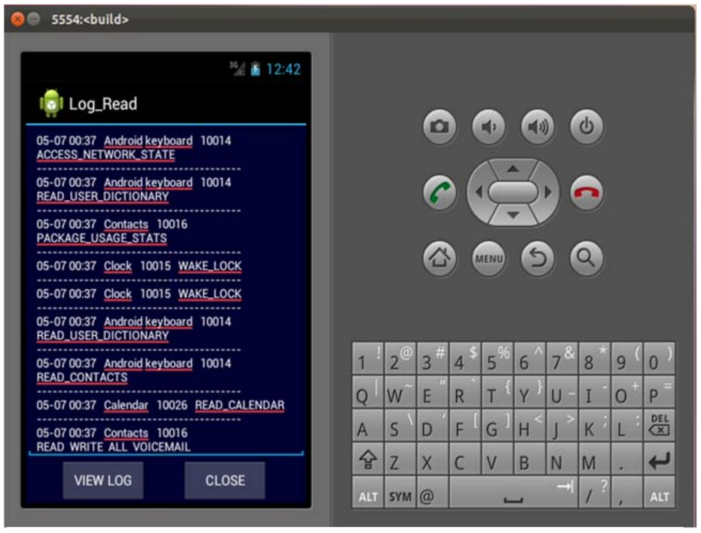


Figure 5: Realization of shadow manifest file

4.4.5 Summary of Implementation details

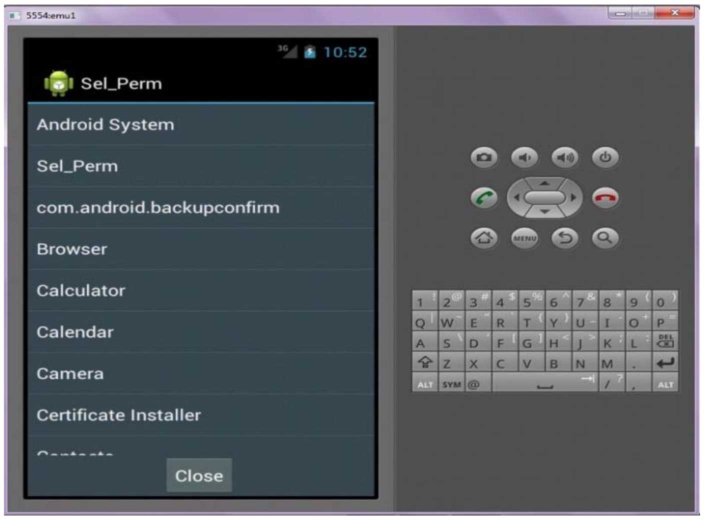
Table 1: Summary of Implementation

MODULE	FUNCTION NAME	ARGUMENTS	CALLS	RETURNS	REMARKS
LOG_READ ACTIVITY GUI	onCreate	Bundle savedInstanceState	onClick	void	Sets up activity by creating views, binding data, etc.
	onClick	View v	extractLog	void	Handles the event of clicking the "ViewLog" button
	extractLog	Nil	getUid	String	Finds the log entries created by the Package Manager with Selective Permissions tag
	getUid	String uid	Nil	int	Gets the uid of the application from the log entry
SEL_PERM ACTIVITY GUI	onCreate	Bundle savedInstanceState	getAppList	void	Sets up activity by creating views, binding data, etc.
	getAppList	Nil	Nil	String[]	Gets the list of installed applications in the android phone
	onItemClick	AdapterView<?> parent, View v, intpos, long id	Nil	void	Sends an intent to launch thepermissionList activity
PERMISSION LIST ACTIVITY GUI	onCreate	Bundle savedInstanceState	getPermList, dynamicview	void	Sets up activity by creating views, binding data, etc.
	getPermList	String appname	Nil	List<String>	Gets the permissions for the given application
	dynamicView	List<String>permList	ASelPerm Provider query	void	Sets up the view with permission checkboxes
	onClick	View v	databaseFunction	void	Handles the event of clicking the "Save" button
	databaseFunction	Nil	ASelPermProvider update, ASelPermProvider insert	void	Updates or inserts the permission entries in the permissions database of aselpermprovider
ASELPERM PROVIDER BACKEND	onCreate	Nil	Nil	boolean	Instantiates an object of databasehelper
	query	Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder	Database System query	Cursor	Queries the contentprovi-der
	getType	Uri uri	Nil	String	Specifies type for single or multiple row outputs
	insert	Uri uri, ContentValues initialvalues	Database System insert	Uri	Inserts rows in contentprovider
	delete	Uri uri, String where, String[] whereArgs	Database System delete	int	Deletes rows from contentprovider
	update	Uri uri, ContentValues values, String where, String[] whereArgs	Database System update	int	Updates rows in contentprovider



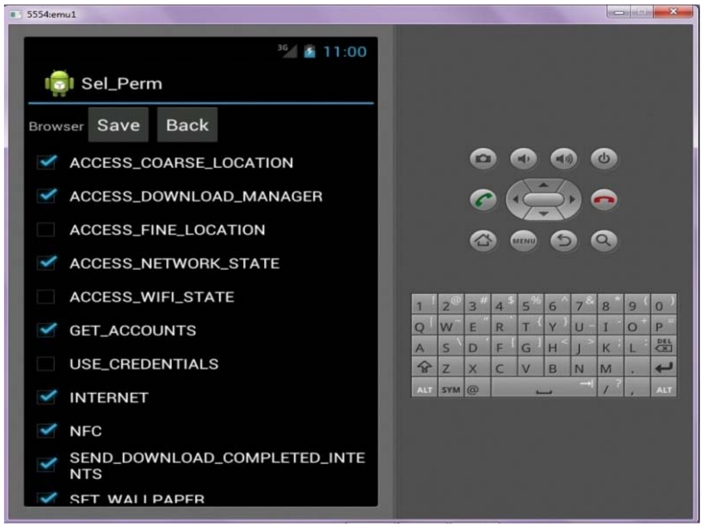
```
05-07 00:37  Android keyboard  10014
ACCESS_NETWORK_STATE
------------------------------------
05-07 00:37  Android keyboard  10014
READ_USER_DICTIONARY
------------------------------------
05-07 00:37  Contacts  10016
PACKAGE_USAGE_STATS
------------------------------------
05-07 00:37  Clock  10015  WAKE_LOCK
------------------------------------
05-07 00:37  Clock  10015  WAKE_LOCK
------------------------------------
05-07 00:37  Android keyboard  10014
READ_USER_DICTIONARY
------------------------------------
05-07 00:37  Android keyboard  10014
READ_CONTACTS
------------------------------------
05-07 00:37  Calendar  10026  READ_CALENDAR
------------------------------------
05-07 00:37  Contacts  10016
READ WRITE ALL VOICEMAIL
```

Figure 6: Screenshot of Log_Read application



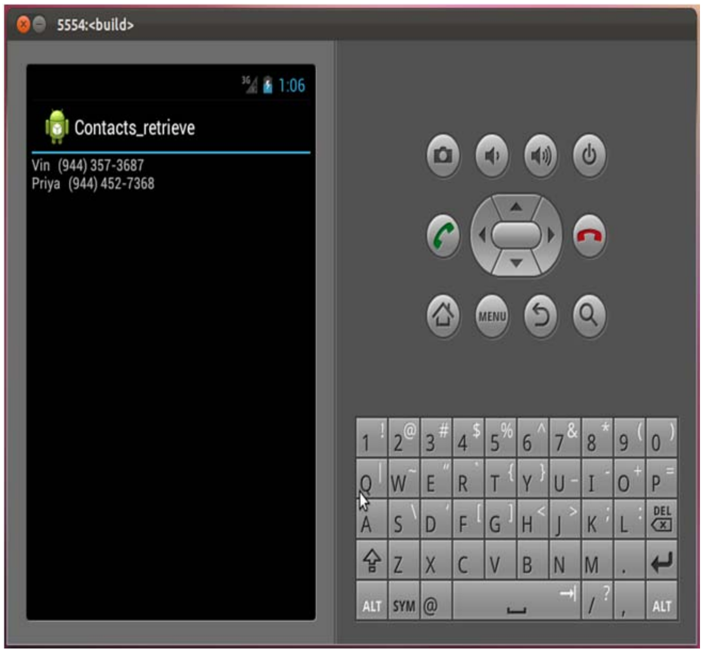
Android System
Sel_Perm
com.android.backupconfirm
Browser
Calculator
Calendar
Camera
Certificate Installer

Figure 7: Screenshot of Sel_Perm application-1



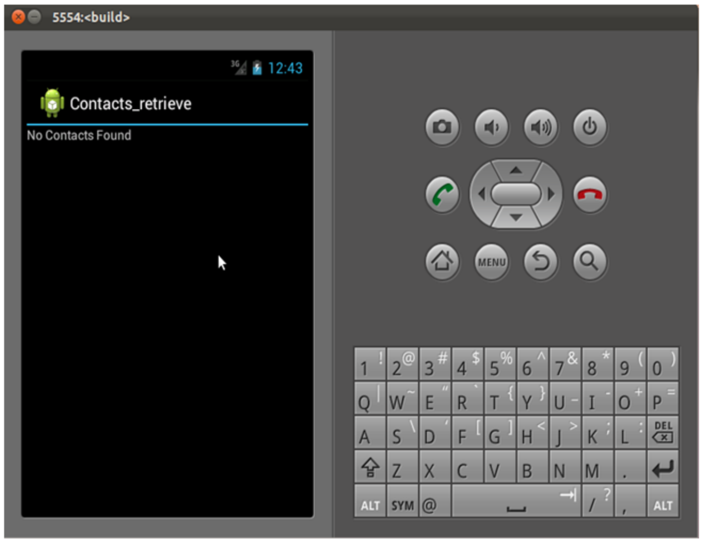
- [✓] ACCESS_COARSE_LOCATION
- [✓] ACCESS_DOWNLOAD_MANAGER
- [ ] ACCESS_FINE_LOCATION
- [✓] ACCESS_NETWORK_STATE
- [ ] ACCESS_WIFI_STATE
- [✓] GET_ACCOUNTS
- [ ] USE_CREDENTIALS
- [✓] INTERNET
- [✓] NFC
- [✓] SEND_DOWNLOAD_COMPLETED_INTENTS
- [✓] SET WALLPAPER

Figure 8: Screenshot of Sel_Perm application-2



Vin (944) 357-3687
Priya (944) 452-7368

Figure 9: Screenshot of Contacts_retrieve application-1



No Contacts Found

Figure 10: Screenshot of Contacts_retrieve application-2

4.4.6 Experiments and Results

The Log_Read, Sel_Perm applications were installed in the Androidemulator and the ASelPermProvider was incorporated in the 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.

Contacts_retrieve is an application which gets the contacts details. Figure 9 shows the Contacts_retrieve when it has READ_CONTACTS permission. When it has the permission, the application works. Figure 10 shows the Contacts_retrieve application when the READ_CONTACTS permission is unchecked. The application then shows that there are no contacts (as null value is returned). It still works, without terminating abnormally.

5. CONCLUSION

The "all or none" permission policy in the current Android Application framework poses a barrier to detect malicious applications. The selective permission policy suggested 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. To distribute the version of Android with this patch included, the source code should go through the Android source code review approval.

6. REFERENCES

[1] Android open source project, Application Sandbox, http://source.android.com/tech/security/index.html.

[2] Android open source project, Elements of Applications, Interprocess Communication, 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-application-framework-docs-tuts.

[7] GrepCode, http://grepcode.com/snapshot/repository.grepcode.com/java/ext/com.google.android/android/4.0.3_r1/