Pers Ubiquit Comput (2012) 16:631–641
DOI 10.1007/s00779-011-0437-6

ORIGINAL ARTICLE

An Android runtime security policy enforcement framework

Hammad Banuri · Masoom Alam · Shahryar Khan · Jawad Manzoor · Bahar Ali · Yasar Khan · Mohsin Yaseen · Mir Nauman Tahir · Tamleek Ali · Quratulain Alam · Xinwen Zhang

Received: 30 January 2011 / Accepted: 30 May 2011 / Published online: 31 July 2011

Abstract Today, smart phone's malwares are deceptive enough to spoof itself as a legal mobile application. The front-end service of Trojans is attractive enough to deceive mobile users. Mobile users download similar malwares without knowing their illegitimate background threat. Unlike other vendors, Android is an open-source mobile operating system, and hence, it lacks a dedicated team to analyze the application code and decide its trustworthiness. We propose an augmented framework for Android that monitors the dynamic behavior of application during its execution. Our proposed architecture called Security Enhanced Android Framework (SEAF) validates the behavior of an application through its permissions exercising patterns. Based on the exercised permissions' combination, the mobile user is intimated about the dangerous behavior of an application. We have implemented the proposed framework within Android software stack and ported it to device. Our initial investigation shows that our solution is practical enough to be used in the consumer market.

Keywords Android security · Permission labels · Smart phone malwares

H. Banuri · M. Alam (✉) · S. Khan · J. Manzoor · B. Ali · Y. Khan · M. Yaseen · M. N. Tahir · T. Ali · Q. Alam
Security Engineering Research Group (SERG), Institute of Management Sciences, 1-A, E-5, Phase VII, Hayatabad, Peshawar, Pakistan
e-mail: masoom.alam@imsciences.edu.pk

X. Zhang
Huawei Research Center, Santa Clara, CA, USA
e-mail: Xinwen.Zhang@huawei.com

1 Introduction

The advanced technological features (such as increased processing power, higher memory, extended screen display, etc) of today's smart phone certainly paves the way for developer to built more attractive and innovative smart phone applications. These applications utilize some cost based and privacy sensitive services such as SMS, MMS, telephony, GPRS, camera, and GPS, etc. The use of such delicate resources prompts the consumer to ensure her privacy and legitimate use of her paid services. Consumer expects the base framework of mobile device to be secure enough or at least aware enough to inform the consumer about any potential harm of an application developed by third-party application developers. Security issues in smart phones are becoming more threatening similar to desktop systems. In last few years, a number of malwares have been designed to target and attack the mobile devices [1, 2].

Android is first comprehensive open-source mobile operating system destined toward consumer market. It certainly grasps the attention of some European and US service providers because of their attractive features. Our implemented framework targets Android, because of its worldwide acceptance, its knowledge base of security policies, and its openness. The openness of Android leads to rapid growth, which certainly give boom to Android application market. However, Android freeware applications are available on Internet in huge number, which makes Android vulnerable to security attacks. Apple app store bounds the application availability with their approval system [3]. Applications are analyzed and evaluated for security concerns before making them available to consumer on Apple app store. There is no such security evaluation mechanism available for Android. It leads the Android customer to unknowingly download malicious applications.

Android provides primitive and intuitive architecture for securing applications installed on it. Developers can provide their initial policy to restrict access of different components of their applications. However, existing Android framework does not enforce any security policy to ensure the trustworthiness of application. Once an application gets installed and system resources are allowed to access, then any mistrusted installed application could act maliciously and system would not investigate dynamic behavior to ensure application and system integrity. Moreover, Android follows all-or-nothing permission model. It means that in order to make successful installation of an application, user has to accept all permissions (that are requested at the time of installation), otherwise installation process would be terminated. There could be a situation when a user wants to customize the permissions assigned to the installed application by allowing some permission and denying others (keeping in view constrain of core functionality) (see Sect. 2.5).

We provide an extended framework for Android to mitigate malwares and Trojans that disguise consumer with their deceptive apparent functionality. In the design, evaluation, and implementation of proposed scheme, this article makes following contributions.

- We identified a couple of major limitations in Android regarding its security policy and access control, respectively. Android does not perform security evaluation to mitigate malwares. The resources acquired by applications in Android can only be revoked by uninstalling corresponding application.
- We build some realistic policies to analyze the behavior of applications. These policies are dynamic in nature and provide an effective means to evaluate the behavior of application during its execution. The policies are based on the knowledge base provided by Android system, which makes them efficient and compatible with the system.
- We provide retrofitting runtime security evaluation engine to mitigate any type of malwares that could misuse the system or other application resources.
- Our implemented framework also equipped user with more fine grained access control and makes her capable of dealing with customization of access permissions after installation.

The rest of the document is organized as follows. The next upcoming Sect. 2 discusses the background of Android and its security policy enforcement mechanism. In Sects. 3 and 4, we discuss the proposed framework and target architecture, respectively. Section 5 discusses implementation, and Sect. 6 is related work, finally Sect. 7 concludes the paper.

2 Background

2.1 Android layered architecture

Android is the first comprehensive open-source mobile platform. Its architecture comprises of four layers. Android applications are placed on top of the Android layer stack, which are supported by underneath three layers that include application framework, Android runtime, and Linux kernel.

Linux kernel is used as an abstraction between hardware and the remaining software stack of Android. Android rely on kernel for managing low-level system resources such as memory management, security model, network stack, and process management. Android runtime's core component is Dalvik virtual machine, specifically designed for Android platform and optimized for mobile devices. Each Android process runs its own separate instance of Dalvik VM [4]. Application framework is a built-in toolkit in Android to provide different set of services to its applications. The inter application communication (normally referred as IPC—Inter Process Communication) is mediated by application framework.

2.2 Android application structure

Applications of Android are written in java, but the execution of java byte code is not directly supported by Android. Android system uses a dx tool that converts java code into Dalvik understandable byte code. All applications in Android are identified with a unique Linux user ID (UID), which allow Dalvik to run multiple applications, each in a separate process. However, multiple applications could share single UID (through `SharedUserId` attribute) and hence can run simultaneously into single process of Dalvik VM [5]. The applications sharing same UID have to be signed with a similar signature (developer certifies application with a signature, which identifies the developer of application). The UID mechanism also provides sandboxing and limits the potential programming damage [6]. An Android application may comprise of four components that are activity, service, broadcast receiver, and content provider. Activity is a visual screen through which user interact with application. It could be list of objects, for example, labels into top up menu or multiple images displayed into dialog box on your mobile screen, etc. An application may consist of one or more activities depending on their architecture and design. Service does not have visual interface. It runs into background, such as play back music, fetching of data over network, calculation of an expression (whose return result will be shown through an activity, as service is nothing to do with visual contents), etc. Broadcast receiver receives broadcast announcements and react according to the arise situation. Android system makes broadcast announcements, such as time zone has changed, picture has been captured, language preferences have changed, or battery power is low. Finally, content providers are normally SQLite based databases that support the sharing and accessing of data among applications. Android developer could also build content provider using Android file storage method or XML. Each Android application has its own `AndroidManifest.xml` file, where along with some other information application declares its components [7].

2.3 Android security policy enforcement mechanism

Reference monitor is a core component of Android mandatory access control—MAC, which is responsible to regulate the inter component communication (ICC) [8]. The basic mechanism behind ICC mediation is a labeling system. Application's own resources (components) are protected by assigning access permission labels, while application also holds the collection of labels which is use to specify the other application's components that it would access during its execution. Whenever, a component wants to initiate ICC, the reference monitor checks the application's access permission labels' collection. If the target component's label is in that collection, ICC establishment is allowed to proceed otherwise, it is denied [6].

In Android, a component can interact with other component within same application and with components of other application as well using a specialized ICC mechanism based on intents. Intent is a message passing mechanism that contains description of action to be performed. Intent can either be sent to a specific component (called explicit intent) or broadcast to the Android framework, which then passes it on to the appropriate component (called implicit intent). Action strings are use to specify the intents implicitly; actually, action string presents the type of action to be performed, and then Android system decide which component is suitable to perform such action [6].

In addition to declaration of components, `AndroidManifest.xml` file also contains some meta-information related to security policy enforcement of Android. *<Uses—Permission>* tag of manifest file mentions components or other resources that application would need to access during its execution. The *<Permission>* tag presents application's own components that other application or component can access. The *<Intent—Filters>* presents those intents that an application can resolve. Besides that, there are four protection levels for permission labels that are declared into manifest file. (1) *Normal*—Permissions are granted by the system without explicit approval of user. (2) *Dangerous*—These permissions are granted at the time of installation after user approval. If the protection level is "*dangerous*," then user has two options; either accept all requested permission or deny all and quit the installation process. (3) *Signature*—System grants these permissions if the requesting and granting application both have same certificates. (4) *Signature System*—System grant it to packages in the Android system image or that are signed with the same certificates.

Some of the Android system resources are accessed through components, while some are directly accessed through their APIS (Such as GPS, network, microphone, etc). Android system provides direct API access to those services that indirectly access the system hardware. Like application components, APIS are also protected through Labels that are defined into manifest file.

2.4 Limitations

The Android current access control mechanism enforces some primitive MAC's functionality. We have observed a couple of limitations in Android, regarding its security policy and access control, respectively.

1. Android permission model lacks a mechanism for customization of permissions and follows all-or-nothing policy enforcement model. It means that user has to accept all permissions (mentioned by an application during installation) to make the installation of an application successful. The user has no option to grant or deny a certain permission requested by an application.
2. Android does not investigate runtime behavior of applications to prevent them from malfunctioning and ensure system integrity.
   We propose an extended framework to existing Android architecture in order to deal with runtime behavior evaluation of any third-party developer application to ensure its trustworthiness. We also equipped the mobile user to customize the permission assignment and bypass the existing Android all-or-nothing policy model.

2.5 Motivating example

We have selected an astrology application in order to demonstrate previously addressed limitations in existing Android system. It helps in understanding the importance of dynamic behavioral evaluation of applications and customization of access permissions. The astrology application needs three permissions, i.e., `INTERNET` and `READ_CONTACT` along with `SEND_SMS`. It gathers latest description of a selected star from Internet through RSS feeds and then allow user to share her star information with their friends. Apparently it looks fine that such sort of any legitimate application would need INTERNET, READ_CONTACT, and SEND_SMS permissions. However, any illegitimate application that spoofs itself as a valid astrology application could practice the info scavenges attack (such as leaking out your personal contacts information from your mobile's address book) by exercising such permissions. The combination of access permissions assigned to astrology application looks dangerous and at the same time it could be safe depending upon the application trustworthiness. It shows that trustworthiness of an application cannot be precisely evaluated through static analysis of access permission combinations at installation time [1]. There should be some runtime evaluation mechanism that observes the behavior of application precisely during its execution. Secondly, user may want to revoke some permission from astrology application without disturbing its core functionality. Astrology application core function is to provide horoscope information rather than sending such information to friend. In this case user should be allowed to revoke READ_CONTACT and SEND_SMS permissions from astrology application in order to keep her privacy intact.

For the sake of brevity, in rest of the paper we use two abbreviated terms. (1) APR—Access Permission Request refers to the request generated by an application to access resources, (2) APS—Access Permission Sequence refers to sequence in which the permissions are exercised or in other words the sequence of APR.

3 Proposed scheme

Android does not investigate application runtime behavior to ensure its trustworthiness and prevent any application from malfunctioning. We propose Security Enhanced Android Framework (SEAF) to analyze the dynamic behavior of an application during its execution.

The proposed technique informs user about any potentially harmful activity of an application before it takes place, so that user could restrict access of an application to delicate resources at runtime. Moreover, SEAF also enrich user control by allowing customization of permissions after installation and bypass the Android existing all-or-nothing policy enforcement model.

3.1 Methodology

Each application running on Android accesses other applications or system resources through a set of permissions. As discussed previously, these permissions are defined in application's corresponding manifest file. We define a function for mapping applications to its required set of permissions.

Definition 1 Application access resources through a set of permissions. We define function that maps an application A to a set of those requested permissions P.

$$\beta(A) \rightarrow (P)$$
$$where \quad P = \{P_1, P_2, P_3, \ldots. P_n\}$$

In current Android permission model, the granted permissions are always equal to the set of requested permissions; otherwise, the application is not installed. Our proposed model provides flexibility to the framework by introducing customization of permissions for an application's requested permissions. Hence, we define granted permissions by P' as:

$$P' \subseteq P$$

Definition 2 An application runtime behavior graph is a directed acyclic graph (S:P) where S is a set of nodes representing states of an application, and P is set of edges representing different permissions exercised. For example, we write $P_1 \rightarrow P_2 \rightarrow P_3$ to describe an application that has exercised permission p1, followed by p2 and p3.

Consider a couple of permissions, i.e., READ_CONTACTS and INTERNET that could be required by any third-party application for its legitimate functionality, and at same time, these combinations of permissions could be harmful if they are exercised by any malicious application in a specific sequence. If an application exercise the INTERNET permission first and then access the READ_CONTACTS, it could be legal requirement of an astrology application (discussed in Sect. 2.5) in order to get astrology information from Internet and then SMS such information to your friend from your phone's contact list. On other hand, if any third-party malicious application that is spoofing itself as legitimate astrology application could read one's phone book by exercising permission READ_CONTACTS first and then access INTERNET to leak out personal contact information.

Similarly INTERNET, READ_CONTACTS, and CALL_PHONE is the combination of permissions that could be either illegal or legal depending upon the trustworthiness of application. Any legal VOIP similar applications need to exercise INTERNET, then READ_CONTACTS, and finally CALL_PHONE in order to establish VOIP call. Whereas, any malicious application that pretending itself as a legal VOIP application can misuse same combination of permissions. Such malicious application could exercise READ_CONTACTS first in order to have one's phone book information and then exercise INTERNET and CALL_PHONE, respectively, to disclose user's credentials (such as names, phone numbers, addresses, email ids, etc.) and at same time establish a VOIP call to deceive consumer with its make up service. Figure 1 depicts a cyclic graph (i.e., divided into two

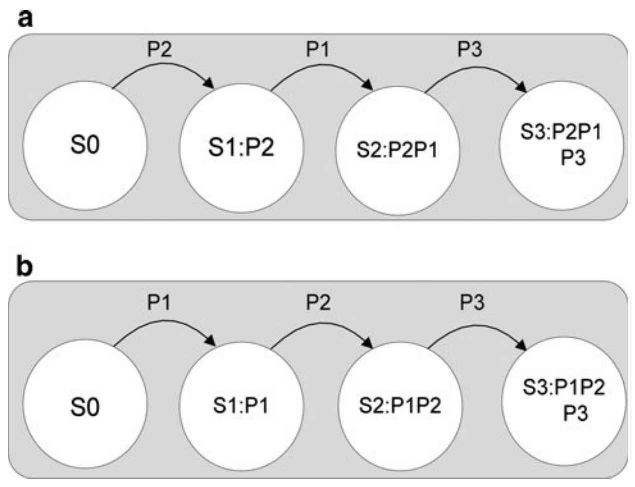

Fig. 1 Shows the harmful and benign sequences of access permissions, where *S* and *P* stand for state and permission, respectively. P1, P2, P3 represents `INTERNET, READ_CONTACTS`, and `CALL_PHONE` permissions correspondingly

portions, 1a and 1b, respectively) that presents harmful and benign sequence of permissions.

Definition 3 We define δ as a function that represents specific sequence of permissions that could be either H (Harmful) or B (Benign).

Generally $\delta(P_1 \rightarrow P_n) = H \vee B$

Our proposed scheme investigates sequence of permissions at runtime and inform user about any potential harmful activity before it actually occurs. User then could stop such malicious activity by restricting access to system resources. The proposed scheme also enriches user control over application's resources and it enables user to grant or revoke permissions from applications at runtime and also let user to decide the application's mode. We introduce a couple of application modes to deal them in a different way at runtime.

Unrestricted: The trustworthy applications are put into this mode. All the Android base framework applications are by default assigned this mode. User can also declare any third-party developer application as unrestricted. The applications placed under this category would not be analyzed at runtime. The purpose of this mode is to reduce the processing burden by avoiding analysis of trustworthy applications.

Restricted: This mode has been introduced for applications, whose trustworthiness is unknown to user. Initially, users are recommended to put any third-party developer application into this mode. The behavior of this group of application would be judged and analyzed at runtime. Whenever any application access potentially harmful sequence of access permissions, user would be informed about the potentially dangerous activity. In restricted mode, user can also perform customization of access permissions and by pass the existing all-or-nothing policy enforcement model.

3.2 Discussion of selected policies

This section briefly describes some of the policies that are included in our proposed model. A policy could be binary or ternary sequence of access permission. Each policy is indicating potentially harmful sequence of access permissions. For the sake of brevity, we have selected five policies (shown in Fig. 2) to discuss their importance with respect to sequence. Note that our proposed architecture targets those malware applications that spoof themselves as a legitimate application, and consumer download such applications keeping in view their legal functionality. The description of each policy has been elaborated considering their legal and illegal aspect. The later mentioned policies ensures that consumer should be aware of any application when it approaches potentially dangerous sequence of access permissions.

RECORD_AUDIO, INTERNET: This sequence of permission could be exercised by any potential voice tracker. While on other hand, any legal VOIP application (such as Skype, Vonage, and VOIPStunt, etc.) exercise `INTERNET` first to establish VOIP call and then record the call by accessing RECORD_AUDIO permission. Formally,

$\delta_1(P_1 \rightarrow P_2) = H$
$\delta_2(P_2 \rightarrow P_1) = B$

READ_CONTACTS, INTERNET, SEND_SMS: As discussed in earlier section that an astrology application exercise the `INTERNET` permission first to gather stars information from Internet and then `READ_CONTACTS` and `SEND_SMS` to message such astrology to your friend. On the other hand, the information scavenges attack is also possible, if permission is accessed in a specified harmful sequence presented by second policy.

$\delta_1(P_1 \rightarrow P_2 \rightarrow P_3) = H$
$\delta_2(P_2 \rightarrow P_1 \rightarrow P_3) = B$

ACCESS_FINE_LOCATION, INTERNET and ACCESS_FINE_LOCATION, SEND_SMS: These are

```
1. RECORD_AUDIO, INTERNET
2. READ_CONTACTS , INTERNET,SEND_SMS
3. ACCESS_FINE_LOCATION , INTERNET
4. Receive_SMS, WRITE_SMS, Send_SMS
5. READ_CONTACTS , INTERNET,CALL_PHONE
```

Fig. 2 Shows the five selected policies, where each policy presents a potential harmful APS

APSs that approached by malware location tracker, while there alternate sequence could be legally utilized by applications, such as SMS Auto Replier [9]. Note that these are potentially harmful APSs, which could also be required by any legitimate application. Such as any location-based media update application would exercise `ACCESS_FINE_LOCATION` first to find out the coordinates of GPS and then access `INTERNET` to gather information related to consumer location.

$$\delta_1(P_1 \rightarrow P_2) = H \vee B$$
$$\delta_2(P_2 \rightarrow P_1) = B$$
$$\delta_1(P_1 \rightarrow P_3) = H \vee B$$
$$\delta_2(P_3 \rightarrow P_1) = B$$

RECEIVE_SMS, WRITE_SMS, Send_SMS: We will not consider the direct pay off attack (sending of spam messages to premium numbers) that accomplish through `WRITE_SMS, Send_SMS` permissions. We suppose that an application is authorized to write and send short messages, but only those messages that are intended by consumer herself. For example, In Pakistan, message info service of Mobilink allows you to write, send, and then receive short messages. For example, consumer write "ckt," then send it to "300," and ultimately receive cricket updates. The legitimate APS is `WRITE_SMS, Send_SMS. Receive_SMS`, while an alternate illegal APS (`Receive_SMS, WRITE_SMS, Send_SMS`) could be utilized for malware hiding attack, where any legal incoming known message is converted to spam message.

$$\delta_1(P_1 \rightarrow P_2 \rightarrow P_3) = H$$
$$\delta_2(P_2 \rightarrow P_3 \rightarrow P_1) = B$$

READ_CONTACT, INTERNET, CALL_PHONE: Recall Sect. 3.1 that elaborates the same permissions' combination. Which shows that `INTERNET`, `READ_CONTACTS`, and `CALL_PHONE` could be benign requirement of any legal VOIP similar applications. On the other hand, the information scavenges attack is also possible, if permission are exercised in a specified harmful sequence presented by fourth policy.

$$\delta_1(P_1 \rightarrow P_2 \rightarrow P_3) = H$$
$$\delta_2(P_2 \rightarrow P_1 \rightarrow P_3) = B$$

It's worth noticing that permissions discussed in many of the policies do not depict complete picture of an application's functionality. In other words, the scenarios and sequential patterns discussed above previously the supporting permissions that complete the sense of application's functionality and at same do not alter the sequence credibility. For example, the VOIP application also needs permission `VIBRATION` along with other three permissions mentioned earlier. However, the permission `VIBRATION` does not have any significance in the decision of maliciousness of application.

4 Target architecture

We will discuss flow of classes and their method calls (that takes place to mediate ICC) in upcoming implementation section. This section focuses on functionality of each module introduced in our proposed framework. SEAF composes of couple of repositories (permission repository and policy repository) and two class modules—Policy Evaluator and Permission Manager. The repositories included in proposed framework are XML files.

Permission Repository stores the replica of each application's corresponding `AndroidManifest` file. The customization of permission and application mode assignment at runtime is carried out through realization of the permission repository. It also maintains each application's history profile. Caller application's exercised access permissions, and their sequences are kept by this repository. Status attribute in permission repository is associated with each permission tag, its value zero shows that user has revoked certain permission and vice versa. The Sequence attribute of permission repository is associated with each permission tag as well. It records the sequence of permissions in a way they are exercised by caller application. Note that sequences are kept for every single session separately. Upon session termination, the value of sequence attribute is reset to zero. Figure 3 shows that `AlarmClock` application has exercised the `CALL_PHONE` permission first and then `ACCESS_COARSE_LOCATION` at second. The sequence attribute having value zero shows that the respective permission has not been exercised yet in current session.

Policy Repository consists of some policies. Discussed earlier, every policy represents harmful sequence of permissions. For instance, Figure 4 shows that access permission `INTERNET` followed by `RECORD_AUDIO` is a harmful sequence and such a policy is stored in policy repository.

Permission Manager module is integrated into application framework. It enables user to grant or revoke access permissions from an application and set the appropriate mode at runtime[1]. It imports each application's `AndroidManifest.xml` file into permission repository. It updates the permission repository upon any query imposed

[1] We have modified the Android current installer in order to prompt user to assign appropriate mode to application and then after installation, Permission Manager could also be used to alter application mode assigned during installation.

```
<?xml version="1.0" encoding="utf-8" ?>
 <permissionList>
  <AppName name="Soundrecorder" Mode"Restricted">
   <PermName name="R_AUDIO" seq="0" stat="1"/>
   <PermName name="android.perm.INT" seq="0" stat="0"/>
  </AppName>
  <AppName name="Alarmclock" Mode"Restricted" />
   <PermName name="CALL_PHONE" sequence="1" status="1"/>
   <PermName name="READ_CONTACTS" sequence="0" status="0" />
   <PermName name="ACCESS_COARSE_LOCATION" seq="2" stat="1"/>
   <PermName name="INTERNET" seq="0" stat="1"/>
  </AppName>
 </PermList>
```

Fig. 3 Shows structure of permission repository

```
<?xml version="1.0" encoding="utf-8" ?>
 </Policies>
  <Policy no ="1">
   <PermName name="Rec_AUDIO" seq="1"/>
   <PermName name="INTERNET" Seq="2"/>
  </Policy>
  <Policy no ="2">
   <PermName name="READ_CONTACTS" seq="1"/>
   <PermName name="INTERNET" seq="2"/>
  </Policy>
 </Policies>
```

Fig. 4 Shows the policy repository

by user, regarding customization of access permissions and application mode.

Policy Evaluator is a core module of our proposed architecture that monitors dynamic behavior of application at runtime and notify user about dangerous sequence of access permissions. It is responsible for managing application session, evaluating APR, and interaction with user at runtime whenever it is essential.

It is worth noticing that realization of application session is necessary in order to keep track of APS in that specified session. The term session refers to the life time of application (i.e., time spam from application starts to its end). Application session is associated with its main activity's (the activity that launches the application) life cycle, which is originally managed by Android Runtime in existing Android base framework [10]. The Policy Evaluator interacts with Android Runtime to get information of application session.

Policy Evaluator keeps track of each caller application's APR (every application APR history profile maintained into permission repository) and compares them with the harmful access permission sequences stored into policy repository. Whenever caller application approaches any harmful sequence, the user is informed and asked about that

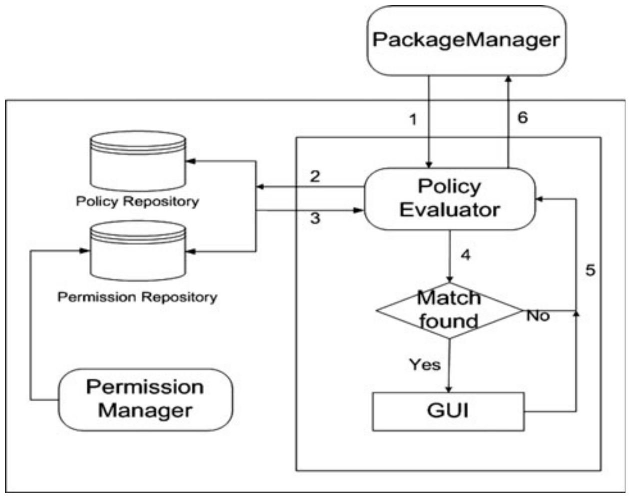

Fig. 5 Shows the proposed architecture

whether APR is allowed to proceed or not? User then decides further execution of application. Note that before investigating the harmful sequence of access permissions, Policy Evaluator checks application permission repository in order to verify that does caller application has right to access the requested resource of called application? If it is not allowed, then APR is terminated here at this point without being further investigated (Fig. 5).

5 Implementation

5.1 Existing Android permission architecture

In Android base framework, `PackageManagerService` class serves as a central point for dealing with every APR, except the APRS generated by system process or any process that belongs to root user, which is resolved earlier in `ActivityManagerService` class. Root user processes are recognized through UID value "0," while system processes are identified by UID value "1000."

Initially, `checkPermission()` method of `ApplicationContext` class receive APR from caller application. `ApplicationContext` is a common implementation of Context API, which is an abstract class that allows access to application-specific resources [11]. The incoming request is then forwarded to `ActivityManagerService` class.

The APR belonging to root user's process or any system process is resolved at this point, while APR generated by normal application components is forwarded to `PackageManagerService` class . It retrieves various kinds of information related to the application packages that are currently installed on the device. `checkUidPermission` method of `PackageManagerService` class receives the APR along with user id and process id of the calling application. The APR is then searched by PackageMangerService class into a HashSet (named grantedPermission) and is allowed to proceed if it is found there, otherwise denied. Figure 6 shows the flow and path of APR, adopted by any application component.

5.2 Modifications into Android base framework

We have placed three new hooks and introduced two additional classes—PolicyEvaluator and PermissionManger, to enhance the current security policy enforcement mechanism of Android. First hook placed in PackageParser class populates the permission status and permission sequence repositories. Second hook positioned at PackageMangerService class forward every incoming APR to newly introduced PolicyEvaluator class. Finally, third hook in ApplicationContext, transfer session information to PolicyEvaluator. These modifications introduce two new features; runtime permission customization and runtime behavior monitoring in the Android platform.

5.2.1 *Modification to incorporate customization of permissions*

We have developed PermissionManager that enable user to customize application permissions. It provides a GUI,

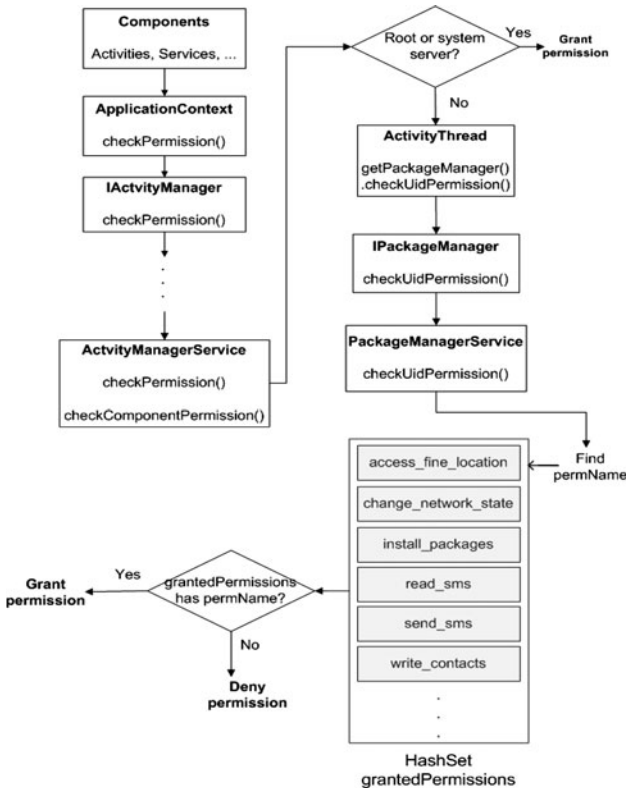

Fig. 6 Shows the path of APR in existing Android architecture

which displays a list of all applications installed on the device and their respective permissions requested by them at install time. GUI allows user to assign application mode either Restricted or UnRestricted. In Restricted mode, user can customize the permissions with the help of check box associated with each permission. The permission repository handles the process at back end. Figure 3 clearly states the application mode and shows that user has revoked the INTERNET permission from ''SoundRecorder'' application. For developers' interest, our mentioned Web link shows the steps involved in integration of PermissionManager into application framework.

5.2.2 *Runtime behavior monitoring*

PolicyEvaluator class is responsible for runtime evaluation of applications. Every APR generated by any caller application is received by PackageManagerService class, where we have placed a hook that forward every incoming APR to PolicyEvaluator class. It first checks the permission file to find out the mode of caller application. The APR generate by any unrestricted mode application is handled through traditional permission checking mechanism of Android [6]. However, if APR is generated by any restricted mode application, then checkPermissionStatus() method of PolicyEvaluator is called. If user revoked the requested permission for that specific caller application using PermissionManager, then APR is denied straight away. Otherwise, it further investigates the APR by calling isHarmful() method. It evaluates APR and act accordingly. In is Harmful() method, there are three nested functions, makePermSeq(), GetSeqList(), and UpdatePermRepo().

makePermSeq() method is responsible to create possible sequences in response to every incoming APR in order to compare them with the sequences already stored into policy repository (each policy in the policy repository contains a set of permissions with a certain sequence that can be harmful for the user's privacy or resources of the device). For example, if an application requests a permission RECORD_AUDIO followed by INTERNET & CALL_PHONE, then possible sequences that has to be considered for comparison are (INTERNET, CALL_PHONE), (INTERNET, RECORD_AUDIO), (CALL_PHONE, RECORD_AUDIO, (INTERNET, CALL_PHONE, RECORD_AUDIO).

After makePermSeq(), GetSeqList() method is called. It extracts policies (where each policy is potentially a harmful sequence) from policy repository, and then isHarmful() compares these extracted harmful sequences with the sequences generated by makePermSeq(). If a match is found, then system notifies user about the potential harmful APS. gui prompt users with couple of allow and deny radio buttons and a check box.

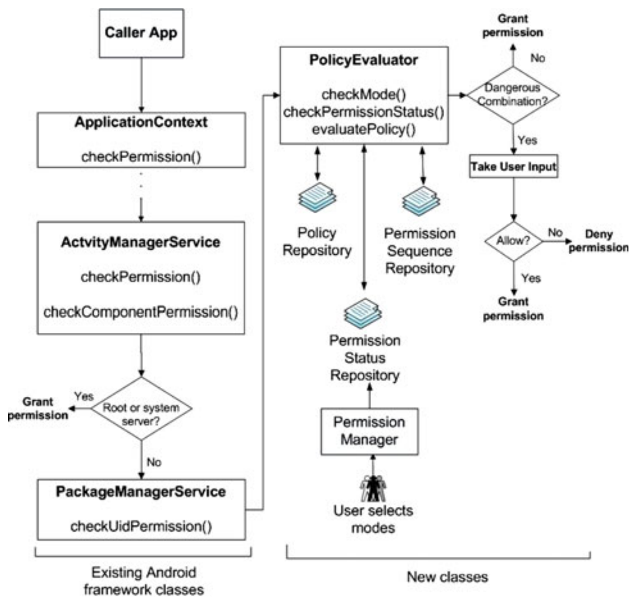

Fig. 7 Shows the implemented framework

The further execution of application is on user disposal by selecting allow or deny, while user can also report to system about not to notify for this particular APS against this current caller application in future by selecting a check box. Finally, `UpdatePermRepo()` method update the permission repository (which is responsible for maintaining history of each application's APRS). Further detail about implemented framework could be found out at SERG [15] (Fig. 7).

6 Discussion and performance evaluation

Google has released latest Android version 2.2 named Froyo in May 2010 [13]. Its source code is not yet available globally; therefore, implementation of our proposed scheme is carried out on Android version 1.5. Although Froyo has got some attractive new features (such as faster browser, efficient backup facility, allows application to be installed on external SD card, and enhanced libraries for developer's assistance, etc), but still got limitations addressed in Sect. 2.4. We believe that like Android older version 1.5, the significance of this article remains same for the newer version as well. We ported SEAF to HTC HERO, which exhibit satisfactory and feasible results.

Policy Evaluator is a core class module in proposed framework that manages incoming APR. We have discussed in Sect. 5.2.2 that `makePermSeq()`, `GetSeqList()`, and `UpdatePermRepo()` are three functions that possess major functionality in Policy Evaluator class. We have calculated the execution time for each of aforementioned methods. We developed an application that exercises seven different permissions during its execution. The noticed execution time taken by each method upon exercised permission is shown in Table 1.

Our proposed framework has a tendency to prompt user more often during execution of any restricted mode application. Whenever application approaches harmful sequence, system intimate user. Apparently, it may end up in number of dialogs intimating user to respond. We try to reduce such prompting by introducing a check box into prompting dialog(see Sect. 5.2.2). It takes user input about whether to ignore specified APS regarding concerning application in future or not. It would allow user to reduce prompting against those applications which have gained trust with the passage of time. Upon gaining complete trust of user, any third-party application could be assigned with unrestricted mode and hence user can purge prompting of such application. It is our feature work to incorporate some machine learning technique in order to reduce number of prompting further, even for those applications who are persistently kept in restricted mode.

Our proposed framework takes user input after prompting her on any suspected malicious behavior. So, if any suspected illegal APS is likely to be legitimate, then user can simply ignore it and can even restrict the system not to prompt again in future. Keeping in view that any suspected illegal APS could also be legal, alert messages are defensive enough to intimate user about potential harm rather claiming the harmful behavior. For instance, we have developed an Astrology application. Whenever it exercises `INTERNET` followed by `READ_CONTACTS`, then modified framework generates a message "Astrology may be performing unauthorized activity."

Table 1 shows execution time (in ms) of each method for 7 APRS/session

APR Number	1	2	3	4	5	6	7
makePermSeq ()	1	2	6	11	28	154	260
UpdatePermRepo ()	23	24	23	25	26	26	26
GetSeqList ()	46	46	47	43	44	45	45
Other statements and supported function	64	68	65	71	71	71	71
Total time (ms)	134	140	141	150	169	296	402

Our proposed scheme has a limitation that if an application reads the contacts information in a session and then in another session uses the Internet permission, the security framework will not be able to judge the malicious nature of the said application (if any). This limitation is in our knowledge and we can overcome it, by introducing information flows within the Android kernel. For example, once an application has read some security critical information, it may not be able to read or write such information to a public channel such as Internet. Details of such information flows can be found in [16].

7 Related work

Not much work has been done regarding ensuring validation of Android third-party applications and securing Android consumer from malicious applications. Most of the recent work regarding Android smart phone security revolves around validation of app permissions at install time. For instance, Light Weight mobile certification [1] (extended malware version of Kirin) and Kirin [15] both decides the potential maliciousness of corresponding applications at install time. The precision of a decision on the basis of install time permission acceptance is questionable. The previously mentioned schemes extract the security enforcement policy from corresponding manifest file and compare the extracted policy with their predefined invariants at installation in order to certify the application trustworthiness or maliciousness. However, none of these schemes observes runtime dynamic behavioral change of applications to ensure the trustworthiness of such third-party developer applications. Avik Chaudhuri also tried to evaluate the trustworthiness of application. Their proposed technique provides static safety analysis of application code on the basis of formal specifications of APIS provided the SDK [16]. Avik's scheme anticipates data flow by analyzing application code, rather than to analyze the data flow when it actually happens. It is another debate (and beyond the scope of this research article) that whether the application code analysis is precise, efficient, and feasible to observe the data flow, which intern use to certify the application integrity. In case of Apple app store, each application's code is analyzed by the concern team and then application is made available for iPhone customers to download; on the other side, Android is open source and third-party developer could make their application available for Android consumer without realization of their developed application code. Machigar Ongtang et al. [8] has proposed SAINT scheme to secure the Android platform. They have highlighted different aspects of security vulnerability in Android system and extended Android frame work to provide a more controlled and secure environment for applications to interact with other applications and resources they acquired. They ensure secure environment for applications, while our work evaluates the application and secure Android consumer from suffering rather than providing secure environment to application itself. In other words, we are ensuring application trustworthiness for consumer trust and safety, while they are focusing on application integrity to protect application itself from corrupting or malfunctioning.

8 Conclusion

As the consumer market for the smartphones is getting wider, the corresponding malware applications for smartphones are deceptive enough to spoof itself as a legal mobile application. These applications—Trojans are attractive enough to deceive mobile users. Mobile users download similar malwares without knowing their illegitimate background threat. Android platform is more vulnerable to malware attack, as it does not have a dedicated panel to analyze the application code. On other side, Apple Inc. analyzes applications' code before making them available on Apple app store.

In this paper, we specifically identify the limitations of Android platforms and propose a complete framework to cater these limitations. The implementation of proposed framework has been carried out on Android platform, because of its worldwide acceptance, its knowledge base of security policy enforcement, and its openness. We have briefly described the Android platform and its security mechanism. Android access control mechanism lacks the runtime evaluation of application trustworthiness and it relies on all-or-nothing policy enforcement model. We enhanced the capabilities of Android by providing a retrofitting runtime security policy enforcement framework to mitigate any type of malwares that could misuse the system or other application resources.

We built some realistic policies with the help of knowledge base provided by Android system. Each policy stores a different permissions' sequential pattern that points to some malicious or suspicious activity. When an application approaches the permissions' sequential pattern mentioned by any of the policy declared in policy repository, the user is intimated about the potential dangerous behavior of the running application. User can then stop any malicious activity before it could compromise system integrity. The augmented framework also enriches user control over Android's resources by allowing her customization of access permissions. As a proof of concept, we ported the proposed framework to Android HTC Hero; result shows that our scheme is lightweight and feasible enough to be introduced in market.

References

1. Enck W, Ongtang M, McDaniel P (2009) On lightweight mobile phone application certification. In: Proceedings of the 16th ACM conference on computer and communications security ACM, pp 235–245
2. Jamaluddin J, Zotou N, Coulton P (2004) Mobile phone vulnerabilities: a new generation of malware. In: 2004, IEEE international symposium on consumer electronics, 1–3 Sept, pp 199–202
3. Apple App Store Approval Process (2007) Available at: http://en.wikipedia.org/wiki/App. Store#Approval process, 5 March 2007
4. Khan S, Khan S, Banuri H (2009) Analysis of Dalvik virtual machine and class path library. Available at:http://imsciences.edu.pk/serg/wp-content/uploads/2009/07/Analysis-of-Dalvik-vm.pdf. Nov 2009
5. Android reference: security and permissions. Available at: http://developer.android.com/guide/topics/security/security.html
6. Enck W, Ongtang M, McDaniel P (2009) Understanding android security. IEEE Sec Privacy 7(1):50–57
7. Android reference: manifest. Available at: http://developer.android.com/guide/topics/manifest/manifest-intro.html
8. Ongtang M, McLaughlin S, Enck W, McDaniel P (2009) Semantically rich application-centric security in android. IEEE: Ann Comput Sec Appl Conf 22:340–349
9. Android Application—SMS Replier 1.61a. Available at: http://developer.android.com/guide/topics/security/security.html
10. Reto M (2008) Professional android application development, by Wrox. ISBN:978-0-470-34471-2, pp 68–73
11. Android reference: class context. Available at: http://developer.android.com/reference/android/content/Context.html
12. Security Engineering Research Group—SERG reference. Available at: http://imsciences.edu.pk/serg/projects/easip/android-runtime-security-policy-enforcement-framework/
13. Android reference: android 2.2 platform highlights. Available at: http://developer.android.com/sdk/android-2.2-highlights.html
14. Shankar U, Jaeger T, Sailer R (2006) Toward automated information-flow integrity verification for security-critical applications. In: Proceedings of the 13th annual network and sistributed systems security symposium. Internet Society, 2006
15. Enck W, Ongtang M, McDaniel P (2008) Mitigating android software misuse before it happens. Technical report NAS-TR-0094-2008, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, November
16. Chaudhuri A (2009) Language-based security on android. In: Proceedings of the ACM SIGPLAN fourth workshop on programming languages and analysis for security, pp 1–7