MOSES: SUPPORTING AND IMPLEMENTING SAFETY ENVIRONMENT USING ANTI-THEFT CONCEPT ON SMARTPHONE

[1]Ms. Roshni M. Sherkar, [2]Prof. Ravi V. Mante, [3]Dr. Prashant N. Chatur
[1]MTech Scholar (CSE Dept.), [2]Asst. Professor (CSE Dept.), [3]Head of Dept. (CSE Dept.)
Email:[1]rinku.sherkar@gmail.com, [2]mante.ravi@gmail.com, [3]chatur.prashant@gmail.com

Abstract- As all we know that the usage of smartphone is increasing very rapidly because of the numerous services it provides to the end user. As a result of this many organization are willing to support customer-owned smartphone in order to increase the productivity of business user. Now, as it is providing so many services there is a necessity to provide security measures in order to protect data on smartphone. This manuscript provides policy based framework called as MOSES which allow user to create separate environment related to different context definition within the same device.

Index Terms—virtualization, security profile, isolation, taint values, context.

I. INTRODUCTION

Today's world is tightening with the progress of cell phone machinery. As the numbers of cell phone users are growing day by day, services are also increasing very rapidly. Beginning with old simple handsets which could be used for making calls, sending messages, cell phones have drastically changed user's life and became one of the most important parts of it. But now it has countless uses such as making calls, playing games, music, browsing internet and many more. And by means of these new emerging equipment's and its services, there is a requirement of new operating systems and software's.

A. What Is an Android?

Many operating systems have evolved in past 16 years. Beginning with black and white cell phone to latest smartphones or tablet, mobile operating system has become popular and come far off. Especially for smart phones, Mobile OS has greatly evolved from Palm OS in 1996 to Windows pocket PC in 2000 then to Blackberry OS and Android. Now-a-days most widely used mobile OS is Android which is nothing but a bunch of software consisting of not only operating system but also key and middleware applications.

B. Final Stage

Android is a most widely used and powerful Operating system which supports a countless applications that runs on smartphones. The usage of these applications has made life relaxed and innovative for smartphone users. It was established by Google. It permits users to choose and download the applications which are developed by third party vendor. Hardware component that supports Android OS are mainly founded on ARM architecture policy. Some of the current features and specifications of android are:

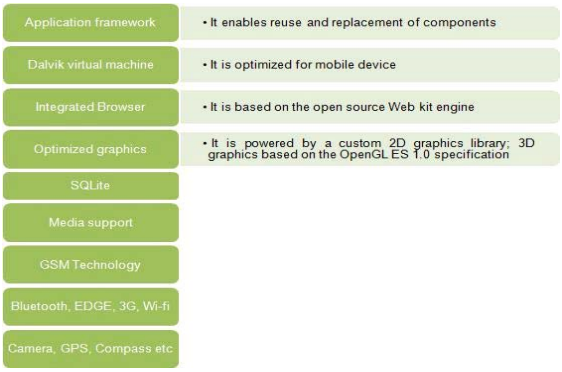

Fig. 1.1 Features and Specification of Android OS

C. Android Applications

Android is nothing but the extension on JAVA platform hence all Android applications are transcribed in java programming language. It is accessible as open source for inventors to invent applications that can be further used for selling in android market. Around 200000 applications were developed for Android OS with near about 3 billion+ downloads were made. Android bank on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model.

Following are the basics of Android applications:

• Android applications are self-possessed of one or more application components such as activities, services, content providers, broadcast receivers and intents.

• The role of each component is different as compared to overall applications behavior and each component can be activated individually.

• Another important part is the manifest file which must include all components in the application and should also include all application requirements, like as the minimum requirement of Android version and any hardware configurations.

• Non-code application resources such as images, strings, layout files, etc. should consist of replacements for dissimilar device.

In short, it is becoming a very effective and efficient tool for increasing productivity of business users. Many companies are willing to support customer owned smartphone because of increasing productivity of their customer and keep updated while they being on the move. Despite of this positive scenario, Android smartphones needs some security concerns. For example, malicious applications may access mails, SMS, and personal/private data stored on cell phone. Not only these malicious applications but an also legitimate application needs security concerns. Hence it is very important to provide internal and external security to it. The solution is given by policy-based framework called as MOSES in which this can be possible by separating data and apps related to corporate world from recreational apps and personal/private data. Within the same devices, separate environments are created which can run in their own address space. Data and apps related to first environment cannot access data from second environment. This can be possible with the help of virtualization. It is nothing but creating virtual versions of different instances of OS that can run separately on the same device. Two types of virtualization:

Para-virtualization is virtualization in which the guest operating system (the one being virtualized) is aware that it is a guest and accordingly has drivers that, instead of issuing hardware commands, simply issues commands directly to the host operating system. This will include things such as memory management as well.

Full Virtualization is virtualization in which the guest operating system is unaware that it is in a virtualized environment, and therefore hardware is virtualized by the host operating system so that the guest can issue commands to what it thinks is actual hardware, but really are just simulated hardware devices created by the host.

D. Android Architecture

Android OS consists of four main layers kernel, libraries, android runtime, applications framework and applications.

Linux kernel: Bottom layer is Linux kernel layer which provides basic functionality of system such as memory management, process management, device management as keypad, camera, display etc. The kernel manages all the things that Linux is thoroughly good at for example networking and a vast array of device drivers, which take out the burden of interfacing to peripheral hardware.

Libraries: On top of Linux kernel layer there is a bunch of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database is an important repository for storing and sharing of application data, libraries to record and play video and audio, SSL libraries are trustworthy for Internet security etc.

Android Runtime: This part provides a vital component called Dalvik Virtual Machine (DVM) which is a sort of Java Virtual Machine specially designed and optimized for Android OS.

The DVM rely on Linux core features such as memory management and multi-threading, which is inherent in the Java language. The DVM allows every Android application to run in its own address space, with its own instance of the DVM. The Android runtime also gives a set of core libraries which allows Android application developers to write Android applications using standard Java programming language.

Application Framework: The Application Framework layer is responsible for many higher-level services to applications in the form of Java classes. Application developers are relying on these services in their applications.

Applications: Android applications are at the top layer. You are able to write your application to be installed on this layer only. For example, Contact Books, Browser, Games, Media player etc.

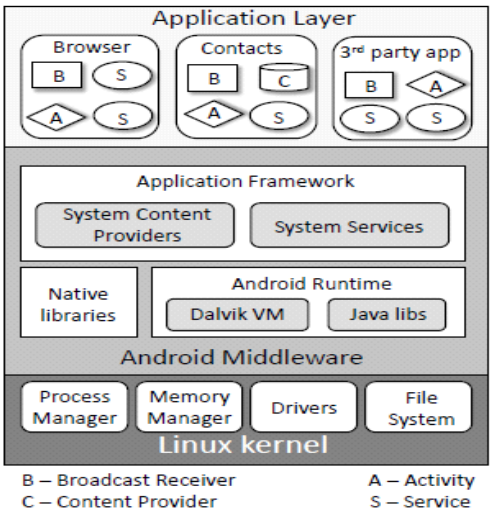

Fig. 1.2 Android Architecture

II. REVIEW OF LITERATURE

Countless solutions are there to plan to advance the safety of Android. We are going to consider only those that are correlated to our system. On Android platform, user has to grant all permissions at installation time which are requested in the manifest file. In short it supports an all-or-nothing approach, which is the user has to either grant all the permissions specified in the manifest or abort the installation of the application. Moreover, permission cannot be revoked at runtime.

In order to solve this problem several solutions have been proposed. This section provides gives review on related work which describes research efforts in order to increased security on Android platform. There are many solutions which help to improve security measures on Android. Some of them that related to our work are considered.

Basically, what happens on Android, at installation time users has to grant applications the permissions which are requested in the manifest file. Android endures an all-or-nothing approach that is the user has to either allow all the permissions specified in the manifest or abort the installation of the application. Moreover, permission cannot be reversed at runtime. In order to solve this problem several solutions have been proposed.

Apex [2] provides policy based framework for android that allows a user to selectively grant/restrict permissions to applications as well as user had several options on restricting the usage of resources i.e. user was allow using some functionality of application while restricting the access to resources which was critical/costly. It also described an extended package installer called as Poly that allowed the user to set these conditions through an easy interface. In this paper authors had incorporated only simple conditions such as restricting the time of usage and the time of the day on which to allow permission. This simplification was for user convenience.

In Secure Application INTeraction [3] aimed at run time, communication between applications were to security policies. Policies which were given in Saint had performed permissions checks by restricting access based on run-time state such as location, phone or network configuration, time, etc. Saint addressed the current security problems on Android. They

were at the beginning of integration of more applications and the policies.

CRePE [4] allowed a user to create policies that could automatically control the granting of permissions during runtime. Current smartphone systems permitted the user to use only contextual information to identify the working of the application. This makes difficult for the wide adoption of this technology to its full potential. This drawback was filled by proposing CRePE, a fine-grained Context-Related Policy Enforcement System for Android. While the concept of context-related access control is not an innovative idea. In particular, a context was defined by the state of variables imported by physical (low level) sensors, like time and location; additional processing on these data via software (high level) sensors; or particular interactions with the users or third parties. CRePE allows context-related policies to be set even at runtime by both the user and authorized third parties locally via an application or remotely via SMS, MMS, Bluetooth, and QR-code.

In MockDroid [5], is a modified version of the Android operating system which allowed a user to `mock' an application's access to a resource. In this system was able to limit the access of installed application by filtering out information they were accessing. This resource was subsequently reported as empty or unavailable whenever the application requests access. This approach allowed users to reverse access to particular resources at run-time, encouraging users to consider the trade-off between functionality and the disclosure of personal information whilst they use an application For instance, an application querying the IMEI number of phone may receive fake results even if the original is on the phone.

The goal of TISSA [6] was to prevent private information leakage by untrusted third-party smartphone application. Suppose any application requested to send a piece of data which may be private at that time it sent request to content provider. Instead of serving the request directly, it checked the current privacy setting for the application because user can change / readjust the permission for application at runtime. If reading was permitted then system returned normal result and if not then provide empty / anonymized / fake result.

Taintdroid [7] was responsible for tracking the flow of information between the applications. Taintdroid automatically taints the data from sensitive sources and applies taint on the data that were moving out through the smartphone over internet. When this happened, Taintdroid kept the record of the tainted data, which application sending the data, destination of tainted data. But the limitation of this is that it only kept the record of tainted data and was not able to provide security measure for sending data through unauthorized third party application. Also some legitimate applications are responsible for leaking out sensitive data which may cause harm to end-user.

This was solved by MOSES [1], in which separation between the safety environments (SE) was given depending upon different context. Each safety environment consists of its own apps and data and associated with its context. For example, application and data related to corporate world is separated from personal application and data. Safety environments are nothing but the set of protocols to limit the execution of application and data on particular environment. Here context is the most important term because; context provides security measures to smartphones. Context is a Boolean term which can be obtained from physical and logical sensors.

When the context returns true value SE associated with it is activated. It may happen that context is associated with one or more SE hence, to solve this each SE has assigned the priority. SE having highest priority will be activated first than the lower one. It may happen that two SE is having same priority in that case SE which had been activated will remain in processing. In MOSES [1] switching between the SE was possible by using virtual phone technique given in the Cells [8].

III. MOSES ARCHITECTURE

As given in [1], MOSES architecture consists of the components presented in Figure. 2. Main part of MOSES is the phenomenon of Context. The Context Detector System is responsible for

activating/deactivating of context. When it happens, the component Context Detector System notifies about this to the Security Profile Manager. The Security Profile Manager handle this information linking a SP with the Context. The component Security Profile Manager is used for the activation and deactivation of Security Profiles. The Security Profile Manager works as follow:

The Moses Hypervisor is the component that acts as a policy decision point (PDP) in MOSES. The Moses Hypervisor provides a central point for MOSES security checks against the policies defined for the active SP to regulate access to resources. The Moses Hypervisor delegates the policy checks to its two managers: the Moses App Manager and the Moses Rules Manager. The former is responsible for deciding which apps are allowed to be executed within a SP. The latter takes care of managing Special Rules.

The Moses Policy Manager acts as the policy administrator point (PAP) in MOSES. It provides the API for creating, updating and deleting MOSES policies. It also allows a user to define, modify, remove monitored Contexts and assign them to SPs. Moreover, this component also controls access to MOSES policy database (moses.db) allowing only applications with special permissions to interact with this component.

The Moses Taint Manager component manages the “shadow database” which stores the taint values used by Taintroid. In MOSES, we can taint specific rows of a content provider: to be able to perform per row filtering when an app access data in the content provider. For instance, it is possible to filter out from the query result data the rows which contain the information about device identifiers or user contacts. Given the fact that the enforcement of policies depends on the information provided by the Moses Taint Manager, this component acts as a policy information point (PIP).

The decisions taken by the Moses Hypervisor need to be enforced by the policy enforcement point (PEP). MOSES affects several components within Android middleware where decisions need to be enforced. For this reason, the PEP includes several Android components offering system services such as Location Manager and Activity Manager Service. Moreover, some Android core classes (such as the OS File System and OS Network System) are modified to enforce decisions regarding the access to the file system and network, respectively.

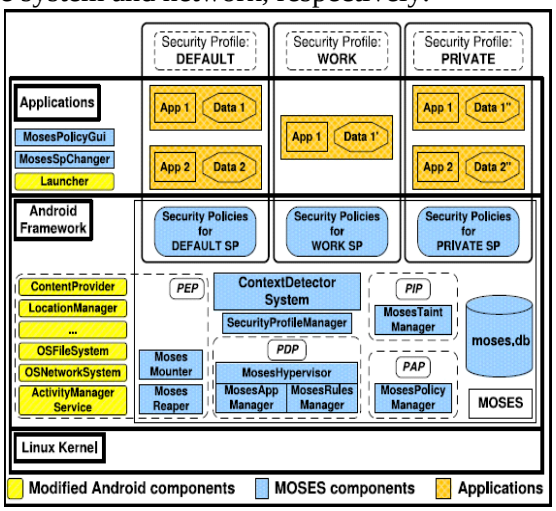

Fig. 3.1 MOSES Architecture

The enforcement of separated SPs requires special components to manage application processes and file system views. When a new SP is activated, it might deny the execution of some applications allowed in the previous profile. If these applications are running during the profile switch, then we need to stop their processes. The Moses Reaper is the component responsible for shutting down processes of applications no longer allowed in the new SP after the switch. In MOSES, applications have access to different data depending on the active profile. To separate data between profiles different file system view are supported. This functionality is provided by the Moses Mounter. To allow the user of the device to interact with MOSES, we provide two MOSES applications: the Moses Sp Changer and the Moses Policy Gui. The Moses Sp Changer allows the user to manually activate a SP. It communicates with the Moses Hypervisor and sends it a signal to switch to the profile required by the user. The Moses Policy Gui allows the user to manage SPs.

The most important thing comes into consideration is storage overhead. As it is explained earlier about MOSES, separation of data for different security profiles may be duplicated. So in general, storage size required by operating system can be given as,

$$s\_size = s\_size(OS) + size((executing\ apps)_i) + size((executing\ apps\ data)_i) \quad (1)$$

where, s_size(OS) is the total size required by the operating system of handset, size((executing apps)i) is the size required for executing the ith application and size((executing apps data)i) is the size required for storing ith application's data and j is the number of installed applications. But in specific case of MOSES, size of application's data is equal to [1]:

$$size\_MOSES((executing\ apps\ data)) = \sum_{k=1}^{n+1} \sum_{i=1}^{j} size((executingappsdata)_{ik}) \quad (2)$$

where, k is the security profile. As given in MOSES architecture, same application can be assign to two different profiles having different priorities. In this case, additional copies of information (n+1) is required to store initial data of application i.e. replication of data. For example, suppose Message app is assign in two profiles then updating present in one profile need to be replicated in another profile.

IV. DESCRIPTION

This section describes overview of implementation about the key terms of MOSES [1]. As we discuss earlier Moses provides abstraction for data and apps devoted to distinct contexts that are mounted in a solo device. For example, commercial data and apps can be separated from private data and apps within a solo device. This approach offers sections where data and apps are stored. MOSES guarantees that data and apps within a section are inaccessible from others sections' data and apps. These sections are nothing but the Security Profiles in MOSES. In general, a SP is a set of policies that regulates what applications can be executed and what data can be accessed.

As described in [1], one of the terms presented in MOSES is the automatic activation of SP depending on the context, in which the device is being used. SPs are associated with one or more definitions of Context. A context definition is a Boolean expression defined over any information that can be obtained from the smartphone's raw sensors (e.g., GPS sensor) and logical sensors. Logical sensors are functions which combine raw data from physical sensors to capture specific user behavior's (such as detecting whether the user is running). When a context definition evaluates to true, the SP associated with such a context is activated. It is a possible situation when several contexts, which are associated with different SPs, may be active at the same time. To resolve such conflicts, each SP is also assigned with a priority allowing MOSES to activate the SP with the highest priority. If SPs have the same priority, the SP, which has been activated first, will remain active. Also it allows user to switch between the profiles manually for this MOSES provides an application which forces MOSES to activate the required SP. Each SP is associated with the holder of the profile and can be secured with the password given by holder. Additionally, it supports remote SP management in which SP is secured with the password provided by organization this helps to control tampering with data.

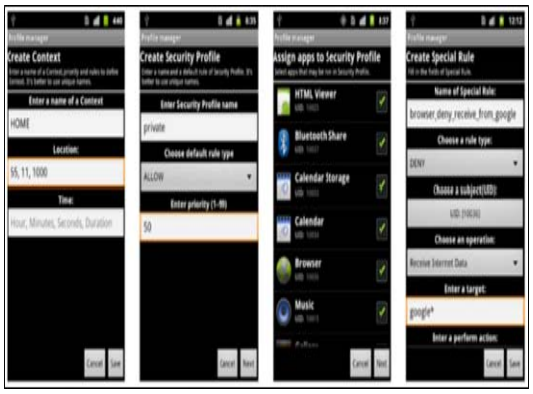

Fig. 4.1 Snapshots of MOSES application [1]: (a) Context creation, (b) Security profile creation, (c) Apps assignment to SP, (d) ABAC rule generation

Here, ABAC rule is attribute based access control [13], which enforces in each SP. The notion is that within each SP, users can define fine-grained access control policies to restrict application behavior. For example, the user might want to negate an application to read the files on an external storage. In this case, the user might write a policy which will still let the application to run within the profile but the access of this application to files on an external storage will be limited.

This is all about internal security which can

secure data present on smartphone. But, what about external threat? It may happen unauthorized user is trying to access data present on the smartphone or suppose it get stolen. In all circumstances, user is trying to secure his/her data. In order to provide safety, following algorithm is given which can protect data from external threat.

Step 1: User has to create or insert Security measures in terms of pattern or pin-password type format in order to provide a kind of security.

Step 2: Check whether the entered measures are correct or not. If "YES", then it will direct to MOSES application. Otherwise, it will direct to Step 3.

Step 3: System will allow user to enter it correctly two more times. If user fails again, then it will direct to Step 4.

Step 4: Check whether internet connection is available or not. If "YES", it will wipe all the data and send it to authorized email-id otherwise goes to Step 5:

Step 5: It will encrypt the data present on smartphone and whenever internet connection become available it will transfer it to authorized email-id. For encryption, AES algorithm is used. The Advanced Encryption Standard (AES) is an encryption algorithm for securing information in commercial transactions in the private sector. AES is a symmetric key encryption standard. The Advanced Encryption Standard consists of three block ciphers. They are: AES-128, AES-192, and AES-256. Each of the above standard ciphers is 128-bit block size with key sizes of 128, and 192 & 256 bits respectively.

In case of stealing, it may happen that unauthorized user is handling the cell phone. At that time, he/she will take out the SIM-CARD first so what the system will do, it will delete all the data from it. So that if in future, he/she accesses the phone he will get blank phone. But, it might be the case that authorized party wants to change the SIM-card then the system must provide the provision that user will be asked about new SIM-card insertion.

Again, daily back-up will be sent to authorize email-id.

This is all about how to secure data internally as well as externally. Because, in today's corporate world data is one of the most important thing in compare with mobile handset. So, this is the security mechanism which can be helpful to secure data.

REFERENCES

[1] Yury Zhauniarovich and et. al., "MOSES: Supporting and Enforcing Security Profiles on Smartphones," (Volume-11, Issue-3), May-June-14.

[2] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending Android Permission Model and Enforcement with User-Defined Runtime Constraints," Proc. Fifth ACM Symp. Information, Computer and Comm. Security (ASIACCS '10), pp. 328-332, 2010.

[3] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically Rich Application-Centric Security in Android," Proc. Ann. Computer Security Applications Conf. (ACSAC '09), pp. 73-82, 2009.

[4] M. Conti, B. Crispo, E. Fernandes, and Y. Zhauniarovich, "CRePE: A System for Enforcing Fine-Grained Context-Related Policies on Android," IEEE Trans. Information Forensics and Security, vol. 7, no. 5, pp. 1426-1438, Oct. 2012.

[5] A.R. Beresford, A. Rice, and N. Skehin, "MockDroid: Trading Privacy for Application Functionality on Smartphones," Proc. 12th Workshop Mobile Computing Systems and Applications (Hot Mobile '11), pp. 49-54, 2011.

[6] Y. Zhou, X. Zhang, X. Jiang, and V. Freeh, "Taming Information- Stealing Smartphone Applications (on Android)," Proc. Fourth Int'l Conf. Trust and Trustworthy Computing (TRUST '11), pp. 93- 107, 2011.

[7] W. Enck, P. Gilbert, B.-G. Chun, L.P. Cox, J. Jung, P. McDaniel, and A.N. Sheth, "Taintdroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," Proc. Ninth USENIX Conf. Operating Systems Design and Implementation (OSDI '10), pp. 1-6, 2010.

[8] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh, "Cells: A Virtual Mobile Smartphone Architecture," proc. 23rd ACM Symp. Operating Systems Principles (SOSP'11), pp. 173-187, 2011.

[9] Y. Xu, F. Bruns, E. Gonzalez, S. Traboulsi, K. Mott, and A. Bilgic, "Performance

Evaluation of Para-Virtualization on Modern Mobile Phone Platform," Proc. Int'l Conf. Computer, Electrical, and Systems Science and Eng. (ICCESSE '10), 2010.

[10]M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter, "L4Android: A Generic Operating System Framework for Secure Smartphones," Proc. First ACM Workshop Security and Privacy in Smartphones and Mobile Devices (SPSM '11), pp. 39-50, 2011.

[11]T.U. Dresden, and U. of Technology Berlin, "L4Android," http://l4android.org/ 2014

[12]C. Gibler, J. Crussell, J. Erickson, and H. Chen, "AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale," Proc. Fifth Int'l Conf. Trust and Trustworthy Computing (TRUST '12), pp. 291-307, 2012

[13]E. Yuan and J. Tong, "Attributed Based Access Control (ABAC) for Web Services," Proc. IEEE Int'l Conf. Web Services (ICWS '05), pp. 561-569, 2005.