Swapnil Powar, Dr. B. B. Meshram / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 3, Issue 2, March -April 2013, pp.907-911

Survey on Android Security Framework

[1]Swapnil Powar [2]Dr. B. B. Meshram
[1]Computer Technology Department, VJTI, Mumbai
[2]Computer Technology Department, VJTI, Mumbai

Abstract: Smartphone with open source operating systems are getting popular now days. Increased exposure of open source Smartphone is increasing the security risk also. Android is one of the most popular open source operating system for mobile platforms. Android provide a basic set of permissions to protect phone resources. But still the security area is underdeveloped. This survey is about the current work done on the Android operating system. Some of the techniques, which can introduce a positive edge to the security area, are analyzed in the present survey paper. These techniques are basically to provide a better security and to make the Android security mechanism more flexible. As the current security mechanism is too rigid. User does not have any control over the usage of an application. User has only two choices, a) allow all permissions and application will install, b) deny all permissions and installation will fail.

Keyword: Android Architecture, APEX, Poly Android Installer

I. Introduction

Android is a Google operating system for mobile platforms with the basic functionality of system utilities, middleware in form of Virtual Machine (VM) and some core application like browser, dialer, calculator and some others as well. The large number of applications is available for user. But the user need trust full applications, which do not harm their privacy and security issues, so it is mandatory for every application to ask for permissions from the user during the time of installation. User has only two choices, either to grant all the required permissions and the application will be installed. And once the permissions are granted, Android does not provide any facility to revoke those permissions, unless the user uninstalls the application.

Consider a weather application that reads user's location from his phone and provide weather updates on the base of time and location. If the user grants the permission to the application so, the application will get install. The drawback is that, the application can collect the user location anytime, even user do not wish so. And if the user does not grant permission to the application during installation, so the application will not be install.

II. Background

Android is a Google operating system launched for mobile platforms. The current architecture of Android is explained below:

A. Android Architecture and Android Application Structure

Android architecture contains four layers. Application layer on top and the rest of three layers are application framework, Android runtime and Linux kernel respectively. Linux kernel is an abstraction of the hardware and software. Android runtime's is a core component of Dalvik virtual machine. Each Android process runs in a separate instance of Dalvik virtual machine. Every application is assign with a unique Linux user ID call as UID. This functionality allows Dalvik to run multiple applications in a separate process. Those applications who run in a single process, must share a single UID. Otherwise every application will have a separate ID.

B. Android's Components

Android composed of basic four components. ICC is used for communication between components.

Activity: Activity provides GUI for interaction of user with the application. Depends upon design, an application may consists of one or more activities

Service: Service is a background process that fetches data from the network.

Broadcast Receiver: Broadcast receiver receive broadcast announcements and response to them according to the situation.

Content Provider: Content provider is a SQLite database, which supports the sharing and accessing of data among applications.

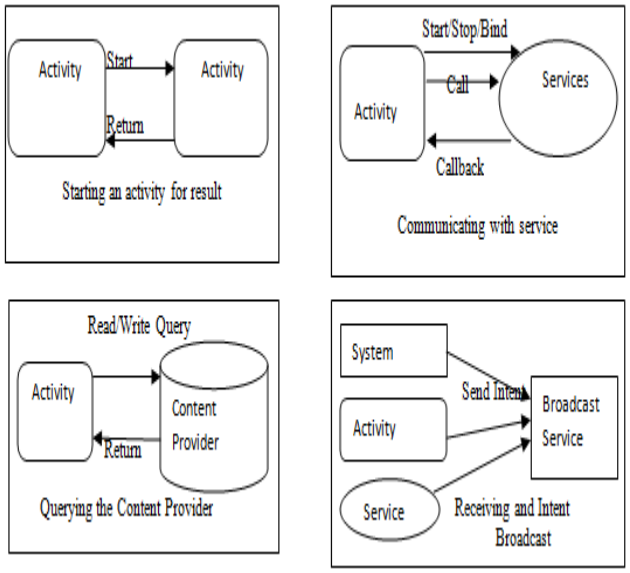

Figure 1: Typical ICC between Activities, Services, Content Providers and Broadcast Receivers

C. Android inter-components communication

In Android inter-components interaction take place using through ICC mechanism based upon intents. Intents are message passing mechanism that also contains nature of the action to be performed. Intents can be sent to a specific component or can be broadcast to the Android framework. <Intent-Filters> specify those intents which can be resolved.

D. Protection Levels

Android has four permission levels, on the basis of which an application can be installed.

Normal: Normal permissions are granted by system without asking any permission from user.

Dangerous: Dangerous permissions ask for the approval from the user at the time of installation. User has two choices, either grant all permission or deny all. Denying of permissions will stop installation.

Signature: System grants these permissions if the requesting and granting application share the same certificate.

Signature System: Same as Signature but use for system applications only.

Android permission lacks the modification of permissions. Android policy is very strict. It walks on all or nothing policy. User should allow all permission to allow any installation. Android do not provide any runtime investigation for the behavior of application.

E. Mobile Phone Threats

Proof of concept: Keeping the Bluetooth device on without the knowledge of the user is an example of this threat. It drained device batteries.

Destructive: Deletion of phone book entries without the knowledge of user is an example of this threat.

Premeditated spyware: This category includes location tracking and remote listening.

Direct payoff: Sending sms without the permission of the user is a threat include in this category.

Information scavengers: Checking the address book, passwords and cookies without the permission of the user, lie down in this part.

Ad-ware: The malware advertisements on cell phones are included in this category.

III. Extending Android Permission Model and Enforcement with User defined runtime constraints

This paper presents a policy enforcement framework for Android that allows users to grant permissions to applications on the basis of their needs. And also to impose constrains on the usage of resources.

A. Problem description

Every application requires some permission which is mentioned in the AndroidMamifest.xml. These permissions are used for performing sensitive tasks like sms sending, using camera etc. At the time on installation Android asks user to grant permissions to the application to install. User does not have any other choice rather than granting all permissions to the application. Otherwise the application will not get install. Once the permissions are granted then user can not revoke those permissions until user uninstall the specific application.

Granting of permission to an application results in providing unrestricted access to the resources. Android existing framework does not provide a security check on the usage of resources. For example, if once sms permission is granted to an application. So, that application can start sending sms any time. There is no way to stop it, unless user does not grant all permissions to it.

Four issues: (1) User must grant all permissions to install any application; (2) No way for restricting the granted permissions to an application; (3) As all permissions are based on install time checks, access to resources cannot be restricted based on dynamic constraints and (4) The only way of revoking permissions are to uninstall the application.

B. Android Permission Extension Framework (APEX)

Different methods of ApplicationContext class in Android are used to handle the installation of application components. ApplicationContext acts as an interface for intents handling..The ApplicationContext implements the IActivityManager interface. It uses the concept of binders and parcels, the Inter Process

Communication for Android. Binder is the base class for remotable objects, that implements the IBinder interface and Parcel acts as generic buffer for inter-process messages which are passed with the help of IBinder.

The ApplicationContext creates a parcel with the help of IActivityManager, and decide that calling application has specific permissions. The ActivityManagerService class receives this parcel and extract the PID, UID and the permissions associated with it. After that, send it to the checkPermission method of ActivityManagerService class. Then these arguments are passed to checkComponentPermission, it perform some checks. If the UID is root or system UID then it grants all permissions. For the rest, it will call PackageManagerService which extracts the package name for the pass UID and validate the permissions. If received permission does not match any of those present in the GrantedPermission so, it throws a security exception.

After checking the present security permissions, control is given to AccessManager. For the purpose a hook is placed in the CheckUidpermission of PackageManagerService. As it is the only entry point for permission checking. It throws the UID and the requested permissions to the AccessManager. AccessManager invokes PolicyResolver, it retrieves attached to the related application and using the PolicyEvaluatinonEngine, evaluate it. The policy includes the condition on the basis of which permissions will be denied or granted. ExpressionParser retrieves the attribute of application from attribute repository and performs some sort of operations on these attributes.

C. Poly Android Installer

Writing policy is complex job for even system administrators. Android targeted at the consumer market and the end users as well. And users cannot complex usage policies. To end this problem the author created Poly. It is an advanced Android application installer. It provides user to specify constraints on the use of an application.

Allow: By default Android allows all permissions. This makes the existing Android installer a subset of Poly.

Deny: This approach opposed the current approach of Android, which is all-or-nothing. As this approach give facility to the user to deny any permission by his owns choice.

D. Runtime Constraint Modifications

One of the limitations of Android security mechanism is the lack of ability of revoking permissions after an application get installs. Uninstalling of an application is the only way to revoke the permissions.

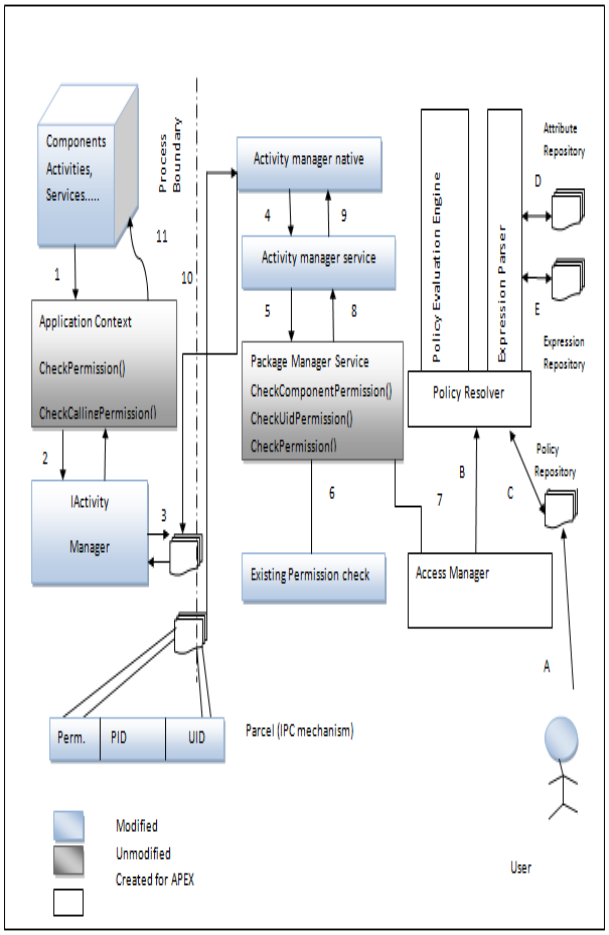

Figure 2: Android Permission Extension (Apex) Framework

Apex allows the user to specify his fine grained constraints at the time of installation through Poly. Once a user come to know that the application is not harmful, and he wants to assign more permissions to it, so Poly will help him in that. For example if a user install an application and grant it some permission and deny the permission of GPS. After some time he realize that this application not harmful and the user wishes to facilitate him with GPS facility as well. For modification the author created a shortcut to the constraint specification activity of Poly in the settings application of Android. This allows the user to modify constraints he specified at the time of installation. Even after, the application has been installed. And the same rule follows for denying of permissions after installation.

IV. Mitigating Android Software Misuse

In this paper we explained a framework; know as Kirin to capture security policy that transcends Android applications.

A. Contributions

The author reverse engineer Android's security model and present it formally. Author

provides a framework for specifying and enforcing stakeholder security policy. Prolog is used for install time installation. Prolog is a common language for security policy evaluation. Author use the proposed framework to identify insecure application policy configurations within Android. Such applications can affect voice, SMS and location services.

B. Kirin

In this paper a model is explained, which states that before system install any downloaded application package, it must first ensure the applications satisfy all security requirements. If any requirements fail to met, the installation will be terminates. This model, of installation ensures the cell phone will remain in its original secure state, without based on user made security decisions.

The policy pre-processor extracts policy from the target applications package, and converts it to Prolog facts. After that it merges it with the existing policy knowledge. The result of the merger represents the security state of the system, if the installation were to proceed. The policy engine after that uses the temporary policy state to evaluate invariants. Policy engine extract invariants from system policy, user policy and applications policy. On the basis of these invariants, policy engine take the decisions. It automatically generates compliance proofs for the target application. If all the invariants satisfied, so installation will continue. If any of the invariant fails to satisfy so the installation will abort.

C. System invariants

Invariant 1: "An application must have explicit permission to make an outgoing voice call."Android uses CALL_PHONE and CALL_PRIVILEGED permissions, to protect the API from making outgoing calls. This invariant makes it sure the no indirect access should be allowed.

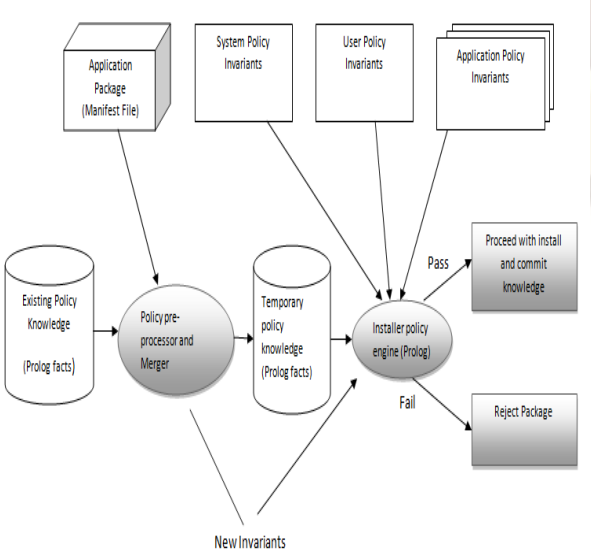

Figure 3: Enhanced Installation Logic. New packages cannot be installed unless all policy in variants passes

Invariant 2: "An application holding a dangerous permission must have no unprotected components" Android framework introduces "dangerous permissions". Which states that user should allow permissions to applications at time of installation. For example, sensitive tasks like making call and sending SMS permissions are mark as dangerous. So that, any application asks from user before using these services.

Invariant 3: "Only system applications can interface with hardware". Android framework introduces high level java APIs for interfacing with hard ware. For the sack of flexibility, Android let any application to interact with the APIs, but with proper permissions. This invariant insures only system applications have direct access to APIs.

D. User Privacy Invariants

Invariant 4: "Only system applications can process outgoing calls."Android framework let applications to receive notifications of outgoing calls, including the destination number. To keep the issue of privacy in eye, user may wish that only system applications should receive such notifications.

Invariant 5: "Applications that can perform audio record must not have network access or pass data to an application that has network access". It is quite dangerous for security, if an application record voice and send it on internet.

Invariant 6: "An application with access to Wifi or Network state must also declare network access."

E. Application invariants

Invariant 7: "An application can only receive SMS notifications from trusted system components." Any application has the ability to broadcast intent.

Invariant 8: "An application can only receive location updates from trusted system components." Some applications take decision on the base of location so, only the system applications have the right to send the location notification.

F. Limitations

Kirin is limited to obtain data from application package metadata. Kirin does not provide any dynamic security check. Kirin provides only install time security.

V. On Lightweight Mobile Phone Application Certification

The proposed Kirin security service for Android, which provides a lightweight certification of application at the time on installation. To certify applications based on security configuration requires to clearly specifying the unwanted properties. For the

identification of Kirin security rules, the author took help of security requirements engineering. On other hand a security language design has been defined, to implement Kirin security service within the Android framework.

- Methodology for adding new security requirements and flaws in current Android are defined.
- Practical method of performing lightweight certification of applications at install time is provided.
- Mitigation of malware rules is mentioned.

A. Kirin Security Rules

1) An application must not have the SET_DEBUG_APP permission label.

2) An application must not have PHONE_STATE, RECORD_AUDIO, and INTEREST permission model.

3) An Application not has PROCESS_OUTGOING_CALL, RECORD_AUDIO and INTERNET permission labels.

4) An application must not have ACCESS_FINE_LOCATION, INTERNET and RECEIVE_BOOT_COMPLETE permission labels.

5) An application must not have ACCESS_COARSE_LOCATION, INTERNET an RECEIVE_BOOT_COMPLETE permission labels.

6) An application must not have RECEIVE_SMS and WRITE_SMS permission labels.

7) An application must not have SEND_SMS and WRITE_SMS permission labels.

8) An application must not have INSTALL_SHORTOUT and UNISTALL_SHORTOUT permission labels.

9) An application must not have the SET_PREFERRED_APPLICATION permission label and receive Intents for the CALL action string.

B. Single Permission Security Rules

Dangerous permissions of Android may be too dangerous in some production environment. The SET_DEBUB_APP permission allows an application to turn the debugging for another application. The corresponding API is hidden in the most recent SDK. Third party does not have access to hidden APIs but it is not a substitute for security. Rule1 ensures third party applications do not have the SET_DEBUG_APP permission.

C. Multiple Permission Security Rules

Voice and location eavesdropping malware need permissions to record audio and access location information. But at same time legitimate applications also use these permissions. So a rule is need as multiple permissions. Rule 2 and 3 protect against the voice eavesdropper. Rule 4 and 5 protect from location tracker. Rule 6 protects from incoming malware SMS. Rule 6 and 7 consider malware interaction with messages. As SMS can be used as a path for malware. And malware owner will not let user Let know about SMS, therefore content provider will be is modified just after receiving a SMS. Rule 7 does not stop SMS sending, but increase the probability that user becomes aware of the activity. Rule8 makes use of the duality of permission labels. Rule 9 provides example of a rule considering both permission and an action string. This stops a malware from replacing the default voice call dialler application without the awareness of the user.

VI. Conclusion

In this survey paper three approaches are discussed for the security of Android. Kirin and Lightweight approaches are basically installing time approaches. If once an application grant some permissions, so there is no security mechanism through which Kirin or Lightweight keep check on the behaviour of application during runtime. Kirin cannot keep on check on dynamic broadcasts. Comparatively to Kirin and Lightweight, Apex approach seems to be more feasible. Which continuously check the application behavior at runtime, and on base of policy do not let an application to do something for which permission is not granted to it. For a larger user community, study of user requirements is required. Secondly it can create problems if user unknowingly grants such permissions to an application which can produce harmful results. This problem can be solved by the conjunction of Kirin with Apex, by analyzing the constraints and permissions to verify that security rules are not being violated.

References:

[1] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification USA, 2009. ACM.

[2] Google. Android Home Page, 2009. Available at: http://www.android.com.

[3] Google. Android Reference: Intent, 2009. http://developer.android.com/reference/android/content/Intent.html.

[4] Apex: extending Android permission model and enforcement with user-defined runtime constraints