POTENTIAL THREATS DUE TO APP PERMISSIONS ON ANDROID AND THEIR DEFENSE MEASURES

Ritu Rana
Indiana University Purdue University Indianapolis
Indianapolis,Indiana

***Abstract*—Smart phones are extensively used to increase the productivity of their users. Android amongst them is widely used platform but is also prone to various due to its open source behavior. Android users data can be exploited by the mischievous application and this kind of unauthorized access may result is data privacy breach. One such vulnerability of Android platform is its Access control mechanism. A user can only decide at run-time on what resources a particular app can access. After the app installation user loses the control on the app. Now user cannot validate app data access behavior. User also does not know what kind of data is being accessed. Also, the user cannot change an app's permission at run-time. Malware apps can utilize this open security flaw and can access user sensitive data. Number of defense techniques have been proposed in recent years to overcome these security flaws- such as dynamic access control, Automatic Risk assessment of the app, data flow analysis within an app. This paper discusses various dynamic access control model proposed recently and compare these techniques based on their effectiveness and performance.**

***Index Terms*—Context, Access Control, Kernel, Sandbox, Middleware**

I. Introduction

Smartphones have now become an inevitable part of human life. Now days, Telecommunication market is giving access to tons of users and the subscriber base is increasing very rapidly. Ericson research predicted that about 9.3 billion devices will be subscribed to mobile network around the world by 2019 and around 6 billion will be smart phones. Smart phones have various operating system such as IOS, Android, and Windows. [28] Amongst them, the Android platform has emerged to be the most popular smartphone operating system, holding more than 75 percent market share. Undeniably the Android dominates the mobile device market. But with the increasing demand of phones, the computational capability, complexity and data storage have also increased in recent years. Along with that, smartphones also allow users to sync their data and provide access to their data while on move. Also, many companies are now providing mobile version of their desktop app. More users are migrating to the use of mobile applications such as for gaming, banking, video calling, voice recording etc. This has led to potential threat to the users data stored on the device. In Android, when the user installs an app, it explicitly requests user a set of permissions. Android operating system informs the user upfront about an app access rights before an installation. By this step Android operating system expect that user can make knowledgeable choice.

The productiveness of such a defense scheme largely depends on choices made by the users [23]. Once user grants access to apps, the apps take the control from user. Now user cannot validate app accesses and verify that the apps is not working outside the given access rights. User also does not know how the sensitive resources have utilized by the apps[13]. Malware apps are a clear example of unwanted and unauthorized behavior. This kind of apps can compromise the user data security and privacy. Users also carry their smartphones everywhere and access public and private internet networks. Unwanted and unauthorized app access can expose their secure and confidential information without user knowledge of the presence of such malevolent actions on their devices.

Many security measures have been proposed to secure operators data and privacy. Variety of techniques have been proposed which identifies sensitive users input and potential information leaks either statistically [14] [15] or dynamically [20] [18]. Several Access control mechanism have also been proposed [16] [17][19] to enforce fine grained security policies on the app such that users private data can be handled on a mobile system. AUTOREB (Deguang et. al. 2015) uses machine learning techniques to assess the risk of an app both at review-level and app-level based on the reviews provided by the previous users on the app store which reflect the security-related behaviors of the given app.RISKMON (Yiming et al. 2015) is an incessant automated risk assessment framework. This framework evaluates the mobile application risks. Through machine learning technique it creates a risk assessment baseline. This baseline tracks appropriate behaviors of the apps.

Techniques [23] focus on effectively communicating the security threats during the installation of an app to the user to assist them understand and make a good decision on whether or not to install a given app. However, there is still a need for the system which could help a user to execute the restrictions he defines. The main contribution of the paper include the following:

- Discuss the Androids current access control mechanism and provide its drawbacks.

- Discuss the adversary model and security extensions that allows user to define and enforce their own restriction policies on each app.
- Provides a comparative evaluation of the discussed techniques based on certain identified criteria such as type of technique, where it is implemented and their efficiency and overhead.

The rest of the paper is organized as follows: Section 2 recalls some preparatory for the Android security, while the security concerns are discussed in Section 3. Section 4 discusses the system and solution requirements based on adversary model. The proposed solutions are discussed in Section 5. While Sections 6, 7 and 8 provides the comparison and discussions of the techniques.

II. ANDROID ACCESS CONTROL MECHANISM

Android is an open source mobile operating system. Android OS is Linux kernel based. It was developed by the Open Handset Alliance (OHA) and released under the Apache license [28].Java programming is the primary language to develop Android applications. These Java applications are compiled into custom byte code and these byte codes run on DVM (Dalvik virtual Machine).Each package is executed in the address space of its own and in a unconnected DVM [2].Android app programming is divided into four components that work together to create every functional application. All the components in the OS interact with the other components and works together in order to obtain a functional application.

- Activities : This represent any single screen with GUI.
- Services : Services handle long-running operations and does not have a user interface.
- Content Provider: It manages a shared set of data across apps.

Data can be stored on different locations or at one location. It can be stored in file system of the device, SQL database, Web or other data location. Broadcast Receiver respond to system-wide announcements [28]. The existing Android security model contains two layers:-

- Application level (Android permissions): The app developers can directly access this layer. This layer gives them direct access. This layer requires set of required permission from app by which user can provide its authentication.

  The developers have to provide application required permissions to the end user as their part of the application manifest file. The user is required to grant the permissions before installing an application.
- Kernel level (Linux system level): Kernel level model resides underneath the uservisible android permission model, which provides the foundational mechanism for application isolation and sandboxing. Normally this model operates without being visible to app developers and users so that an app does not attempt o violate the restrictions imposed by the kernel [16] [5]. Android uses Linux discretionary access control (DAC) to implement these controls.

A. *Android Discretionary Access Control (DAC)*

Android employs DAC in two ways:First, It uses DAC to block the system resources usage from the apps. If there is a requirement for the app to use system resources then DAC provide indirect access to the system resources. DAC can also authorize an app to access system resources directly. Secondly, DAC is used to separate one app from another [28]. All android files which are stored in memory comes under the Linux access control. When an app gets installed by the user it is provided with a unique Linux user identifier (UID) and a group identifier (GID. Every app gets executed as a dissimilar user procedure contained in its own file within its own isolated space location. The file access permissions in Linux are divided into three kinds of user's set. For each type of user read, write and executed (r-w-x) permissions are assigned. All file under user home directory can be read, written and executed by the owner and users with proper role. Any user outside of its own group is disallowed to work with these files. A separate application with a different identifier cannot access other application by other identifier.

However with the use of DAC characteristics to modify the Linux kernel some of the problems related to DAC model has been lessened by Android. Since it represents the applications directly and not users. Then too, some of the drawbacks of the existing system remain which include the capability of the malevolent apps to leak the data access to the outside world, the granularity of the current DAC permissions which tend to be coarse. Along with that, the DAC has no measures to restrict the apps or programs that use the super user identity [31].

The existing use of SELinux targets the protection of components in the android system and its genuine apps from the manipulations done by any third party app installed by the user. As mentioned in [32] all the third party apps are classified in an untrusted app domain. These apps can only obtain access or protection from other apps using Android permissions and Linux DAC support. This serves as a substantial drawback as a malicious app can get the benefits merely by their own policy description.

B. *Android Mandatory Access Control (MAC)*

In the MAC, The application components control is provided through inter-component communication (ICC) reference monitor [5].In to discretionary access control, a component cannot pass the permission to other component. Protected features like Application components and system services (e.g. Bluetooth) are designated with a unique security labels permission. [5] [2].So in order to use these protected features and to access these protected components an app developer declare the required permissions in its package's main file (AndroidManifest.xml.)As an example, consider an app that needs to reads the users data, the AndroidManifest. xml included in the apps package would specify:uses-permission android:name=android. permission READ-OWNER-DATA. At the app installation time, these permissions access are given to an app and these access rights cannot be modified later.

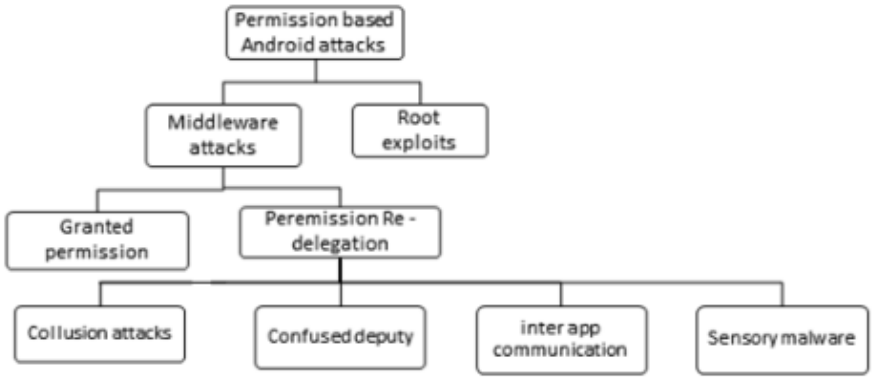

Fig. 1. Classification of Permission Based Attack Strategies

According to Android official document, the attribute protectionLevel is used to characterize the potential risk of permission. All the permissions are distinguished from each other using four categories:

- Normal permissions: In these kind of permissions, access rights are automatically given by Android.
- Dangerous permissions: In these kind of permissions, user manually provides the access rights.
- Signature permissions: In this kind of permissions a signature request gets generated by an app and it must be signed with the identical key as app which is declaring the permission.
- Signature or System permission:In this kind of permission, a package which has the system key gets permission [2] [5].

In dangerous permission request, the user can opt for only two actions. Either he can Agree to the permission request or Refuse the permission request. If user opt second option then the user cannot install the application. As mentioned in [5]. Android has no mechanism to modify the permission.

C. *Sandboxing*

This is also known as application containerization and it is used in software development and mobile application management (MAM) This kind of development setup restrict the environment Where few apps can execute. Android OS for application sandboxing uses Linux discretionary access control (DAC) mechanism. This Sandboxing improves the security. Through sandboxing app gets isolated from other apps and this mechanism improves the security and prevent from malware, intruders, system resources and other applications to interact with protected app. Linux user IDs are assigned to each installed app which is unique to each currently-installed apps on a particular device and also ensues privilege separation among them. Each application process is executed in its own UID and it provides access to low level resources such as app-private files and low-level IPC. Linux DAC also controls domain sockets.

D. *Intent*

Intents are asynchronous messages and information of an operation to be performed. It allows the three core application components such as Activities using startActivity, startService(Intent) or bindService(Intent, Service- Connection) to communicate with a background Services, and broadcastIntent to communicate with Broadcast Receivers and request functionality from each other [1]. The components communicate between each other inside the application's personal process or across external applications. Intent usually a activity which gets executed in the starting and it also creates a easy access for late runtime code and apps bindings.It generally carries two piece of information:

- Action: The action such as ACTION- VIEW, ACTION-EDIT, ACTION-MAIN; needs to be executed.
- Data: The data on which the action needs to be performed such as editing a contact information from the contacts database.

Intent provides information for the actions to transfer data among apps, request services,and request to an application's action [30].

E. *Android Permission Groups*

Permission group contains the collection of information of related premissions. Android collects these related permissions logically. PermissionGroup attribute of the permissions element lets individual permission join the group. Members belonging to a particular group are displayed together in the user interface. This element declares a category in which permission can be placed. For instance, SOCIAL-INFO collects the permissions information and theses permission information have a users contacts and call logs information. Most groups permissions are so clear and self-explanation able that do not require special messages like Location and Camera.Permission group also contains attributes such as User-readable text that describes the group and group name

III. Android Security Issues

The current permission based model (Android mechanism) encounter some problems.The user does not know the behavior of the application based on the app required permission. It is really hard of user to understand it is genuine app or malicious app. This is because to the relationship between a permission and its associate methods. Second, permission mechanism is static, users cannot edit or modify the permissions of an application .The permission are given from the application developer at run time. The attacks can be divided into two categories: Attacks on Android Middleware and Root exploits.

A. *Android Middleware Attacks*

Following are the two kinds of attacks that are possible due to the access privileges given to mobile application[8]:

Granted permission to malicious application : Malevolent applications often provide wrong permission information to the user and by this they get the access from the user. Many malevolent apps request to access the SMS permission. Malicious apps use this kind of permission to send SMS to the premium numbers. Access to INTERNET permission is also frequently requested by the apps and the developer of malware uses this permission to collect sensitive information such as user's contacts numbers,access to gallery, etc [8].

Permission Re-Delegation (PRD) Attack: In this kind of attacks (PRD), a malicious application requests a permission from a different application. This attack is so advanced that it is harder to find. For example, an application did not request for any dangerous permission during the time of install and hence its direct request to send SMS would be rejected by the system. This infected app sends an intent to another app and the second app has the SMS functionality access. The second app happens to acknowledge the intents. Hence, an SMS would be successfully sent if the callee app does not check for the permissions of the caller app and accepting any intent it receives. Along with that, callee can also notify the caller if the request has been fulfilled. Privilege escalation attack at an application level is an example of PRD attack. A Privilege escalation attack can be defined as an application with fewer permissions (a non- privileged caller) can access components of a more privileged application (a privileged callee) [17]. Some of the types of Privilege escalation attacks are as follows:

- Collusion attacks allows apps to indirectly execute operations, which they should not be able to execute, based on their declared permissions. For instance, disclosure of users private data (e.g., contacts or GPS location) to a destined attacker, by apps that do not have direct access to such data or cannot directly establish remote connections [17].
- In the confused deputy attacks infected application exploits the vulnerable interfaces of the another confused application.
- Code or Data cooperation between applications, In code cooperation a Low level access application calls the modules of the higher level application while in data cooperation low level application access the sensitive's information from the higher level app.
- Sensory malware allows the information from device sensors to obtain privacy sensitive information, like user input.

Category	Behavior Description
Over-privileged Permission	App requests too many permissions
Data-Leakage	App accesses user's data without his knowledge
Inter-app communication	Under-privileged app access data by sending intent to privileged app
Sensor malware	Device sensor collects user's privacy sensitive information

TABLE I
SECURITY AND PRIVACY RELATED BEHAVIOR

B. *Root Exploits*

Rooting allows certain apps to bypass security restrictions exposed by android OS. [23] [24]Android certain apps to function properly and execute their proper work depends on root privilege. These kind of apps control the actions and avoid the builtin security sandbox without user consent and giving any kind of information. For example, an app can obtain a privilege to directly access contacts database file directly without bypassing the Contacts Provider permission.

Table 1 provides a brief description of the type of attacks and their behavior description.

IV. Requirement analysis for security extensions

The section discusses about the types of requirements needed for security extensions to be implemented on Android.

A. *Types of Requested Permissions*

(Huikang et. Al 2015)conducted an analysis on a group of diverse datasets. The dataset contained 7737 apps. The applications data set were collected from Android Store.They collected 1260 malware samples from Android Malware Gene Project and 7099 safe application. The apps were divided into 29 sub categories based on their type such as entertainment, gaming, education, reading etc. From sub categories, it has surveyed the INTERNET was the most requested permission i.e. 27 among all the 29 categories. There are few top request from every category. They are:- ACCESS NETWORK STATE, WRITE EXTERNAL STORAGE, READ PHONE STATE, ACCESS WI-FI STATE.

While permissions such as INTERNET, ACCESS NETWORK STATE, and ACCESS WI-FI STATE, WRITE EXTERNAL STORAGE and READ PHONE STATE are considered as primary requests by an app and are required to achieve

functionality. More than 70 percent apps in every category requested these permissions. (Bhaskar et al 2012) has classified 26 permissions as critical (which contained critical information of security and privacy of end users) out of 122 Androids current list of permissions. Besides INTERNET, ACCESS COARSE LOCATION and ACCESS FINE LOCATION are also widely requested by both malware and benign apps. (Felt et al. 2011) After analyzing 100 paid and 856 free Android Apps obtained the result which states that almost all applications (93 percent of free) and (82 percent of paid) request for atleast a single risky permission. Their analysis concluded that users are familiar with the risky permissions asked by an application during the time of installation. (Motiee et al. 2010) Based on the conducted survey on a group of participants,it was observed that 69 percent of the participants neglected the UAC dialog and directly began to work on the administrator's account. Microsoft confessed that participants answers yes to nearly 90 percent of the prompts, this indicates that users usually respond to such questions due to their habit while it is necessary that users focus on the important prompts to making a wise choice [24] [25] [26].

B. *Requirements for security extensions*

Mostly third-party apps are not necessarily reliable but some of their features are desired by the user. The user who is acquainted with his privacy choice wishes to install and use the app on his device. By this user allows access to his personal data and perform risky operations. Based on the adversary model the necessary requirements for flexible and access control architecture on Mobile devices on Android OS [1] [4].

Access Control on Multiple Layers: Kernel Level MAC provides inadequate protection against security weaknesses. The system middleware and application layers are also vulnerable by the security flaws. This type of control lacks High level semantics to enable a fine-grained filtering at those layers [4]. Access control solutions at middleware level should be able to address these flaws of kernel level MAC. It should be able to provide simultaneous MAC defenses at the two layers. The two layers can be dynamically synchronized at run-time over mutual interfaces.

Multiple stakeholders policies: Since Mobile devices involve multiple stakeholders, such as the end-user, the device manufacturer, app developers, or other 3rd parties (e.g., the end-users employer). These stakeholders also store their sensitive data on the device. Data belonging to different stakeholders are subject to different security requirements and might not be always aligned. Thus the solution should support handling the policies of multiple stakeholders if needed.

Context-awareness: As users or stakeholders access permission might depend on the current context of the device, the policy restrictions should be applied automatically as the devices location changes. The application should be able to change/ fake the device location and time. It should not be able to escape the policy restrictions applied by the user on the device in a specific context

Device Performance: The solution should not cause noticeable delays in the functionality of the device which could lead to remarkable delays in the performance of the system .

Source code security: The solution should not allow the developers of an application to alter the Androids source code or apply any additional prerequisite on their apps. Also, underprivileged should not be allowed to obtain resource access by sending intent to other app.

C. *Defense Strategies*

In order to address the limitation of static access right on android,lot of research has been done. Researchers have come up with a number of solutions which can provide in depth control of Android apps. The solutions fall into two categories: First, the category of solutions are those which enlarges the Android and the kernel framework to allow fine-tuning of the apps permissions. But according to [6][27]. Due to the large numbers and vendors, the need to enhance main components of the Android could lead to fragile practical distribution. Also, vendor customization could impact overall Android security. The second proposed approach provides In-App mechanisms to control app access to sensitive APIS by either Dalvik Bytecode rewriting or native code interposing [6][27].

These solutions are easy in deployment because their process is easy and can work easily with Android framework as they do not require any changes to the Android framework. Bytecode rewriting solution have an inline mechanism to check and monitor the apps behaviors. On the other hand, dynamically loaded classes could also pose serious challenges to these systems as the bytecode which is loaded dynamically is unavailable when the app is rewritten statistically.

V. Proposed Solutions

This section briefly discusses some of the recent proposed solutions of each type as discussed in Table 2 to provide dynamic access control to Android apps.

Context-based Access Control (CBAC) was proposed by (Bilal et al.2015) is a mechanism where user has the rights to set up app configuration policies based on their behavior and device resource usages.Policies set by the user works automatically, when user change the device location and new

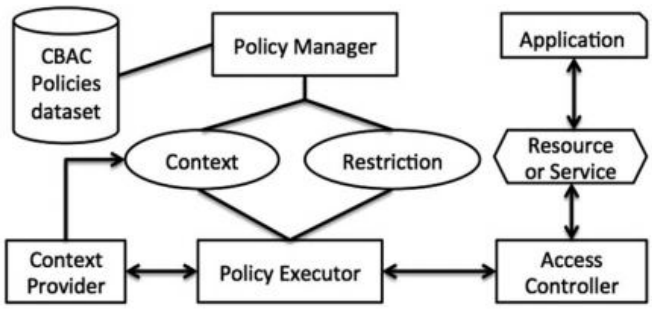

Fig. 2. Security Framework of Context Based Access Control System [1]

Type of Security Enhancement	Key Points	Strengths	Weakness
Application Level	• Apps that are developed at Application level and available at Google store. • Eg. APP Lock, Gallery Private, Free Data Vault etc	• Safe • Ease of access • Free of cost	• Unlocking a file is time consuming • User needs to remember his password to unlock files • No privacy advisor, web protection
Identify apps permission requests	• A application is analyzed based on the set of permissions it requests	• Easy to implement • Simple	• Might not be very accurate at identifying real issues • Does not take into account the problem of colluding applications performing privilege escalation attacks
Framework extension	• Solutions that extend the Android framework level or kernel level by enforcing fine-grained policies, • Separating the runtime environment with light virtualization • and implementing full-virtualization by hypervisor	• Does not disturb a regular user in any noticeable way. • Can be used at Run time.	• Practical deployment could be impaired by Android components due to the deep fragmentation of the Android platform. • Need extensive changes to Android OS • Effective only within the Android middleware • Require maintaining a customized version of Android for each hardware configuration • Can be bypassed by malware in the wild
In-app mechanisms	• Control the apps access to sensitive APIs through (Dalvik) bytecode rewriting or native code interposing • Third-party apps are modified to inject security hooks • When the application package is downloaded, extra-security checks are added to the application code generating a hardened version.	• Does not change Android OS and can be readily deployed. • The hardening (versions with extra security extensions) tools can be easily deployed as standard applications on the device.	• Dynamically loaded bytecode might not be available when statically rewriting the app. • Methods containing dynamic classes could be more complex and tedious • Rely on the user to install the hardened version of an application • Data associated with the previous version is lost after installing the hardened version. • Can only control the bytecode of an application

TABLE II
SECURITY AND PRIVACY RELATED BEHAVIOR

location matches with predefined context of a user defined policy, all predefined policies gets applied to the apps. Context (Location) is determined using Context Provider (CP) which collects the physical location parameters using cellular triangulation coordinates whenever available, Wi-Fi parameters, GPS and device sensors and stores them in its own database. It links the collected locations to user defined logical locations and also verifies and updates them when the device is relocated. The Access Controller (AC) is a fine-grained control permissions system, built on exiting OS access controller and is responsible to control the authorizations of applications and prevents any unauthorized use of device resources or services. In CBAC device restrictions are applied with the help of Policy Executor (PE) by comparing the devices context with the configured policies. users can also define sub areas within the same location. It is obtained by extending the Androids broadcastReceiver class. In CBAC policy is defined as [LoC, ST, ET, R]. Where LoC is the location of the device St and ET are the starting and ending time of the restriction and R is the type of the restriction policy that is applied on the particular app. The user creates his policies by using PolicyManager Class which serves as user interface while the policies are stored in Policy Manager Data directory.

CBAC also provides a solution to avoid inter app communication. It uses Intent manager to update the Activity class methods. Activity class contains startActivity(), startActivityForResult() and the ContextWrapper class and these classes have SendBroadcast() and startService() methods to intercept the intents. These classes and methods also control the actions performed on various Intent objects.

MOSES (MOde-of-uses SEparation in Smartphones) by (Yuri et. al. 2014):- It is a policy based framework. This framework rules enforce the Software application and data isolation on the Android platform.It creates separate Security Profiles within a single smartphone.By the MosesPolicyGUI users can dynamically create his policies and policies contexts. It separates data and different contexts apps. While Context changes are determined through Logical sensors and physical sensors. Data and app related to different contexts are stored in different compartments (called Security Profiles, SP). Same compartment data and apps are isolated form other compartments data and apps and cannot interact by any way. User defined set of SP states that what application can be execute and what data can be used inside the specified context. Based

on the profile context, Security Profile Manager automatically activate and deactivates the SPs.

Whereas, user can also manually activate an SP using MosesSpChanger which communicates with the MosesHypervisor (the component which serves as a policy decision point (PDP)) and sends a signal to change to a different profile needed by the user. MoSES also preserves information related to context and its corresponding SPs in a runtime map in the form of (Ci,(SPk; prtk)i), where Ci is the identifier of Context and (SPk; prtk)i is a tuple, which corresponds to the Context Ci and consists of SP identifier SPk and the priority prtk that corresponds to this profile. Hence, policies applied to one application will be automatically extended to the other apps with same UiD. Along with that, some fine-grained policies are built on top of Taintdroid [11] functionality and so MOSES also innates the drawbacks of Taintdroid. Also, applications with root access to the system can dodge MOSES protection. MOSES was tested on a Google Nexus S phone and was executed on three types of systems: Stock Android, MOSES with and without SP changes[2]. Results showed that MOSES incur minimal energy overhead in case of both running and switching between contexts.

(Daniele et. Al. 2014)Provided two solutions to address the problem:

- Data Shadowing
- Dynamic Permission Management

Both solutions deliver the solution to the problem by application-level anonymity.Data shadowing hides user sensitive data protect user identity which can be accessed by the data. It shadows by obscuring data which contains user identity from apps and supplies a placeholder for the user's sensitive data which does not represent the user or his device. Dynamic Permission Management revokes Android's apps permissions dynamically associated with sensitive information at run-time. Data shadowing and Dynamic permission management both have two additional features.

- Fresh Start: Fresh start blocks apps to leave any revealing data in their own data storages.
- Intent Filtering: Intent Filtering blocks inter-app communication at run time while anonymous session take place.

Same Android system mechanism is used to enforce the security policies and application UiDs is used to block all package names associated with a UiD. IdentiDroid is a customized Android operating system which was developed to implement these solutions. IdentiDroid Profile Manager is a profile-based configuration tool which allows user to set his own configuration policies for every installed Android app. These profiles can be configured by the user to use either one of the ambiguous solutions or can also be customized based on the application. IdentiDroid was implemented on the Android Google Nexus 7 tablet and the Google Nexus 4 phone to correlate the anonymity behavior of the device after implementing IdentiDroid. The solution was analyzed on 250 Android apps to determine what information, services, and permissions can identify users and devices. Experimental results demonstrated the usefulness of these solutions on user's private data since neither the user of the device were identified outside without any remarkable impact on most of the device applications.

FlaskDroid (proposed by Sven et. al 2013)is a generic and practical access control architecture for Android. It works on Kernel and middleware both. From enforcement level, it decouples the policy management and decision making and combines both. It stipulates adequate and highly flexible access control over middleware and Kernel level operation both. It is inspired by Flask security architecture. Stock SE android is a building block at Kernel level to harden Linux Kernel. It set the policy enforcement at the middleware level. Based on the current system date , the Dynamic policy support of SELinux allows to reconfigure the access control at runtime. Userspace Security Server control these policies and works as the central policy decision point for all Userspace access control decision. Kernelspace security server gives all Kernelspace policy decisions. Uninstallation of application packages for a preferred component at runtime is governed by PacakaManagmentService. ActivityManagerservice manages the stack of Activities of different apps, Activity life-cycle management, as well as provides the Intent broadcast system. While apps share their data through content provider. Context manager manages the device context and is responsible for activating or deactivating specific policy related to certain context. FlaskDroid was prototyped on Android 4.0.4 and evaluation shows that FlaskDroid is the effective and efficient and benefits the API-oriented design of Android.

CREPE by (Mauro et. Al 2012) is a dynamic pulverized Context-related Policy enforcement security extension for Android. Users and authorized third parties both are allowed to set context related policies (Even at runtime) locally or remotely (Via SMS, Bluetooth, and Or-code). It supports physical context(Location, time and online) associated to physical sensors(GPS, Clock, Bluetooth Etc.) and logical contexts defined by functions over physical sensors (sensors which distinguish if the user is running in an open space or is still). "CREPE supports dynamic policy management, thus

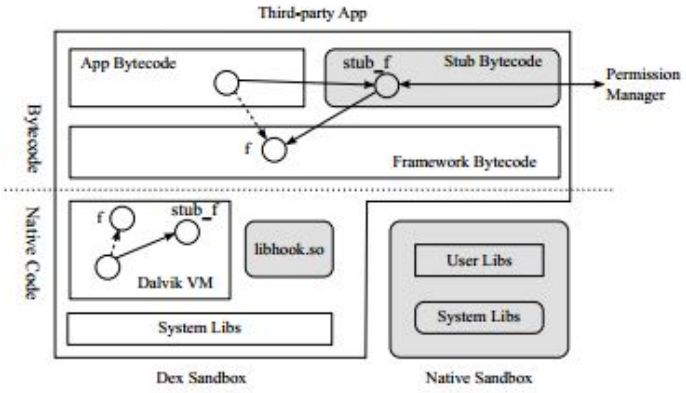

Fig. 3. System Architecture of AppCage [6]

at any time the administrator can set, delete and modify new contexts/policies at runtime". There are two entry points for CREPE policies. The Local-Administrator which serves as GuI for user to set his policies and RemoteAdministrator which allows authorized third parties to manage CREPE remotely. The behavior of an app is specified by the set of context C and policies P associated with the context as (C,P). Crepe uses standard Android mechanisms to avoid inter component communication and is installed with the certificate of a root certification authority (CA) to prevent unwanted access from third party. Experimental results showed that crepe increases system security with negligible system overhead in terms of energy consumption, time, and storage.

AppCage proposed by (Yajin et. al. 2015) App cage is a secure app mechanism. This mechanism regulate the apps access to delicate APIs and unsafe operations (e.g., dialing calls). AppCage uses two sandboxes (dex and native). By this mechanism it grants apps to interact with key Android APIs and impose rules defined by the users. It combines a packaging app (original app and components) to the sandboxes for every concerning app. This packaging app sets the dex sandbox. This Packaging app build a connection with Dalvik VM instance and concerning app. App redirect sensitive framework APIs to their respective stubs. The remains place the access to those APIs and enforce the user rules. The native sandbox restrain the native sandbox by supporting software fault isolation (SFI). This SFI process warrant that the apps inherent libraries cannot escape from sandbox. This confirms that the any app cannot mutate dex sandboxing by their inherent libraries. It also prevents the apps native code from requesting the sensitive data directly or performing dangerous operations via binder. Unlike in-app mechanisms, the main advantage of AppCage is that it does not need to modify the Android framework; nor it does requires the root privilege. It can be freely deployed With API hooking. It also supports dynamically loaded classes. AppCage was tested on a Google Nexus S phone with Android 4.1.2. Results demonstrated that it successfully eradicated the attempts to disclose user's private data or perform any risky operations by malevolent apps with acceptable system overhead.

ChainDroid proposed by (Qihoui et.al.2013) is a light-weight permission management system which is used to protect against Confused Deputy attack through inter-application communication in Android. It does this is two steps. First, it constructs a call chain (using Android activities). This call chain procedure collects components inter communication activities among components and stops unauthorized application access. Second, it creates an access policy file (.XML file) for each system API based on call chains. By this steps it limits certain attributes for example Numbers of application in call chain . Distinctive call chains generated based on invoked System APIs while access policies are created based on the call chain. Decisions are made based on the parsing of access policies and analysis of call chains. Based on call chain examination and parsing of access policies ,collected results create conclusions. ChainDroid also allows users to set a constraint attribute for the target system API and the API is protected by both call chains and access policies. ChainDroid was tested on phone calling API of the Android and the performance evaluation experiments on Samsung S5PV210 with Android 2.3 (Gingerbread). Two applications were developed Contacts (with permissions of phone calling and accessing contacts provider) and CityWeather (without any access privileges it can start components in Contacts to make a phone call or approach contacts data). Results showed that ChainDroid was successful in detecting Confused Deputy attacks and have a negligible effect on Android system's performance.

Dynamic Role Based Access Control Android (DR BACA) by (Felix et.al. 2013) is a security model which extends the current android permission model. DR BACA is closely related to CRePE [2]. CRePE emphases keep a tight rein on the non-malicious application capabilities based on the current context information. For example, the phones location (GPS) and the device time. DR BACA offers same functionality but it uses different mechanism. This is a role based access control mechanism which provides a better company environment map. This component attaches with android framework and allows DR BACA system to make allow or deny decisions for both permission and application execution request. The key difference between an execution request and permission request is that the permission request first consults the android reference monitor and an execution request invokes DR BACAs permission and execution checker. If a permission request gets rejected by Android DR BACA does not consult and immediately reject the request. DR BACA by user manager provides multi-user capabilities .This allows different users to log into the same mobile. Based on the security policy, users have different policy permission. It warrant that at the different user login no security sensitive information remains running. DRBACA allows single user to login into different devices until user belongs the same security policy. In DRBACA, The context monitor component provides role based access control. The rule sharing agent in Android application increase the capabilities of DR BACA access controls and NFC (Near field communication) technology is the key factor for Dynamic RBAC capabilities. NFC allows for radio communication between NFC enabled devices. Applying RBAC model with current android permission model provides advantages such as to use context-aware capabilities of mobile model and to utilize available NFC technology. Utilizing these advantages ,DR BACA gives elegant access control to Static and dynamic RBAC both on mobile devices. It also provide multi-user capabilities and these capabilities are used and ideal for enterprises. By these capabilities enterprises implement a universal security policy on the devices.

MacDroid Macdroid uses pre- defined rules and convert into its own policies. Macdroid policies bundled into different groups and each group policies are specific policies to share Android OS resources. MacDroid accomplishes this by employing a hooking mechanism to enforce establishment of its policies. This Hooking mechanism is used to change or supplement the behavior of the installed applications. MacDroid

creates a policy object of every installed application under its own uid. Users can modify the policies at the application level by using the policy manager named MacDroidPolicyManager which serves as a User Interface provided by MacDroid. Users can also configure MacDroids policies according to required methods and permissions. It is accomplished by intercepting function calls (intent calls) or events passed between software components in the device. Whenever a request get placed to share a resource by a certain method , Macdroid tracks the application UID and hooks the invoked request to the application [9]. After finding the policy which resembles, the invoked method's execution is then controlled by the MacDroid as per its policy.

(Boheung et. al. 2014) proposed a strict and light-weight privilege restriction mechanism to make secure execution environment for users or apps. It expands the concept of Androids sandboxing and classifies application sandboxes into secure sandboxes where users can define their app as secure and non-secure sandboxes for non-secure apps. This is done with the help of policy manager which serves as a GUI for user. Whereas, the user defined policies and secure apps were stored in SecurityApp class. A secure app or app with higher priority can control the permissions of the others running another sandbox such as disable their permissions or stop them from running. The implementation was carried out considering three phases. To create special and isolated execution space for safe apps to provide independent user work space. To control and restrict other apps permissions which could cause security problem such as data leakage. Finally, to implement this mechanism into Android so that it does not impose burdens on the device. Androids security checking mechanism was modified by the approach and altered and supplied new Java classes to implement and verify if the method can correctly restrict apps permission or not. The steps were carried out Android version 4.0.1 (IceCreamSandwich) and was tested on its emulation environments. Androids PackageManagerService and ActivityManagerService class were modified to add the database of cache and to check the apps authenticity of permissions at run-time respectively.

AppGuard is a standalone app and does not require any change to the Android OS. It is based on Inline Reference Monitoring (IRM) technique which allows the user to apply policies on third-party apps. IRM consists of two steps: In the initial step, it alters the binaries of an app and then invokes a security monitor at a run time before any concerned security operation on a program. Secondly, the security monitor assures that the security related policies either permits the operation or substitutes a placeholder code to avoid termination of the app. AppGuard consists of four components i.e. Policies, Rewriter, Management, and Monitor. It provides a set in-built policies to define the security restriction. These policies are written in AOP (Aspect-Oriented Programming Language) to increase code modularity while the security monitor controls the function calls. The task of the rewriter is to take apart classes.dex file from the application package; combine the security barrier defined into the policy database into the code of the application and finally reunite classes.dex with .apk file. The user can apply his policies with the help of Management which is a simple toggle button. Monitor enforces user-defined policies by monitoring the execution of the program. What policy restriction needs to be applied is decided on the configuration of the policy.

Dr. Android and Mr. HideJinseong et al. provided a solution to the problem of Android access control mechanism by dividing the current permissions into more fine-grained permission mechanism. It makes use of three components to achieve the goal. RefineDroid, Mr. Hide (Hide interface to the droid environment) and Dr. Android (Dalvik Rewriter for Android). The authors provided a classification of permissions behavior of apps into four categories and proposed fine-grained alternatives for each category. RefineDroid serves as a fixed examination tool that infers the use of any fine-grained permission in existing apps based on the four proposed categories. Mr. Hide enforces the permissions with no need to modify the OS. It combines many Android API which has the privilege and then it applies the set designated for those API dynamically. The checking of the permission is

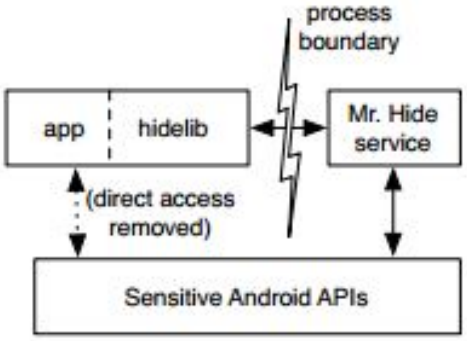

Fig. 4. Mr. Hide Architecture

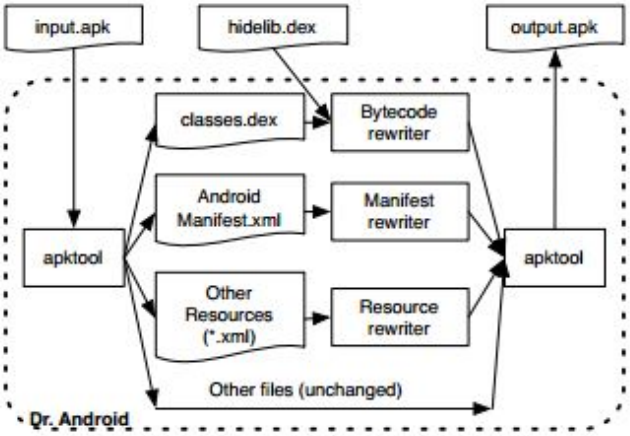

Fig. 5. Dr. Droid Architecture

Technique	Uses Context Information	Type of Security Enhancement	Access Control Policy Type	Policy Executes at	Requires Change in OS	Policies applied to individual apps	Support remote access
CBAC (Bilal et al. 2015)	Yes	Framework Level	Discretionary	Installation and run-time	Yes	Yes	No
MOSES (Yuri et. al. 2014)	Yes	Application Level	Mandatory	Run Time	Yes	No	Yes
Daniele et. Al. 2014	Yes	Framework level	Discretionary	Run-time	Yes	Yes	No
FlaskDroid (Sven et. al 2013)	Yes	Middleware and Kernel	Mandatory,Role Based	Run-time	Yes	Yes	Yes
Crepe (Mauro et. al 2012)	Yes	Middleware layer	Mandatory and Discretionary	Run-time	Yes	No	Yes
AppCage (Yajin et. al. 2015)	No	In-app mechanism	Discretionary	Run-time	No	Yes	Yes
ChainDroid (Qihoui et.al.2013)	No	Application Level	Discretionary	Run-time	Yes	Yes	Yes
DR BACA (Felix et.al. 2013)	Yes	Application Level	Role Based	Run-time	Yes	No	Yes
MacDroid (June et.al.2014)	No	Framework level	Mandatory	Run-time	Yes	Yes	Yes
Boheung et.al. 2014	No	Application Level	Mandatory	Run-time	Yes	Yes	Yes
AppGuard (Michael et. Al 2012)	No	In-app mechanism	Discretionary	Installation and run time	No	Yes	No
Dr Android and Mr. Hide(Jinseong et al 2012)	No	In-app mechanism	Discretionary	Installation and run time	No	Yes	No
FireDroid (Giovanni et al 2013)	Yes	In-app mechanism	Discretionary	Run time	No	Yes	No
Aurasium (Rubin et al. 2012)	No	In-app mechanism	Discretionary	Run time	No	Yes	No

TABLE III
GENERAL EVALUATION OF THE TECHNIQUES

done in Mr. Hides own processes. Hidelib is also provided as a library of the client, and it is responsible for handling every communication that takes place with Mr. Hide and also substitutes a placeholder data in place of users data so that the app can run with no interruptions. Dr. Android installs the fine-grained permissions to the apps with no requirement to access the source code of the app. It replaces existing authorization system with the set of the fine-grained type which is obtained by Mr. Hide.

Aurasium hardens the service of installed application. During the time of app installation, instead of following the traditional procedure, the app is pushed through Aurasium black box. The hardened version of the app is obtained after the app comes out from the black box and is then installed on the users phone. The technique consists of two components: Repackaging mechanism enforces the policies on the app. It inserts an instrumentation code into the app. The second procedure is the monitoring code that is inserted into the apps. The monitoring code interrupts any communication of the app with the Android system and enforces security policies. The communication is intercepted by employing user-level sandboxing and intrusion. The Second task is to reassign a signature to the application which gets destroyed during the hardening process. In order to re-sign the app, the technique itself performs a check on the validity of the app signature, and if it is found to be valid, signature is provided to the app by Aurasium. The Security Manager component enables the user to not only apply policies to any distinct app but also to several apps at a time.

Firedroid interrupts the mechanism of the Android to im-

plement the security policies to the apps. It does not require the internal modules of the OS to recompile. The task to enforce policies is divided into several modules in Firedroid. FireDroid Application Monitor (FDAM) monitors the process of Linux such that whenever an app executes any system call, before performing the system process the notification is sent to FDAM. In the interior of FDAM the Policy Enforcement Point (PEP) gathers the information which is required and passes it to Policy Decision Point (PDP) which recovers the important set of policies from the repository. The repository contains all the policies of concern to a particular process which is monitored. Depending on the policy the PDP can either allow, deny or even terminate the actual procedure. In any process if the Firedroid needs to ask user permission it is done using Firedroid Service (FDS) which serves as UI while the policies at the user level are managed by Policy Administration Point (PAP) are controlled by the user. FireDroid also allows updating the security policies remotely either by SMS, MMS or Bluetooth with the help of Remote Policy Manager (RPM) component. The new policies are first authenticated before being pushed into the PRs of their corresponding FDAMs. However, if the policies are validated they are first stored in the Global Policy Repository (GPR) before being pushed in the FDAM.

VI. General Evaluation of the Techniques

The section focuses on the general aspect of the classification of the technique such as the following:

- Type of Security Enhancement: Where the security enhancement is added such as at application level, Android framework level which includes Android middleware and kernel level or In-app mechanism?
- Type of Access Control policy: Which kind of access control policy is used by each of the technique?
- Type of Policy restriction: When does the technique control the access rights of an app? Whether it is installation time, run time or both?
- Change in existing OS: Does the technique requires to change the Android OS?
- Policies applied to individual apps: Does the technique allows the user to apply policy restriction to individual app or to a group of app defined by the user?
- Support Remote Access: Does the technique also allows policies to be applied or configured remotely by a third party organization, such as a company which provided the device to its employee?
- Uses device context information: Does the approach makes use of user's context to apply the policy restrictions?

A. Type of Security Enhancement

As described in Table II, the security mechanisms are divided into four categories. The techniques that are implemented at the application level such as CBAC, MOSES, ChainDroid, DR BACA, Boheung et.al. 2014. These techniques are standalone and do not require a change to Operating system. These techniques are very easy to be implemented by the user. Most of the techniques are available to the user free of cost. Techniques such as AppFence, FlaskDroid, and Identidroid very closely aligns with the work of CBAC regarding access control based on user location. However, the latter works add an extension to the Android framework. These techniques including MOSES can also be complimented by CBAC for more refined access control mechanism. The other two closely aligned techniques are Crepe and Dr BACA. While CRePE emphasizes on constraining the apps access based on users location DR BACA also provides the same functionality but instead makes use of role-based access control mechanism to achieve an improved company environment map.

Since Android serves an as an open source platform, its source code is available to the outside developers, some of the techniques such as Daniele et. Al. 2014, FlaskDroid (Sven et. al 2013), Crepe (Mauro et. al 2012) focused on extending the Android framework level or to refine the reference monitor of Android. However, these techniques require extensive modification of the Android OS and are very complex. Unlike techniques implemented at the Application level, these methods run in the background, and the user is notified only when it is needed such as in critical decision operation. Crepe lets the user define his own policies which control the behavior of the app at run-time. AppFence adds an enhancement to this technique by blocking an apps access to data from the superior app which has obtained more privileges than the other app.

Some works such as MOSES focus on using virtualization technology into the Android smartphones. Virtualization provides the capability to Android OS to run its many versions on the same hardware. This provides separation and isolation but in return increases the cost of performance. The rest of the techniques implements Secure Container, which serves as a mobile client application and provides an isolation environment at the application layer on the phone. This approach too does not need to modify the OS. For example, AppGurad separates apk file and performs security checks as per the policy selected by the user ahead of execution of dangerous permissions and throws an exception if the permission is not allowed. AppCage makes use of hybrid sandboxes i.e. dex and native to regulate apps access. Aurasium calls the library function by injecting a code during function call. It then dynamically replaces the GOT entries of a process with pointers to safe functions monitored by Aurasium. Similar to AppGuard, Dr. Droid also injects control code around method invocations which are identified as dangerous. FireDroid lets the user install the hardened version of the app for more control on its security policies.

B. Uses Device Context Information

To secure the users data in a device, context information becomes very necessary. Techniques CBAC, IdentiDroid, FlaskDroid, and MOSES makes use of users context to impose their policies restriction at run-time. All the techniques obtain the users context related information either by GPS location, Wi-Fi access points or cellular data. Approaches

Technique	Code and Data Co-operation	Collusion Attacks	Confused Deputy	Sensory Malware	Root Exploit
CBAC (Bilal et al.2015)	✓	✓	✓	✓	-
MOSES (Yuri et. al. 2014)	✓	-	no	-	no
IdentiDroid (Daniele et. Al. 2014)	✓	-	✓	-	✓
FlaskDroid (Sven et. al 2013)	no	-	✓	✓	✓
Crepe (Mauro et. al 2012)	✓	-	no	✓	✓
AppCage (Yajin et. al. 2015)	✓	✓	✓	-	no
ChainDroid (Qihoui et.al.2013)	-	-	✓	-	-
DR BACA (Felix et.al. 2013)	✓	✓	✓	-	✓
MacDroid (June et.al. 2014)	✓	-	-	-	-
Boheung et al. 2014	✓	✓	✓	-	-
AppGuard (Michael et. Al) 2012))	✓	-	no	-	✓
Dr. Android and Mr.Hide(Jinseong et al 2012)	✓	-	no	-	-
FireDroid (Giovanni et al 2013)	✓	✓	✓	-	✓
Aurasium (Rubin et al. 2012)	✓	✓	✓	-	-

TABLE IV
COMPARISON OF TECHNIQUES BASED ON ADVERSARY MODEL

such as MOSES, CBAC deny the permission to use data in certain locations defined by the user such as his work location. FlaskDroid relies on its Userspace Security Server that makes use of device context and state of the system to create Boolean variables which then are used to apply policy restriction. Similarly FireDroid uses contextual information to obtain boolean variables to apply the policies.CBAC uses Androids LocationManager service to obtain GPS co-ordinates and WifiManager service to obtain the Wi-Fi parameters. MOSES uses ContextDetectorSystem component to discover users context. When user changes his location it notifies SecurityProfileManager component which keeps a record of security policies related to specific context. SecurityProfileManager then either activates or deactivates the security policies based on users context. Crepe uses Context Detector Component to continuously keep track of users context from the device sensors and notifies the Policy Administration Point if any recorded context gets active, the Policy Administration Point then activates or deactivates the security policy based on that context. DR BACA uses its context monitor component to continuously keep track of device location.

VII. COMPARISON OF TECHNIQUES BASED ON ATTACK MODEL

This section compares the discussed techniques based on different types of Permission Re-Delegation (PRD) Attacks

on the Android's Middle-ware Security Mechanism and Root Exploits as following:

- Code and Data Co-operation
- Confused Deputy Attack
- Collusion Attacks
- Sensory Malware
- Root Exploits

A. *Code and Data Co-operation*

CBAC implements Intercommunication between apps and Multitasking policies to avoid these type of attacks. It corresponds to type which disallows apps from communicating with each other by blocking intents from one app to another. In CBAC the user depending on his context can disable apps that are running in background and enable any one app of his choice at run-time. Identidroid uses intent filtering to avoid apps from exchanging information with each other. AppCage prevents these attacks by stopping IBinder Interface from gaining its instance of a system service which is known. AppCage implements Security Checks in the function iotcl which is in libc. It also does not permit the system calls to be directly issued by native libraries. MOSES provides security compartments to separate apps related to different contexts and only apps in one compartment can communicate with one another. (Daniele et. Al. 2014) proposed a solution to also hide users sensitive data. While this solution also includes feature of intent filtering which is responsible for preventing apps from communicating with one another at run time. Techniques such as Aurasium hardens an application before being installed on the users phone which also prevents any malicious intent requests within apps.

Crepe uses standard Android mechanisms to avoid inter component communication and is installed with the root certification authority (CA) certificate to prevent unwanted access from third party. Similar mechanism is also used in DR BACA. FlaskDroid operates at the depth of an apps inputs/outputs but is unable to prevent the intent exchange within apps. ChainDroid keeps a track all connections among system mechanisms when a protected API of the system is called to avoid poor apps access. MacDroid employs a hooking mechanism with app which is installed by the user which changes the behavior of an app. This mechanism also perceives any intent calls between apps in the device. Technique proposed by Boheung et al 2014 uses sandboxing concepts defined for both secure and non-secure apps by the user. This sandboxing concept is also responsible to control intent exchanges between apps in different sandbox. AppGuard rewrites the binaries of an app and provides a hardened version of the app without changing the Android OS. These binaries are called by security monitor at run time ahead of every security related sequencer procedure.

B. *Confused Deputy Attack*

The other security concern Android OS faces is confused deputy. Crepe uses the device context location to apply security policies to detect and prevent apps from colluding. DR BACA adds an enhancement to this technique by providing Role based access control to eliminate permission redelegation attacks. MOSES implements lightweight virtualization to support Bring Your Own Device (BYOD) where apps and data related to business are separated from the rest of the device and cannot be accessed from outside apps installed by the user. Bugiel et al. 2012 proposed an approach to detect malevolent behavior of the apps at run-time, which gets overlooked by the user. AppGuard, AppCage and Dr. Droid insert control code if they find any dangerous Java method request.

As mentioned in FireDroid, malware in the wild can use the native code to perform malevolent activities. AppCage uses native sandbox to prevent against this attack. However, AppGuard, MOSES and Crepe are unable to protect the system when it comes to native code misuse. Aurasium successfully prevents app from colluding by interposing library function call by altering the entries of the GOT of a procedure dynamically with pointers to functions monitored by Aurasium. Hence it can detect the dlopen() system call which is used to replace the existing procedure with a new image loaded from built-in code. In case of FireDroid, when FDAM executes the system calls it is able to identify this attack and enforces the essential policies. CBAC modifies the OS to attain the name of the package application which performs an action by calling the PackageManagers getPackagesForUid(int uid) and applying restrictions on package name rather than UIDs. FlaskDroid uses fine-grained access control rules on ICC which restricts certain types of apps to broadcast intents to system apps to prevent confused deputy and collision attacks.

C. *Collusion Attacks*

Very limited techniques provide protection against collusion attacks.CBAC gets the apps package name which is executing an action by using the PackageManagers getPackagesForUid(int uid). Hence CBAC applies limitations not grounded on UID but are converted to denote packages name. So colluding attacks are jammed using all package names related with a UID by means of the getPackageForUid() method. MOSES on the other hand, only separates apps of work use to the users installed apps in separate users profile. However, no information is described on how to prevent this attack in users personal profile. AppCage uses dex and native sandbox to authenticate any apps request to dangerous API or any intent exchange with the system or other apps. When the user installs an app, AppGuard prompts the user to install the modified version of the app secured by AppGuard. Similar kind of mechanism is used by Aurasium to harden the unsafe version of the app to a secure version to provide security against many types of attacks. FireDroid uses Binder to control IPC capable of applying policies which disallows two procedures to interchange descriptors of a file, indicating shared regions of a memory.

Technique	Run-time Overhead	System Memory Overhead	Battery Consumption
CBAC (Bilal et al.2015)	50.981 ms	2.5 percent	Around 5 percent
MOSES (Yuri et. al. 2014)	1544 ms	0.273 0.868 ms	negligible
IdentiDroid (Daniele et. Al. 2014)	-	-	-
FlaskDroid (Sven et. al 2013)	0.452 ms	15.673 -16.1814 micro sec	-
Crepe (Mauro et. al 2012)	1264.46 ms	0.023-0.0044 percent	less than 1 percent
AppCage (Yajin et. al. 2015)	10.7 percent	30 pecent per app	-
ChainDroid (Qihoui et.al.2013)	less than 20 ms	-	-
DR BACA (Felix et.al. 2013)	1430 micro sec 1.43 ms	-	8.4 percent
MacDroid (June et.al. 2014)	50.969 ms	10 percent	-
Boheung et.al. 2014	-	-	-
AppGuard (Michael et. al)	0.0248 to 0.6 ms	Negligible	-
Mr. Hide(Jinseong et al 2012)	10 percent to 50 percent	-	-
FireDroid (Giovanni et al 2013)	0.59 percent	3.9 percent	3.3 percent
Aurasium (Rubin et al. 2012)	1293 ms	-	-

TABLE V
COMPARISON OF TECHNIQUES BASED ON PERFORMANCE AND EFFICIENCY

D. *Sensory Malware*

Not many techniques takes into consideration the role of sensor's malware to Android's security threat.FlaskDroid implements a mechanism to filter data obtained by sensor and sent to sensor listeners which are registered, keeping the keyboard on screen active. This process is done by SensorManager USOM of FlaskDroid. This policy avoids sensor malware attacks such as TouchLogger and TapLogger. Another policy repudiates any admission to the audio record functionality which is implemented in the MediaRecorderClient USOM while a call is in progress. This policy prevents against SoundComber attack. CBAC employs resource restriction policies to deny any access to system resource such as camera,Bluetooth, mic etc depending on the context selected by the user. Similar policies are used in Crepe to enable or disable device sensor. however, in both these techniques the user cannot disable sensors which shows device's context such as GPS,Wi-Fi.

E. *Root Exploits*

Crepe uses PKI system which has X.509 certificates in order to avoid root exploits. The scheme is mounted with the certificate from CA; so every message the scheme receives, it assures if it is coming from CA such as an explicit organizations branch. Since, MOSES implements TaintDroid, it also takes over its drawbacks such as the inefficiency to overcome root exploits like rootkits. In order to identify a root attack, Firedroid counts the number of fork executed by each procedure and then limits the number of forks to be executed by each procedure. By this process it restricts the users procedure limit forced by the Android OS. AppGuard monitors API-calls such as Runtime.exec() to overcome this attack. However, AppCage is based on the assumption that an attacker does not have root privilege and hence no such mechanism is developed to avoid root exploits. SE Linux when used in compliment to DR BACA adds to the system security to prevent these attacks. Aurasium interprets function calls such as ioctl(),getaddrinfo() and connect(),dlopen(), fork() and execvp()read(), write(), open() and sends a warning by executing the su command whenever an app tries to obtain root access on a possibly rooted phone.

VIII. COMPARISON OF THE TECHNIQUES BASED ON THEIR EFFICIENCY

This section focuses on the performance metrics observed in the above techniques.The following criteria is studied:

- Run-time Overhead
- System Memory Overhead
- Battery Consumption

A. *Run-time Overhead*

The performance overhead in CBAC is measured due to interrupting apps authorizations, user data accesses, Intents, and system peripherals access. It was observed that most of the system memory was consumed by restricting apps and certain methods which continuously run in background such as LocationService. The performance of MOSES was compared with Stock Android and it showed overhead from 82.1 percent to 131.1 percent without ABAC rules. Also, time consumption increased with the increased rules. It was observed that writing to a file took most of the time since Java reflection method implemented in the third check strategy of the MOSES was time consuming operation. On the other hand in CREPE the time overhead is not directly proportional to the rules quantity. However, when a new rule is added it is checked by the CREPE which also incurred some overhead adding up to the total overhead since it can alter the apps execution path. Along with that deactivating a policy took more time than activating it.

AppCage shows minimal overhead in performance for byte-code and native code. Also, in order to restrict branches and writing into the memory instructions are added by the AppCage. The performance overhead of native sandbox is higher (10.7 percent) since it uses string sort benchmark which makes use of memmove function in libc extensively in order to move large data blocks. Firedroid incurres a large overhead of 97.5 percent in I/O tests since it normalizes the path whenever file are opened demanding additional period. Also the function call ptrace adds to overhead to change contexts to pass control to FDAM. AppCages process of adding components to original app and creating a wrapper app and then signing it along with rewriting the large binaries adds to time consumption. DR Hides interprocess communication adds to the performance overhead. FlaskDroid incurs more deviation in standard due to varying system loads. Most of the overhead in Aurasium occurs whenever an app requests API which involves IPC with remote services of the system. This adds to most of the overhead since the communication with the Binder needs to be parsed fully by Aurasium.

B. *System Memory Overhead*

Storage overhead of MOSES is relatively low it simply works with one copy of apps executables. While Components such as policy and database of contexts, and authorized third parties certificates cache are the most space consuming in CREPE. Size of each application is increased by 52kb in Aurasium. Similarly, AppCage adds 50kb to each original app by adding other components and generating wrapper app for sandboxes.

C. *Battery Consumption*

The battery consumption in CABAC dropped by 5 percent every hour while not applying the policies and updating users context every 5 seconds. The battery usage turned out to be negligible. Similar to CBAC, context detecting sensors added a contributed to decrease in battery power in CREPE and DR BACA. Energy consumption of MOSES is also less when it is running or switching locations. Techniques which makes use of users context usually have battery overhead due to device sensors which collect users location such as GPS, Wi-Fi etc. Since these sensors continuously run in the background to monitor the location.

IX. Discussion- Towards Efficient scheme

Based on the discussion the following points can be summarized on how to develop security extensions for Android to provide dynamic access rights to apps:

- According to Chin et al. 2011 Changes are required to be done in the Android OS to avoid accidental disclosure of its components to third party apps.
- Intents used for both type of communication (such as inter and intra application), if not properly handled by its developer may render its app to outside attack [37].
- Communication taking place within apps or outside of apps needs to be handled by dissimilar ways [37].
- If the access to a permission is denied by the user in particular context then the security mechanism should be able to provide placeholder to avoid the app from crashing.
- The flow of the data and control through various modules should also be tracked by the scheme to obtain the generalization of the behavior of an app.
- An approach could also warn the user if his private information gets accessed in a particular context denied by him.
- Developers should also keep in mind on root exploits and sensory malware while formulating their approach. Techniques should be robust if the app uses native code.
- A scheme should also assure that an app does not fake the time and location of the device to obtain access to certain APIs.
- From the evaluation of techniques it can be observed that methods which hardens the application usually increases the size of an app and also re-installation time of the app. An efficient approach should be able to perform these operations with minimal overhead.
- Along with that, battery consumption should also be considered while designing an approach.

X. Conclusion

Androids permission system however notifies the user on the dangerous permissions requested by an app at the installation time. But it gives no control to the user to control those permissions. The paper discusses the flaws in the Androids current permission mechanism and the vulnerabilities in its security components. The paper focuses on the latest defense solutions that have been proposed to address the problem and provides a brief description and classification of those techniques. Also, the paper provides a proportional assessment of discussed techniques based on specific benchmarks such as their general characteristics, attackers model and their performance and efficiency. It was observed that a specific tactic did

not provide wide-ranging result to all the considered factors. The paper also discusses some of the significant features that needs to be well-thought-out for forthcoming development of a proficient defense mechanism. Hence, there is still a necessity for a comprehensive solution for dynamic access rights which could provide substantial practical supplies to the Android mechanism.

REFERENCES

[1] Bilal Shebaro, Oyindamola Oluwatimi, and Elisa Bertino, Fellow, IEEE, *Context-Based Access Control Systems for Mobile Devices*, in Proc. IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 12, NO. 2, MARCH/APRIL 2015

[2] Yury Zhauniarovich, Giovanni Russello,Mauro Conti, Bruno Crispo and Earlence Fernandes *MOSES: Supporting and Enforcing Security Profiles on Smartphones*, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 11, NO. 3, MAY-JUNE 2014

[3] Daniele Midi,Oyindamola Oluwatimi,Bilal Shebaro,Elisa Bertino *Demo Overview: Privacy-Enhancing Features of IdentiDroid*, CCS '14 Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security

[4] Sven Bugiel,Stephan Heuser,Ahmad-Reza Sadeghi *Flexible and Fine-Grained Mandatory Access Control on Android for Diverse Security and Privacy Policies*, 22nd USENIX Security Symposium. August 14 to 16, 2013 Washington, D.C., USA

[5] Mauro Conti,Bruno Crispo, Earlence Fernandes, and Yury Zhauniarovich, *CRiPE: A System for Enforcing Fine-Grained Context-Related Policies on Android*, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL. 7, NO. 5, OCTOBER 2012

[6] Yajin Zhou,Kunal Patel,Lei Wu,Zhi Wang,Xuxian Jiang *Hybrid User-level Sandboxing of Third-party Android Apps*, ASIA CCS '15 Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security Pages 19-30

[7] Qihui Zhou1, Dan Wang, Yan Zhang, Bo Qin, Aimin Yu, Baohua Zhao *ChainDroid: Safe and Flexible Access to Protected Android Resources Based on Call Chain*, 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications

[8] Felix Rohrer,Yuting Zhang,Lou Chitkushev,Tanya Zlateva , *DR BACA: Dynamic Role Based Access Control for Android*, ACSAC '13 Proceedings of the 29th Annual Computer Security Applications Conference Pages 299-308

[9] June-seung Na, Younghoon Kim, Young-June Choi,Wooguil Pak *Mandatory Access Control for Android Application*, Information and Communication Technology Convergence (ICTC), 2014 International Conference on Oct. 2014

[10] Boheung Chung, Youngsung Jeon, and Jeongnyeo Kim *User-defined Privilege Restriction Mechanism for Secure Execution Environments on Android*, Information and Communication Technology Convergence (ICTC), 2014 International Conference on Date of Conference: 22-24 Oct. 2014

[11] 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.

[12] W. Enck, M. Ongtang, and P.McDaniel, *On lightweight mobile phone application certification,*, in Proc. CCS 09, 2009, pp. 235245

[13] Yuan Zhang, Min Yang, Zhemin Yang, Guofei Gu, Peng Ning, and Binyu Zang, *Permission Use Analysis for Vetting Undesirable Behaviors in Android Apps*, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL. 9, NO. 11, NOVEMBER 2014

[14] Yuhong Nan, Min Yang, Zhemin Yang, and Shunfan Zhou, Guofei Gu, Xiaofeng Wang, *UIPicker: User-Input Privacy Identification in Mobile Applications* , Indiana University Bloomington 24th USENIX Security Symposium August 1214, 2015

[15] ARZT, S., RASTHOFER, S., FRITZ, C., BODDEN, E., BARTEL, A., KLEIN, J., LE TRAON, Y., OCTEAU, D., AND MCDANIEL, P., *Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps*, In Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (2014), ACM, p. 29.

[16] SMALLEY, S., AND CRAIG, R., *Security enhanced (se) android: Bringing flexible mac to android.*, In The 20th Annual Network and Distributed System Security (NDSS) (2013).

[17] BUGIEL, S., DAVI, L., DMITRIENKO, A., FISCHER, T., SADEGHI, A.-R., AND SHASTRY, B., *Towards taming privilege escalation attacks on android.*, In NDSS (2012).

[18] ZHANG, Y., YANG, M., XU, B., YANG, Z., GU, G., NING, P., WANG, X. S., AND ZANG, B. , *Vetting undesirable behaviors in android apps with permission use analysis.* , In Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security (2013), ACM, pp. 6116

[19] HEUSER, S., NADKARNI, A., ENCK, W., AND SADEGHI, A.- R. , *Asm: A programmable interface for extending android security*, In Proc. 23rd USENIX Security Symposium (SEC14) (2014).

[20] ENCK, W., GILBERT, P., CHUN, B.-G., COX, L. P., JUNG, J., MCDANIEL, P., AND SHETH, A. N., . *Taintdroid: an information flow tracking system for real-time privacy monitoring on smartphones.* , Proceeding OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation Article No. 1-6

[21] Deguang Kong, Lei Cen, Hongxia Jin , *AUTOREB: Automatically Understanding the Review-to-Behavior Fidelity in Android Applications* , Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security CCS '15

[22] Yiming Jing, Gail-Joon Ahn, Senior Member, IEEE, Ziming Zhao, Student Member, IEEE, and Hongxin Hu, 98Member, *Towards Automated Risk Assessment and Mitigation of Mobile Applications*, IEEE IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 12, NO. 5, SEPTEMBER/OCTOBER 2015

[23] Zhou, Y., and Jiang, X. Dissecting android malware:, *Characterization and evolution*, In S and P (2012), IEEE Computer Society.

[24] Porter Felt, A., Finifter, M., Chin, E., Hanna, S., and Wagner, D. *A survey of mobile malware in the wild.*, In 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM) (2011), ACM.

[25] Bhaskar Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, Ian Molloy TJ Watson, *Android Permissions: A Perspective Combining Risks and Benefits*, Proceeding SACMAT '12 Proceedings of the 17th ACM symposium on Access Control Models and Technologies Pages 13-22

[26] S. Motiee, K. Hawkey, and K. Beznosov, *Do Windows Users Follow the Principle of Least Privilege?: Investigating User Account Control Practices,*, Proc. Sixth Symp. Usable Privacy and Security, 2010.

[27] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang. *The Impact of Vendor Customizations on Android Security.* In Proceedings of the 20th ACM Conference on Computer and Communications Security, 2013.

[28] http://www.informationweek.com/mobile/mobile-devices/smartphone-sales-poised-for-rapid-growth/d/d-id/1112214?,

[29] How to make android apps: https://blog.udemy.com/how-to-make-android-apps/ ,

[30] http://developer.android.com/reference/android/content/Intent.html ,

[31] Stephen, Smalley and Robert, Craig *Security Enhanced (SE) Android: Bringing Flexible MAC to Android* Trusted Systems Research National Security Agency Jun 2014.

[32] Enrico, Bacis and Simone, Mutti and Stefano, Paraboschi *AppPolicy-Modules: Mandatory Access Control for Third-Party Apps* 2015.

[33] Michael, Backesand Sebastian, Gerling and Christian, Hammer and Matteo, Maffei and Philipp, von Styp-Rekowsky *AppGuard - Enforcing User Requirements on Android Apps* 2015Proceedings of the 19th international conference on Tools and Algorithms for the Construction and Analysis of Systems, 2013, pp 543-548

[34] Jinseong, Jeon and Kristopher, K. and Micinski, Jeffrey and A., Vaughan and Ari, Fogel and Nikhilesh, Reddy and Jeffrey S. Foster and Todd, Millstein*Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications* "Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices,SPSM 2012 pp: 3-14

[35] Giovanni, Russello and Arturo Blas Jimenez and Habib Naderi and Wannes van der Mark *FireDroid: Hardening Security in Almost-Stock Android* Proceedings of the 29th Annual Computer Security Applications Conference 2013 pp 319-328

[36] Rubin Xu and Hassen Sadi and Ross Anderson *Aurasium: Practical Policy Enforcement for Android Applications* 21st USENIX Security Symposium 2012

[37] Chin, E. and Felt, A.P and Greenwood, K. and Wagner, D *Analyzing inter-application communication in Android* Proceedings of the 9th International Conference on Mobile Systems, 2011 pp 239-252