PRIVACY PRESERVATION ON MOBILE SYSTEMS USING CONTEXT-AWARE ROLE BASED ACCESS CONTROL

A Thesis submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Engineering

by
Juhar Ahmed Abdella

June 2016
İZMİR

We approve the thesis of **Juhar Ahmed ABDELLA**

Examining Committee Members:

Assist. Prof. Dr. Mustafa ÖZUYSAL
Department of Computer Engineering, Izmir Institute of Technology

Assist. Prof. Dr. Şerap ŞAHİN
Department of Computer Engineering, Izmir Institute of Technology

Assist. Prof. Dr. Tuncay ERCAN
Department of Computer Engineering, Yaşar University

06 June 2016

Assist. Prof. Dr. Mustafa ÖZUYSAL
Supervisor, Department of
Computer Engineering
Izmir Institute of Technology

Dr. Emrah TOMUR
Co-Supervisor, Department of
Computer Engineering
Izmir Institute of Technology

Prof. Dr. Yusuf Murat ERTEN
Head of the Department
of Computer Engineering

Prof. Dr. Bilge KARAÇALI
Dean of the Graduate School of
Engineering and Sciences

ACKNOWLEDGEMENTS

I am very grateful to my thesis supervisors Asst.Prof.Dr.Mustafa Ozuysal and Dr.Emrah Tomur for their continuous support, encouragement and guidance from the initial step to the final stage. This thesis work could not have been realized without their intelligent counseling. They have been always there to help me when I need them. It is amazingly pleasant to work with advisors like them. They gave me the opportunity to work on my own and at the same time the advice to get back when my steps faltered. They were more than thesis advisors for me.

Besides my advisors, my sincere thanks goes to all my professors in IYTE from whom I have taken courses and advices during my graduate study. The knowledge I got from them has helped me to develop an understanding of my research subject.

I would like to thank my family. The care, love and support I received from them helped me to stand challenges and concentrate on my study. I would also like to thank all my friends who have stayed besides me during difficult times.

I would like to also appreciate the financial support I received from Turkish government during the course of my graduate studies.

Last but not least, I am indebted to my wife Elham Asmamaw for her tireless patience of the lonely nights she spent without me when I left her for study in Turkey. Her love and support has been a constant source of power and strength all these years.

ABSTRACT

PRIVACY PRESERVATION ON MOBILE SYSTEMS USING CONTEXT-AWARE ROLE BASED ACCESS CONTROL

Existing mobile platforms require the user to manually grant and revoke permissions to applications. Once the user grants a given permission to an application, the application can use it without limit unless the user manually revokes the permission. This has become the reason for a lot of privacy problems. One of the solutions suggested by a lot of researchers is Context Aware Access Control (CAAC). However, dealing with policy configurations at permission level becomes very complex as the number of policy rules to configure will become very large. For instance, if there are A applications, P permissions and C contexts, the user may have to deal with A x P x C number of policy configurations. Therefore, we propose a Context-Aware Role-Based Access Control (CA-RBAC) model that can provide dynamic permission granting and revoking while keeping the number of policy rules as small as possible. We demonstrate our model based on Android. In our model, Android applications are assigned roles where roles contain a set of permissions and contexts are associated with permissions. Permissions are activated and deactivated for the containing role based on the associated contexts. Our approach is unique in that our system associates contexts with permissions as opposed to existing similar works which associate contexts with roles. As a proof of concept, we have developed a prototype application called CA-ARBAC (Context-Aware Android Role Based Access Control). We have also performed various tests using our application and the result shows that our model is working as desired.

ÖZET

MOBİL SİSTEMLERDE BAĞAM BİLİNÇLİ ROLE TABANLI ERİŞİM DENETİMİ İLE KİŞİSEL GİZLİLİĞİN KORUNMASI

Mevcut mobil platformlar kullanıcıların uygulamalara izinleri elle vermesini ya da iptal etmesini zorunlu tutmaktadır. Kullanıcı bir uygulamaya bir izin verdikten sonra, elle izni iptal etmediği takdirde, o uygulama o izni sınırsız olarak kullanabilmektedir. Bu pek çok kişisel gizlilik sorunlarına neden olmaktadır. Bir çok araştırmacı tarafından önerilen çözümlerden biri bağlam bilinçli rol tabanlı erişim denetimidir. Ancak, izin düzeyinde politika yapılandırmaları ile ilgilenmek çok karmaşık hale gelir, çünkü politika kurallarının sayısı çok fazladır. Örneğin, A tane uygulama, P tane izin ve C tane bağlam varsa, kullanıcı A x P x C adet politika yapılandırması ile uğraşmak zorunda kalabilir. Bu nedenle politika kural sayısını mümkün olduğunca küçük tutarken aynı zamanda dinamik izin verme ve iptal etme fonksiyonu sağlayabilecek bağlam bilinçli rol tabanlı erişim denetimi kullanan bir model önermekteyiz. Modelimizi Android üzerinde göstermekteyiz. Modelimizde, Android uygulamaları için izin kümesi içeren roller atanmakta ve bağlamlar da izinler ile ilişkilendirilmektedir. İzinler, onları içeren rol için ilişkili bağlamlara dayalı olarak aktive edilmekte ya da devre dışı bırakılmaktadır. Yaklaşımımız roller ile bağlamları ilişkilendiren benzer çalışmaların aksine izinler ile bağlamları ilişkilendirdiği için benzerlerinden ayrılmaktadır. Önerilen kavramların kanıtı olarak CA-ARBAC (Context-Aware Android Role Based Access Control) adı verilen bir prototip uygulama geliştirilmiştir. Ayrıca, prototip uygulamamızı kullanarak çeşitli testler gerçekleştirilmiş ve sonuçlar önerilen modelin arzu edildiği şekilde çalışıtığını göstermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AA DB: Role Assignment Database

APS: Android Permission System

ARM: Application Role Mapping

ASF: Android Security Framework

ASM: Android Security Modules

BYOD: Bring Your Own Device

CAAC: Context-Aware Access Control

CA-RBAC: Context Aware Role Based Access Control

CA-ARBAC: Context Aware Android Role Based Access Control

PAP: Policy Administration Point (a user interface used to configure security policy)

PA DB: Permission Assignment Database

PDP: Policy Decision Point

PEP: Policy Enforcement Point

PRM: Permission Role Mapping

RBAC: Role Based Access Control

CHAPTER 1

INTRODUCTION

With the advent of more powerful, affordable and multi-purpose mobile phones, the world is shifting towards mobile computing. Nowadays, mobile devices are capable of doing many of the things which were normally done by the traditional desktop and laptop computers. Thus, in addition to personal use, companies have also started using mobile devices for enterprise Bring Your Own Device (BYOD) environment. According to the report from market research team eMarketer [1], there were around 1.13 billion smartphone users in 2012. Nearly 2.5 billion people or 35 % of the global population is expected to use smartphones by the end of 2017. In spite of these figures, the security and privacy aspect of smart phones is at its infancy stage. There are a lot of security and privacy issues related to mobile devices. As mobile phones are always attached with the user, users' privacy is of special importance. A study from [2], shows that currently Google's Android is the most popular mobile platform dominating 85% of the smart phone market. Due to this popularity, its open nature and the weaknesses in the permission system, Android is also the most targeted mobile platform by attackers. Kaspersky Labs [3] estimate that Android is the recipient of more than 98 percent of the mobile threats currently in existence. In the first half of 2014 alone, Kaspersky researchers identified 175,442 new, unique malicious programs designed for Android. A joint report from Kaspersky Lab and INTERPOL [3], collected based on more than five million mobile devices secured by Kaspersky security products between 2013 and 2014 indicates that the number of attacks per month exploded from 69,000 per month to almost 650,000. In that same timeframe, the number of users attacked also increased rapidly, from 35,000 to 242,000. It's worth noting that almost 60% of malware detections are related to some form of information theft.

The current permission systems do not support the dynamic alteration of applications' privileges based on the context of the user opening the door for malicious use of permissions and privacy leak without the user's consent [4], [5], [6]. In mobile systems, the environment around the user changes frequently which makes it implausible to rely on static policy configurations. The dangerousness of permissions depends on the present condition of the mobile user. A permission which is normal at some condition may be very dangerous at other occasion. As is the case in the existing systems, manually adjusting permission grants from time to time seems impractical for a long list of permissions like that of in Android.

Because of a lot of privacy issues, Google started modifying Android Permission system (APS) beginning from Android version 4.3 although it did not officially declare the change until Android version 6. Google announced the new permission system on Android version 6 with more other modifications including making the permission system fine-grained and dynamic. The old static and coarse-grained APS has been discussed in more detail in the related works part.

Google has also made improvements concerning usability issues in Android version 6. In earlier versions of Android, every time the user wants to install new application, he had to review a long list of permissions and accept all of them before installing the application. This had significantly affected the usability of the system. The new APS is similar to that of Apple's iOS permission system in that permission is not requested during installation time as before. Instead, users have to grant permissions to applications during runtime through a popup window that asks for permission. Once the user grants permission to a specific application, the permission will be added to the list of allowed permissions for that application and it will permanently stay granted until the user manually revokes the permission. However, if the user denies the requested permission, the decision will not be permanent. The application has the chance to request the permission at a later time.

The negative side of this approach is that permission requests could become annoying if users have to be asked confirmation for each specific permission requested by applications. As a solution to this problem, Google grouped related permissions together. Therefore, when a user is asked to grant permission, he is actually being asked to grant many permissions at a time not just a single permission. For example, when the user grants PHONE permission, he is granting six permissions: Directly call phone numbers, Write call log, Read call log, Reroute outgoing calls, Modify phone state and Make calls without user's intervention permissions. This may result in privacy problems because users are being made to grant all permissions inside the group even though they do not want to grant some of the permissions inside the group. The other important point in APS is that the new system grants Normal Permissions such as internet permission to all applications by default. Users will not be asked to grant access to the Internet and it is not even possible to revoke it, even if they wanted to do so.

Despite a lot of other changes, Google did not take context awareness into consideration yet. Context-aware access control is still not possible with the new APS. Once the user grants permission to an application, the application can use the permission at all conditions without limit. Nevertheless, taking the advantage of mobile device sensors and the dynamic nature of mobile phones, other researchers have introduced context awareness to Android permission system as discussed in the related works section.

However, existing context aware access control models give little attention to usability in contrary to the fact that most users of smartphones are ordinary users which have little or no knowledge about security and privacy. To the extent of our knowledge, most of the systems require users to laboriously configure detailed policies. Privacy policy rules have to be configured for each individual entity separately. The problem with such kinds of models is that the user has to deal with large number of policy configurations. For example, in APS, we may have to deal with approximately 140 permissions. Generally, if we have A number of applications and P number of permissions, in the worst case, we need to deal with A x P number of policy rules. In addition, in the case where context is considered, context policy configuration has to be performed for each permission per each application. Users usually need to associate more than one context with a single permission. If we have C number of contexts for each permission on average, we will end up with A x P x C number of policy rules. For systems which have large number of permissions and installed applications, configuring this much number of policy rules does not seem fascinating especially for the ordinary users.

In fact, different studies show that most mobile device users are not interested or not able to configure detailed policies; rather they prefer to accept every permission request without careful examination resulting in over privileged applications. [7] for example, made an investigation to see if Android users understand APS and pay attention to privacy risks during application installation. The result shows that only 3% of users could correctly understand the permissions and only 17% of the users give their attention to permissions requested by applications.

To overcome the previous problems, we propose a permission system that combines RBAC with CAAC. Our model named Context-Aware Android Role Based Access Control (CA-ARBAC) works by assigning roles to applications where roles consist of a list of permissions which will be activated and deactivated for the containing role depending on a set of contexts. We found that CA-RBAC is a promising method to implement better privacy preservation without significant effect on the usability of the permission system. In CA-ARBAC, users are not required to deal with large number of permissions; instead they just need to assign roles to applications. In other words, the kind of permission grouping adopted by Google to promote usability is replaced by RBAC in our system without compromising the privacy. However, there is small amount of overhead at the beginning. The user has to configure policies initially. Once created, roles can be used for as many applications as needed. Furthermore, it is possible to have default roles such that ordinary users who have difficulty in creating their own roles can use them.

Therefore, in our proposed method, the number of policy rules can be reduced to A x R where R is the number of roles. Moreover, in our model, since role-permission and permission-context maps are independent of applications, we do not need to redo these configurations if applications have to be uninstalled and installed again. But without RBAC, every time we need to uninstall and install back a given application, all the rules related to that application have to be reconfigured since they are dependent on the application. Altogether, our method satisfies three requirements at the same time: least privilege, dynamic permission granting and revoking and keeping the number of policy rules as small as possible.

To create roles, we followed a method of categorizing applications into logical groups. Examples of functional groups include messenger applications, photography applications, multimedia applications, travel applications etc. Roles correspond to these functional groups. Different functional groups require different type and number of permissions and hence will be assigned different kinds of roles. This approach should not be taken as the best way of creating roles. It is rather a one kind of approach chosen by us to demonstrate our model. We recognize that this approach has its own limitations. Being able to create roles which contain an optimum number of permissions is one of the challenges of our system. We believe that there can be better way of doing this. However, as the main goal of this thesis is not providing an appropriate method of creating roles, we have chosen to postpone this work to the future. Detailed discussion about role creation and the limitation of this approach is presented in Chapter 6.

We argue that CA-ARBAC yields an improved privacy preserving system with little or no effect on the usability of the system. Primarily, CAAC helps protect the privacy of the user by allowing dynamic alteration of application privileges based on pre-defined contexts. Secondly, as RBAC is a known method of applying Principle of Least Privilege (PLP), it allows user privacy protection by enabling the user to give a minimum number of permissions as he wishes preventing over-privileged applications. Furthermore, we also believe that CA-ARBAC promotes usability. As explained earlier, the numbers of policy rules needed to be configured is less in CA-ARBAC as compared to the access control models suggested by others.

Contributions:

The concept of CA-RBAC is not totally new to the mobile environment and to Android. There are few prior works which have introduced CA-RBAC to Android for different purposes. The following are the novel contributions made by this thesis:

- A new CA-RBAC model for APS that assigns roles to applications and associates contexts with permissions allowing PLP and dynamic granting & revoking of application permissions with little effect on usability
- A dynamic and fine-grained permission system for Android versions earlier than Android version 6
- A new CA-RBAC architecture for Android permission system that can possibly be integrated to Android Security Modules (ASM) [8]

The rest of this thesis is organized as follows: Chapter 2 discusses related work. We revise Android background and its security mechanisms in Chapter 3. Our proposed design for CA-ARBAC system is presented in Chapter 4. Chapter 5 explains the implementation of our system. Our system is further demonstrated with examples and experimental test in Chapter 6. We discuss our work and indicate future works in Chapter 7. Finally, Chapter 8 summarizes and concludes the thesis.

CHAPTER 2

RELATED WORKS

Going back to Android versions earlier than Android version 4.3, Android permission system (APS) was not only coarse-grained but also static. By coarse grained, it means that the user has to accept all the permissions requested by the application during installation to be able to install the application. It was not possible to accept only some of the permissions and yet be able to install the application. Moreover, there was no way of revoking permissions later on during run time which makes it static. This weakness of the permission system had been the cause for a lot of over privileged applications that harm the privacy of the user. In response to these security and privacy risks, a lot of researchers have tried to enhance Android's security system in different ways. Hence, the first generation of researches focused on making Android permission system fine-grained and dynamic. Some of the most prominent papers published on this topic are Apex [9], AppGuard [10], BlurSense [11], [12], Flaskdroid [13], TISSA [14], MockDroid [15] and Dr. Android and Mr. Hide [16].

The second generation of researches started introducing context aware access control to the mobile environment. The majority of them are designed for all platforms in general and some of them are for Android in particular. Some of the most recent papers which fall under this category include [17], ConUcon [18], CRêPE [19] and ConXsense [20]. At the same time, there are other researchers who worked on role based access control. Furthermore, a limited number of papers have been published on context-aware role based access control.

Our CA-ARBAC system is a combination of two types of access control models: RBAC and CAAC. In the consecutive sections, we will see at previous publications related to our work in different ways. Some of them are related to pure RBAC. Others are linked to CAAC only. Few others are associated to both RBAC and CAAC. Therefore, we chose to examine previous articles closely related to our work by dividing them into three groups: those that focus on RBAC, those that deal with CAAC and those that combine both of these (CA-ARBAC models).

2.1.1. RBAC Models

2.1.2. MPDROID

MPDROID [21] is good example of pure RBAC model that is close to our approach. It is a security framework that supports two kinds of access control models at two layers of Android system: role-based access control at the application framework layer and mandatory access control at the kernel layer. At the application framework layer, it enhances APS with role-based access control to provide fine-grained access control. This enables users to define their own security policy and control malicious applications. At the kernel layer, it implements mandatory access control to allow administrators enforce fine-grained access control. Administrators can limit activities of applications and their processes according to a centralized security policy. Similar to our system, users authorize Android applications by assigning roles instead of permissions. But MPDROID doesn’t take context into account.

2.2. CAAC Models

Mobile devices are dynamic by nature. A lot of things around mobile environment change from time to time which makes them suitable for context information collection. Leveraging this nature of mobility, a lot of researchers have tried to present various kinds of context aware access control models on different mobile platforms including Android mobile phone. We would like to have a look at four of the most common and recent ones: Bilal Shebaro Et Al. [17], ConUcon [18], CRêPE [19], and ConXsense [20].

Our context aware access control policy model part is analogous to that of Bilal Shebaro Et Al. [17]. Similar to our proposed system, it associates android permission with contexts. However, Bilal Shebaro Et Al. [17] is a pure CAAC model unlike our model which is a hybrid of RBAC and CAAC. Moreover, Bilal Shebaro Et Al. [17] works only for two kinds of contexts: location and time. Our system is designed to support different kinds of contexts.

2.2.1. ConUcon

ConUcon [18] proposed a general context-aware usage control model that can be used in different mobile platforms. It uses context information to protect privacy and to control resource usage. ConUcon is a context aware access control model that is applied in a system wide manner. It doesn’t allow per application configuration of context policy. ConUcon is different from other context aware access control models in that it supports active context usage control i.e. context check is not only performed prior to resource access, but also during the access. In addition to proposing the model, they also implemented the model in Android to provide an interface that enables users to configure their policy dynamically in a context-aware and fine-grained manner. Two kinds of contexts (system and environmental contexts) such as CPU rate, battery, device location and time are used in ConUcon.

2.2.2. CRêPE

CRêPE [19] developed a system that enforces fine-grained context- related policies on Android. In addition to local configuration, CRêPE allows remote policy configurations via methods such as SMS and Bluetooth. It also allows phone users and administrators to define context policies in a system-wide manner. The kinds of contexts supported by CRêPE include sensor contexts like time & location, contexts generated by further processing on these data or contexts coming from particular interactions of these sensors with the users or third parties. CRêPE allows dynamic and active context policy management i.e. it is not only possible to create and or modify contexts policies at runtime but also it is possible to stop ongoing service/application. For example, disabling audio recording while entering to meeting room does not only require denying new requests to record audio but also needs to stop ongoing recording operations if any. In CRêPE system, access control policies are stored as context and policy pairs. Since all of these couples may not be active at a given time, CRêPE works by keeping the subset of active policies at a given time in a different place.

2.2.3. ConXsense

ConXsense [20] is a framework for context aware access control on mobile devices based on context Classification. ConXsense's main goal is aimed at solving the usability issues ignored by most other context aware access control models before it. It is unique from other similar models in that it doesn't require users to configure policies. Instead, it is based on a probabilistic approach that automatically classifies contexts according to their security and privacy risks. It uses machine learning and context sensing for automatic classification of contexts. Earlier works on context-aware access control systems usually need either users to laboriously configure policies or they rely on pre-defined policies not necessarily indicating the real preferences of users. ConXsense is applied for protection against device misuse using a dynamic device lock and protection against sensory malware. It is implemented on Android permission system focusing on usability.

2.3. CA-RBAC Models

There exist also works that combine aspects of RBAC model with context awareness. Some of the recent studies that fall under this category are: DR BACA [22], CtRBAC [23], CA-RBAC [24], Kangsoo Jung Et Al. [25], and RBACA [26]. We will look at the these five papers in the following sections

2.3.1. DR BACA

DR BACA [22] offers an RBAC system similar to that in traditional desktop computers. It allows the management of multiple users on a single Android mobile device by controlling resource access based on the role of the current user using the device. It also allows a single Android device to be used by different users without interference. At the same time, a single user can use different devices seamlessly. Similar to the traditional RBAC system, users are assigned roles. However, instead of associating roles to permissions directly, DR BACA introduces an additional layer they called rule. The rules can either be applied on applications or on permissions. At the application level, DR BACA can control user's execution of applications. At the permission level, it is used to allow or deny application permission requests based on the role of current user.

DR BACA also supports dynamic RBAC by taking advantage of the context-aware capabilities of mobile devices and Near Field communication (NFC) technology. By associating rules with context, DR BACA provides fine-grained Role Based Access Control (RBAC) at both the application and permission levels.

2.3.2. CtRBAC

CtRBAC (context-related role based access) [23] proposes a finer access control mechanism for mobile systems based on traditional role based access control enhanced with context aware access control. In CtRBAC, users are categorized according to their access rights and each user is allowed to possess one role at a time. Access to resources is determined based on the role of the user currently using the device. CtRBAC can also dynamically change the permissions of users depending on the contextual information obtained from information of the user and system environment. The administrator is responsible for creating roles and defining access control policies. The phone owner is considered as the system administrator. CtRBAC did not provide implementation.

2.3.3. CA-RBAC

CA-RBAC [24] proposes an access control model that combines RBAC with context awareness for users in ubiquitous computing environments. As opposed to the traditional RBAC model where User assignment (UA) and Permission assignment are handled by administrators, in CA-RBAC model, UA and PA are performed dynamically depending on context satisfaction.

To be able to achieve dynamic UA and PA, CA-RBAC model uses various access control algorithms including role assignment, role delegation, role revocation, permission modification, and permission restoration. Personalized access control that considers the user’s preferences is also included. They haven’t implemented it. CA-RBAC is similar to our system in that it dynamically assigns permissions to roles according to the current context. However, like most others, CA-RBAC is designed for mobile users in ubiquitous computing environments not for applications. In addition, our system does not change roles contextually to avoid unnecessary creation of roles.

2.3.4. User Relation Ship based Context Aware Role Based Access Control

Dynamic RBAC has also been designed using user relationship as contextual information. Kangsoo Jung Et Al. [25] is a relationship based context aware role based access control approach for mobile users in enterprise environment. It considers the relationship between employees of a company as contextual information. The access control design uses NFC technology in mobile devices. In real world, employees have different kinds of relationships with each other for cooperative work to perform organization's task. For example a manager or supervisor may want to share his privilege to his employee to do some work on behalf of him temporarily. The manager can delegate the employee using NFC technology as long as the employee is around the office.

2.3.5. RBACA

This is probably the closest existing work to our approach. They proposed RBAC approach for Android mobile systems in order to mitigate the security risks caused by over-privileged applications. In this system, similar to our system, roles are assigned to applications and roles contain a subset of android permission. The main difference between our system's design and RBACA system design is the way context is handled. In RBACA system, context is associated with roles. Application's roles are switched manually or dynamically depending on some contexts i.e. i.e. applications will have different roles at different conditions. In CA-ARBAC system, roles assigned to applications stay the same and do not change. Context is applied on permissions i.e. permissions are turned on for the role they belong only when the associated context is fulfilled. Figure 1 and 2 below show a comparison of CA-ARBAC system and RBACA system context handling methodologies. P_1, P_2… represent permissions and C_1, C_2… represent sample contexts.

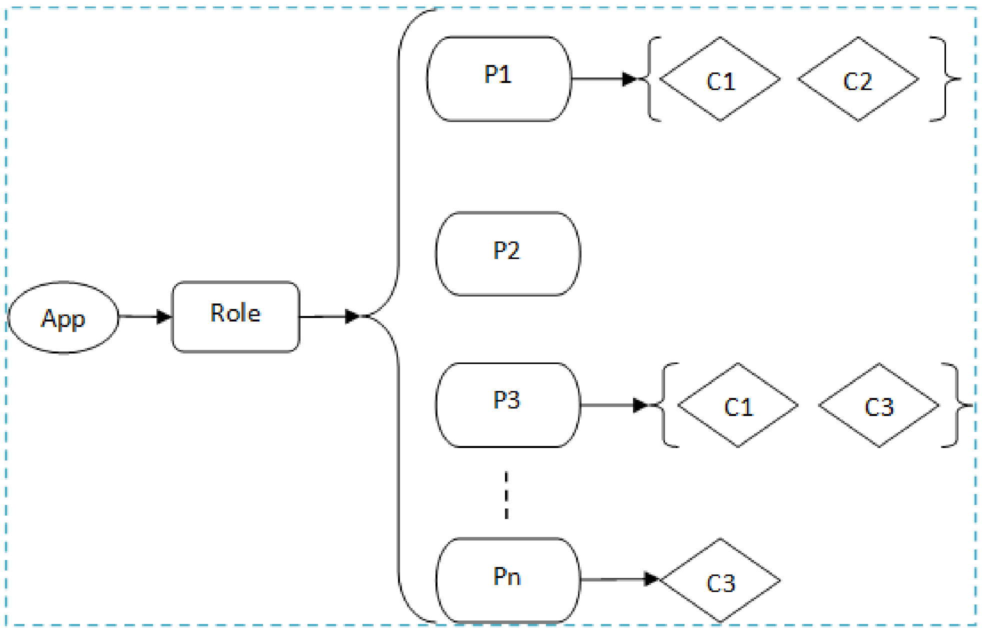


Figure 1. CA-ARBAC system way of context usage

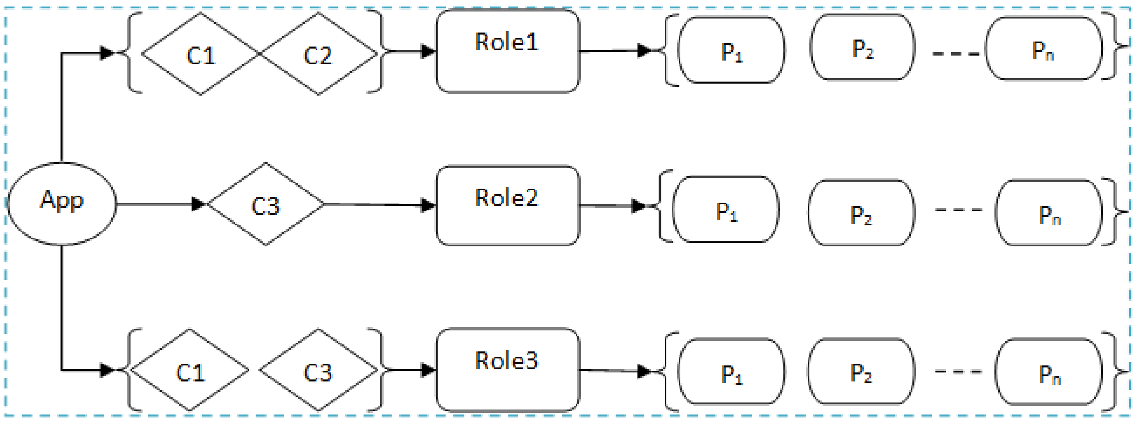


Figure 2. RBACA system way of context usage

We believe that our way of context usage has two advantages over that of RBACA system's method. First of all, allowing applications to have different roles at different contexts leads to the unnecessary creation of large number of roles. To explain this by example let's assume that a given user has installed application A which requires five permissions P_1, P_2, P_3, P_4 and P_5. Moreover, let's assume that the user wants to associate three contexts C_1, C_2 and C_3 with permissions P_1, P_3 and P_5 respectively. i.e. P_1, P_3 and P_5 are allowed for application A only when contexts C_1, C_2 and C_3 are satisfied consequently. However, P_2 and P_4 are always allowed for application A as there is no context associated with them. To satisfy the previous requirement, in our model, only one role needs to be created for application A as shown in Table 1 below.

However in the case of RBACA, three roles need to be created for application A as shown in Table 2. Application A will be assigned either role R_1, R_2 or R_3 based on the contexts C_1, C_2 and C_3.

Table 1. CA-ARBAC way of creating role for application A

Role	Permissions	Condition to use the permission
R_1	P_1	When C_1 is satisfied
	P_2	Always
	P_3	When C_2 is satisfied
	P_4	Always
	P_5	When C_3 is satisfied

Table 2. RBACA way of creating roles for application A

Role assigned to application A	Role	Permissions inside the role
At context C_1	R_1	P_1
		P_2
		P_4
At context C_2	R_2	P_2
		P_3
		P_4
At context C_3	R_3	P_2
		P_4
		P_5

Secondly, being able to associate contexts with permissions rather than associating contexts with roles allows a more flexible and finer-grained context policy configuration i.e. users will have the ability to set contexts at permission level.

The other important difference between our access control model and that of RBACA is that in RBACA, each application should have at least one default role. In our system, applications may not be given a role at all which reduces the burden on users. In addition to this, RBACA didn’t provide any architecture and implementation. We have designed and implemented a new architecture for our CA-ARBAC system as you will see in the subsequent sections.

2.4. Summary of Related Works

The summary of related works is presented in table 3 below.

Table 3. Summary of Related Works

Name	Type of Access Control Model			Description
	RBAC	CAAC	Hybrid	
MPDROID	✓			- RBAC at the application frame work layer - Assign roles to applications - MAC at the kernel layer - Enhances APS with RBAC
Bilal Shebaro Et Al. [12]		✓		- Associates Android permissions with context - Works for only two kinds of contexts
ConUcon		✓		- A general context-aware usage control model that can be used in different mobile platforms - System wide policy; does not allow per application policy configurations - Supports active context management
CRêPE		✓		- Fine-grained context- related policies for Android in a system-wide manner - Remote context configuration is possible via SMS, MMS, Bluetooth, or QR-code. - Supports active context management
ConXsense		✓		- Context aware access control for mobile devices that does not require users to configure context policies - It is uses a probabilistic approach that uses machine learning and context sensing to classify user context according to their risks - For protection against device misuse using a dynamic device lock and protection against sensory malware
DR BACA			✓	- Multi user management on Android mobile devices - Associates users with roles and roles with rules - Rules are either applied to applications to control user’s execution of applications or to permissions to grant/deny permission requests to applications
CtRBAC			✓	- RBAC for multiple users on Android phone - Permissions change based on context
CA-RBAC			✓	- Dynamic RBAC based on contextual information for users in ubiquitous computing environments. - Both UA and PA are performed dynamically
Kangsoo Jung Et Al [22]			✓	- Context aware RBAC based on user relationship to promote cooperation between employees in enterprise environment - Employees can share privileges based on contexts
RBACA			✓	- Similar to our system except that context is associated with roles and not permissions - Applications should have at least one default role - No architecture and implementation

CHAPTER 3

ANDROID BACKGROUND

3.1. Android System Overview

Android is a complete software stack consisting of different layers. It is based on the Linux kernel. It is developed by the Open Handset Alliance (OHA), which is led by Google. Android system is made up of four layers, each layer manifesting well-defined behavior and providing specific services to the layer above it as shown below in figure 3. A more detailed figure showing the components inside each layer is also shown in figure 4.

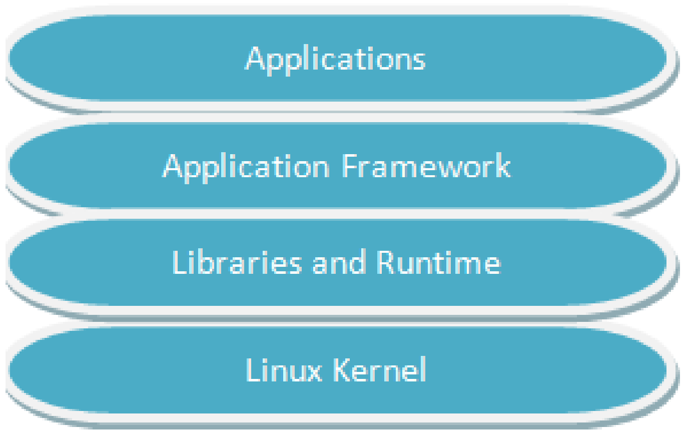


Figure 3. High level Picture of Android Architecture Layers

3.1.1. Android Kernel

The Android Kernel is the first layer of Android system that interacts with the device hardware. This is the layer that acts as a bridge that connects the device hardware and the Android software layers above the kernel. Android Kernel is a modification of the traditional Linux Kernel for an embedded environment. Android Kernel has also made many enhancements to the original Linux Kernel. Android kernel takes care of duties such as process management, memory management, device drivers, networking, power and security.

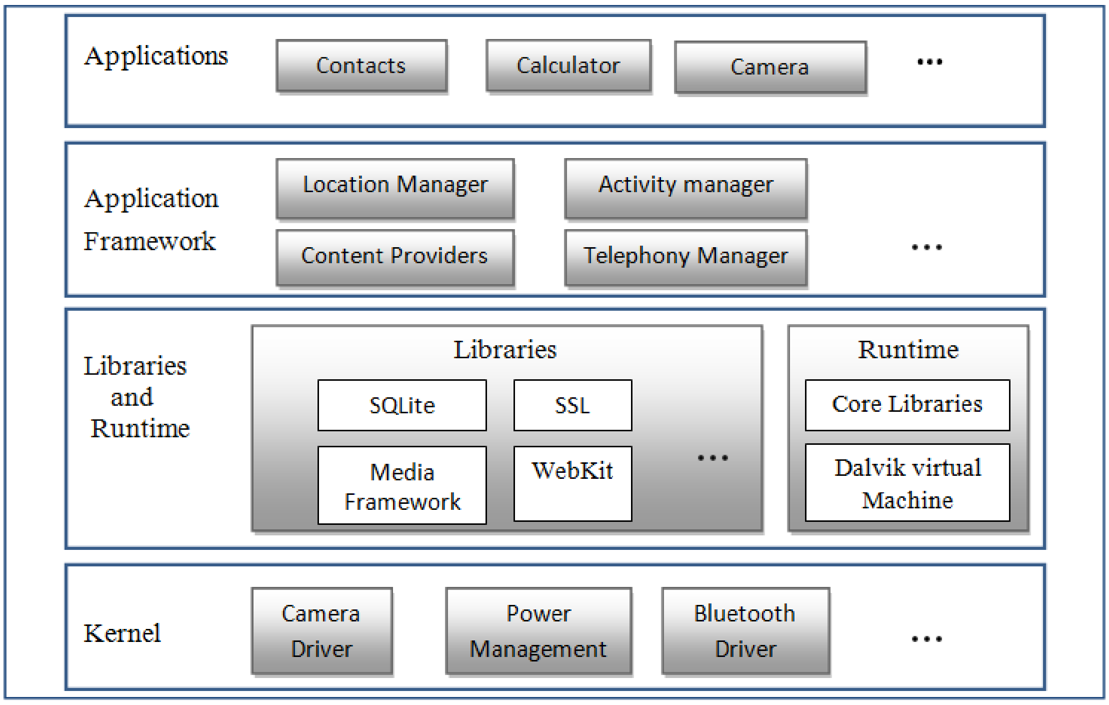


Figure 4. More Detailed Android Architecture Layers

3.1.2. Libraries

The libraries component consists of a set of C and C++ libraries used by different components of the Android system. This layer is also called the "native layer" because of the fact that the libraries are written in C and C++ and optimized for the hardware, as opposed to the Android applications and framework, which are written in Java. It acts as a translation layer between the kernel and the application framework. Developers use these libraries through the Android application framework. Android applications can access native capabilities through Java Native Interface (JNI) calls.

3.1.3. Android Runtime

The Android Runtime has two parts: the Dalvik Virtual Machine (DVM) and Java Core Libraries. The DVM executes java class files compiled into .dex format. It is analogous to the Java Virtual Machine (JVM) that exists on personal computers and servers today. In Android, every application runs in an isolated process inside a separate Dalvik virtual machine instance allocated for that application. The Dalvik VM relies on the Linux kernel for providing lower level functionality (e.g., memory management). Android relies on Java core Libraries for most of the services related to Java programming language. Java core libraries depend on the service they get from the kernel and Dalvik VM.

3.1.4. Application Framework

The application framework provides a collection of services or systems for the developer to be used when writing applications. The presence of the application framework simplifies the reuse of components. The services in the application framework publish interfaces for different functions so that any other third party applications can then use those functions without the need to reinvent the wheel.

3.1.5. Applications

The application layer of the Android operating system is the closest to the end user. By default, Android comes with rich set of applications, including the browser, the SMS program, the calendar, the e-mail client, maps, Contact Manager, an audio player, and so forth. User applications also belong to this layer.

3.2. Android System Security

Android is a truly open mobile platform. To secure an open platform, we require a robust security architecture. Android is considered as one of the most secure and flexible operating system for mobile platforms. Android security mechanism is a multi-layered security system that is designed by imitating traditional Linux system security approaches. Hence it provides similar security services such as protecting user data, system resources and providing application isolation. It is designed with both developers and device users in mind. Android achieves these security objectives based on the following key security features provided by the Linux kernel:

1. Mandatory application sandbox (process isolation)
2. Secure inter process communication mechanism
3. User-based Permission model

4. Cryptography
5. Mandatory access control using Security-Enhanced Linux in Android(SEAndroid)

3.2.1. Mandatory Application Sandbox

Android makes use of one of the key security features (user-based security model) provided by Linux kernel to provide mandatory application sandbox. The user-based security model allows application resources to be identified and kept isolated. The Application Sandbox is performed in the Linux Kernel. Since the Application Sandbox is implemented in the lowest level of the Android software stack (in the kernel), this security model is also applicable to native code applications and to operating system applications. The application sandboxing encompasses all user applications, the application framework, the application runtime and operating system libraries. With the exception of the kernel and some operating system code running as root, all of the other components run within the application sandbox. Moreover, each component above considers that the parts beneath it are trusted and properly secured. A given application or component is sandboxed by executing it in separate process with a unique user ID (UID). Each application is assigned its own set of private data structures and is prevented from interfering with other processes' execution or from performing sensitive operations such as accessing the recording audio, making phone calls, or receiving SMS messages. The private data structures are also labeled the application's UID.

3.2.2. Secure Inter Process Communication

In addition to any Linux-type Inter process communication methods such as local sockets, file system and signals, Android provides different kinds of new IPC mechanisms such as Binders, Services, Content Providers and Intents. Application sandbox can talk to other applications via such kinds of secure IPC mechanisms.

Binder: Binder is a type of remote procedure call (RPC) mechanism in Android. It allows communication between processes in the same application and also between processes in different applications. Traditional Linux driver is used to implement it.

Services: Services are one of the Android application components that run in the background to accomplish long lasting tasks. Services also expose RPC communication interfaces that are reachable through binders.

Intent: Intent is a lightweight messaging object used by applications to tell other applications that they want to do something. In other words, it is a notification message from one process to another to accomplish some work.

Content Providers: A Content Provider is a way of having access to data stored on the device. Applications get access to data exposed by other applications through Content Providers. An application can also define its own Content Providers to expose its data to other applications.

Although it is possible to implement inter process communication using Linux IPC mechanisms such as network sockets and world-writable files, the new Android IPC mechanisms are the recommended ones.

3.2.3. Application Signing

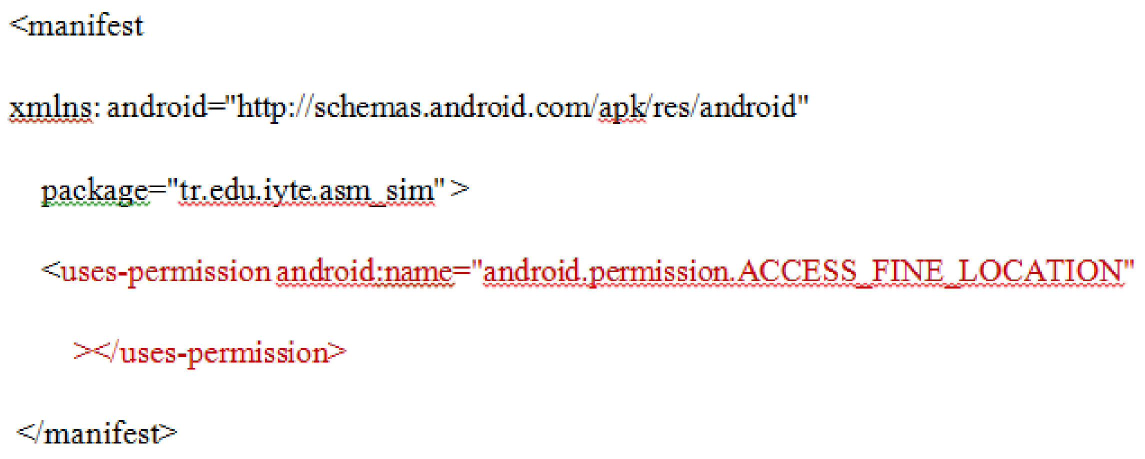
All Android applications running on Android system must be signed by the developer before they can be installed. An android system refuses installation of applications that are not signed. The main purpose of application signing is to distinguish applications from one to another. Developers sign their applications with their own private keys (self-signed). Even though it is allowed, it is not a requirement to sign Android applications with a certificate authority which also shows that no applications are trusted. Android system will not be able to place an application in an Application Sandbox unless the application is properly signed. The Android system uses the signed certificate to determine which application is represented by which user id. With application signing, since different applications run under different user IDs, one application cannot access private data of other applications except through one of the secure IPC mechanisms discussed earlier.

When an Android application is going to be installed, the new application can choose to share a UID with other applications already installed on the device. Applications ask to share UID with other applications by specifying it in their manifest file. In such cases, the Android system checks if the public key certificate of the requesting applications matches the key used to sign the other application installed on the device. If the two keys are the same, the two applications are then treated as being the same application regarding security issues. Besides this, Applications signed with the same key can also have different Application Sandboxes and UIDs with the option to share security permissions at the signature protection level. Therefore, Android system also uses Application signing to allow and or deny access to signature protection level permissions.

3.2.4. Android Permission System

As mentioned earlier, all user applications installed on Android system run in their own sandbox. Hence, by default, an application is only allowed to access data in its sandbox (its own private data) and a limited range of less sensitive resources such as internet. The permissions associated with such kind of less sensitive resources are automatically granted at install time and the user will not be able to revoke them later. These permissions are called Normal Permissions.

For all other resources and services, application’s access to resources is performed through a strictly controlled access procedure which is managed by Android system security. The access controls are implemented in two ways. Some functions are totally not allowed to be used by user applications. For example, there is no way to access the SIM card directly. In other cases, the sensitive APIs are allowed for trusted applications and are protected through a security mechanism known as Permissions. Some out of the many protected APIs include: Telephony functions, SMS/MMS functions, Camera functions, Location data (GPS), Bluetooth functions and Network/data connections. To be able to access such kinds of security and privacy sensitive resources other than their own private data directories, applications need to be granted different application layer permissions by the user. Applications have to explicitly request the permissions they need in order to use those permissions and execute successfully. They do so by declaring the required permissions in the manifest file as shown in Figure 5.



```
<manifest
xmlns: android="http://schemas.android.com/apk/res/android"
   package="tr.edu.iyte.asm_sim" >
   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
      ></uses-permission>
 </manifest>
```

Figure 5. Sample Manifest file showing permission declaration

User approval is required before an application can get access to critical operations (e.g., making calls, sending SMS messages). Starting from Android version 6, user approval is done during runtime. When an application is going to perform some operation that requires a given permission (more accurately group of permissions), Android prompts the user to either allow or reject the requested permission/permissions. The left side of Figure 6 shows a screen shot of permission user approval in Android. Furthermore, the user can later modify permissions (revoke previously allowed permissions or allow new permissions). The right side of the same figure (figure 6) shows runtime permission modification.

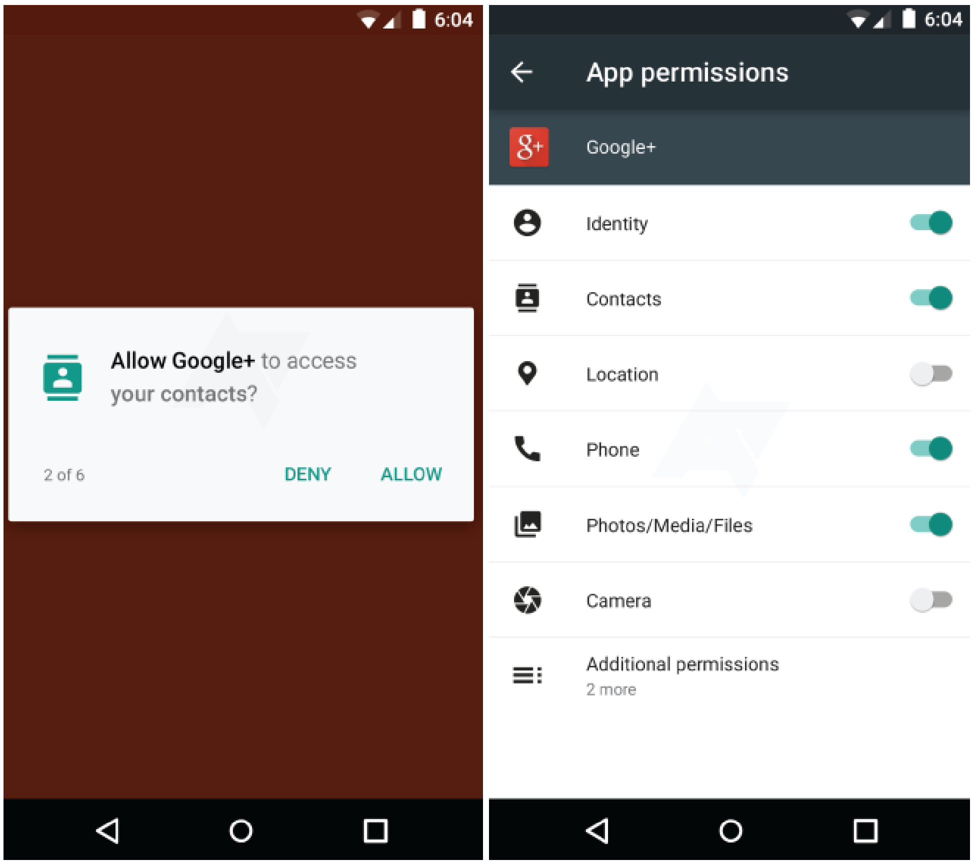


Figure 6. Android User Permission Approvals

Once the user has granted permission, the permissions will be applied to the application as long as the application is not uninstalled. Permissions granted to an application are removed if the application is uninstalled. Therefore, re-installation of the application will result in requesting all of the permissions again. If the application attempts to use a sensitive resource or service without declaring the necessary permission in its manifest, a security exception will be thrown back to the application.

3.2.4.1. Permissions

Android defines a set of core permissions for protecting OS resources and services. These are called system-built-in permissions. Android defines around 140 system-built-in permissions. All of the system-built-in permissions in Android are listed in Android developers' website [27]. Some examples of system-built permissions are CALL_PHONE, INTERNET, CAMERA, READ_CONTACTS, READ_LOGS, READ_SMS, RECEIVE_SMS, SEND_SMS, WRITE_SMS and so forth. System-built permissions provide a means to get access to restricted content and APIs.

Third-party application developers can also define new permissions that are enforced using the same mechanisms as system-built permissions. We call these later ones as user-defined permissions or custom permissions. User-defined permissions are used by third-party application developers to protect their applications/components from other applications. However, some device capabilities are not available to developers such as the ability to send SMS broadcast intents because these permissions are signature level permissions. The main concern of this paper is about Android system-built permissions and not about user-defined permissions. Therefore, from now on, we will simply use the word permission to refer to Android system-built permissions.

3.2.4.2. Permission Protection Levels

Android classifies permissions into four levels depending on the potential risk that may come from granting a given permission. This classification helps the system in determining whether or not to grant the permission to an application requesting. The four protection levels are described below.

Normal Permissions: Normal permissions are permissions that are considered to have relatively minimal risk to the system, other applications or the user. Permissions such as ACCESS_NETWORK_STATE, CHANGE_NETWORK_STATE, INTERNET and SET_TIME_ZONE are categorized under normal permissions. Appendix A shows the list of all normal permissions in Android. Applications require normal permissions to access data or resources outside their sandbox/private data. Permissions with normal protection level are automatically granted to a requesting application at installation time without explicitly asking approval from the user. For example, if an application declares in its manifest file that it needs INERNET, Android system automatically grants INERNET permission to the application at installation time.

Dangerous Permissions: These are risky permissions that could potentially harm the operation of other applications or user's data if granted. Examples of dangerous permissions are CONTACTS, LOCATION, PHONE and SMS. Android system does not automatically grant dangerous permissions to a requesting application because it could introduce a negative impact on the private user information or on the device. To get access to dangerous permissions, applications should declare in their manifest file that they need the permission and the user has to explicitly grant the permission.

Dangerous permissions requested by an application will be displayed to the user during run time for confirmation before allowing the application to use it. The exhaustive list of dangerous permissions is listed in Appendix B.

Signature: As explained previously, Android system grants permissions declared by one application to another application if the two applications are signed with the same certificate. Permissions achieved this way are called signature permissions. Android system automatically grants the permission without asking user's explicit approval if the certificates of the two applications match.

SignatureOrSystem: Similar to the Signature protection level, applications that are signed with the same certificate as the application that declared the permission are automatically granted the permission. The difference between Signature and SignatureOrSystem protection levels is that SignatureOrSystem can also be requested by an application that came with the Android system image. Hence, SignatureOrSystem protection level permissions can be granted to Android operating system applications even though they are not signed with the same certificate as the application declaring the permission. The "SignatureOrSystem" permission is used for certain special situations where multiple vendors have applications built into a system image and need to share specific features explicitly because they are being built together.

In addition to the above four divisions, there are also some special permissions in Android that are very sensitive and are not allowed for most applications. SYSTEM_ALERT_WINDOW and WRITE_SETTINGS are two examples of special permissions. If an application wants to use one of the special permissions, it should first declare the permission in the manifest. In addition, it should create an intent that is used to request confirmation from the user. The Android system displays authorization screen to the user for approval.

3.2.4.3. Permission Grouping

Android system permissions are grouped into related permissions. For example, the following four permissions are grouped under the “PHONE” permission group:-

-Directly call phone numbers; this may cost you money

-Write call log (example: call history)

-Read call log

-Reroute outgoing calls

-Modify phone state

-Make calls without your intervention

The list of Android permission groups is listed in appendix C. Starting from Android version 6.0 (API level 23), permission requests are presented to the user in the form of permission groups and not as a single permission. As previously stated, Android system automatically grants Normal permissions to a requesting application. However, a dangerous permission must be approved by the user before granted. When an application requests a dangerous permission that belongs to some permission group that is declared in its manifest file, Android system performs one of these two things:-

If the application had previously been granted one or more other permissions in that permission group, the system automatically grants the permission without informing the user. For instance, let say an application which had previously requested and been allowed WRITE_CALENDAR permission requests READ_CALENDAR permission, then the system immediately grants the READ_CALENDAR permission.

However, if the application does not have any prior granted permission in that permission group, the system displays a dialog box to the user for approval. The authentication window shows a description of the permission group that the requested permission belongs but does not show a description of the specific requested permission within that group. As an example, if an application requests the READ_CALENDAR permission, the system displays an authentication window saying the application needs access to your CALENDAR. The system grants the permission only if the user accepts it.

3.2.4.4. Permission Enforcement

As stated previously, in Android, every time the user installs application, a unique user ID is generated for the application and the application is assigned that UID. The application runs under that UID for the whole of its life. In addition, all data stored by that application is assigned that same UID, whether a file, database, or other resource. Every application sandbox accesses its own private resources by direct system call to the kernel. The Linux Kernel enforces private resource access by comparing the UID of the requesting application with the UID of the requested resources. The Linux

Kernel permissions on private resources for that application are set to allow full permission by default. However, an application needs permission from the user to access resources other than its private resources.

An application can access resources other than its private directory using two different ways. Firstly, when an application is granted less sensitive public resources such as SDCard and CAMERA permissions, it is added to a Linux group that has access to the corresponding resources. Thus, the application is assigned a group ID (GID) in addition to the UID. Such kinds of public resources are also accessed by directly interacting with the underlying kernel through system calls in a similar fashion to private resource access. The Linux Kernel enforces the access control policy i.e. the access control in the file system ensures that the application has the necessary permissions. E.g. it checks whether the application is allowed to open a file on the CAMERA by checking the GID of the application with the GID that is privileged to access the CAMERA. The file system access control uses traditional Linux Discretionary Access Control. The Linux Kernel access control also supports a Mandatory Access Control (MAC) scheme called Security Enhanced Android (SEAndroid) starting from Android v4.3. SEAndroid is Security Enhanced Linux (SELinux) enabled for Android.

Secondly, to access highly sensitive public resources such as SMS, PHONE and CONTACTS, applications should use the Middleware Layer Android system API in a strictly controlled way. Applications are not allowed to access highly privileged resources by direct system call to the Kernel. Such kinds of resources are accessed through Middleware layer system services and applications that implement the target API. For example, the Location Service provides the API used to communicate with the GPS or other location providers. Therefore, if an application wants to get users' location, it communicates with the location service instead of directly interacting with the GPS or other location providers. Permission check is also performed by system services/applications at the Middleware Layer. The system services/applications use Android Permission Validation Mechanism to check whether the caller application with the given UID has the necessary permission or not. The system service/application gets the UID of the caller application from the Binder IPC. Fig. 7 elaborates Android application resource access procedure.

The fact that in Android, applications are uniquely identified by their UIDs makes assigning roles to applications possible. In our CA-ARBAC system also applications are identified by their UIDs. Our system assigns roles to applications based on their UIDs. Therefore, permission check is also performed by UID during resource access.

Figure 7 below elaborates Android application sandbox and resource access procedure. The application APP_1 has UID of 1. It also has two groups IDs (2 and 3). This shows that the application has access to two public resources with GID of 2 and 3 (BLUETOOTH and CAMERA for our example) and can access them by direct system call to kernel. However, APP_1 cannot access highly privileged resources directly. It has to go through a system service API at the Middleware Layer.

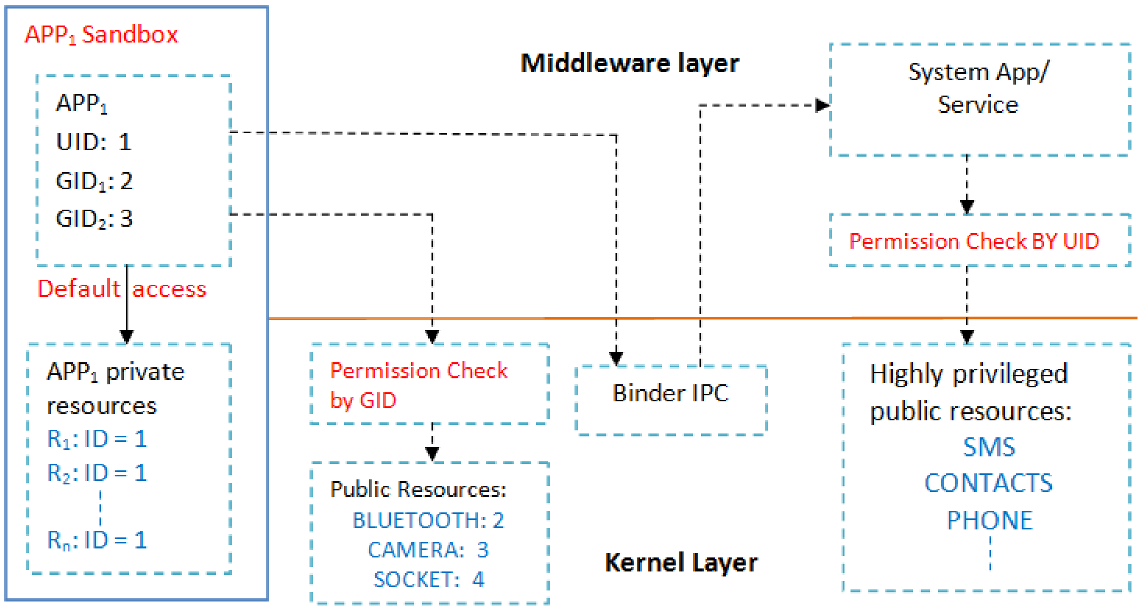
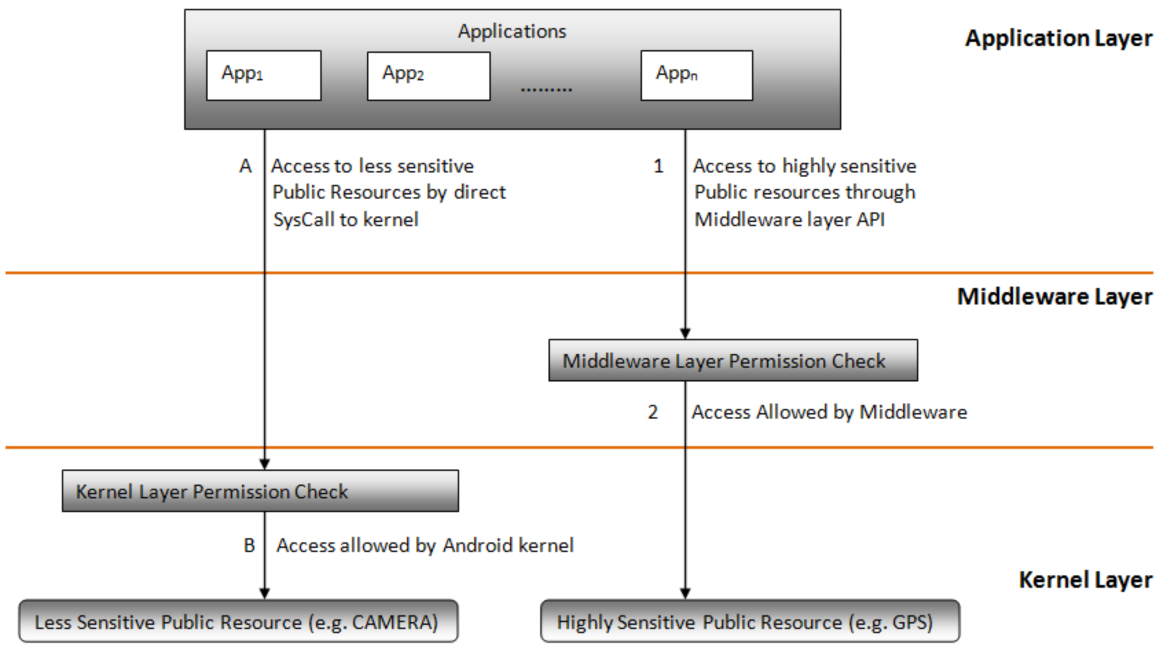


Figure 7. Android application resource access procedure

In our CA-ARBAC system design, we used a simplified version of the above Android resource access procedure shown in Figure 8 not to show much details and make the design more complex.


Applications
App1
App2
....
Appn
Application Layer
A
Access to less sensitive Public Resources by direct SysCall to kernel
1
Access to highly sensitive Public resources through Middleware layer API
Middleware Layer
Middleware Layer Permission Check
2
Access Allowed by Middleware
Kernel Layer Permission Check
B
Access allowed by Android kernel
Kernel Layer
Less Sensitive Public Resource (e.g. CAMERA)
Highly Sensitive Public Resource (e.g. GPS)
Figure 8. Simplified Android application resource access procedure

CHAPTER 4

CA-ARBAC DESIGN

4.1. CA-ARBAC Access Control Model

CA-ARBAC is an adaptation of the traditional RBAC model. The three main components of the traditional RBAC are users, roles and permissions. In CA-ARBAC, applications replace users. Applications are considered as users. In a traditional RBAC model, user assignment (UA) and permission assignment (PA) are handled by administrators. In case of CA-ARBAC, smart phone users are expected to perform application assignment (AA) and permission assignment (PA). Application assignment is assigning applications to roles and permission assignment is assigning permissions to roles. In addition to these, our model contains a fourth component: context. In CA-ARBAC, users are also expected to configure contexts associated with permissions (if any). If permission is associated with one or more contexts, it will not be allowed for the application that owns it unless the contexts are satisfied. Context checking and granting is done by the system dynamically during resource access.

In the following sections, we describe CA-ARBAC access control policy model. Here is the definition of the basic system components:-

Definition 1 (Applications): An application is any user Application in the system. Let A represent the set of all user applications installed on the device.

Definition 2 (Roles): In CA-ARBAC, a role is a functional category of Android applications which consists of a set of permissions. Let R stand for the set of roles in the system.

Definition 3 (Permissions): The permissions in our system are any one of the permissions defined in Android system. Let P represent the set of all permissions in Android system.

4.1.1. Application Assignment (AA)

CA-ARBAC policy is designed in such a way that applications are assigned roles and roles contain a set of permissions. Application assignment (AA) is a mapping that associates an application with an assigned role.

Definition 4 (Application Role Mapping): Let ARM be the list containing the mapping between applications and roles.

The elements of ARM are duplets:

$<A_i, R_j>$ where $A_i \in A$ and $R_j \in R$.

The user manually creates roles and assigns it to one or more applications. When the user assigns roles to applications, it is added to the ARM. The Application Role Mapping is static and do not change dynamically based on contextual data. A many-to-many mapping (application-to-role assignment relation) exists between applications and roles:

$ARM \subseteq A \times R$.

Figure 9 shows application role assignment relation. As we can see from the figure, a role can be assigned to multiple applications at the same time. Similarly, an application can also have more than one role simultaneously.

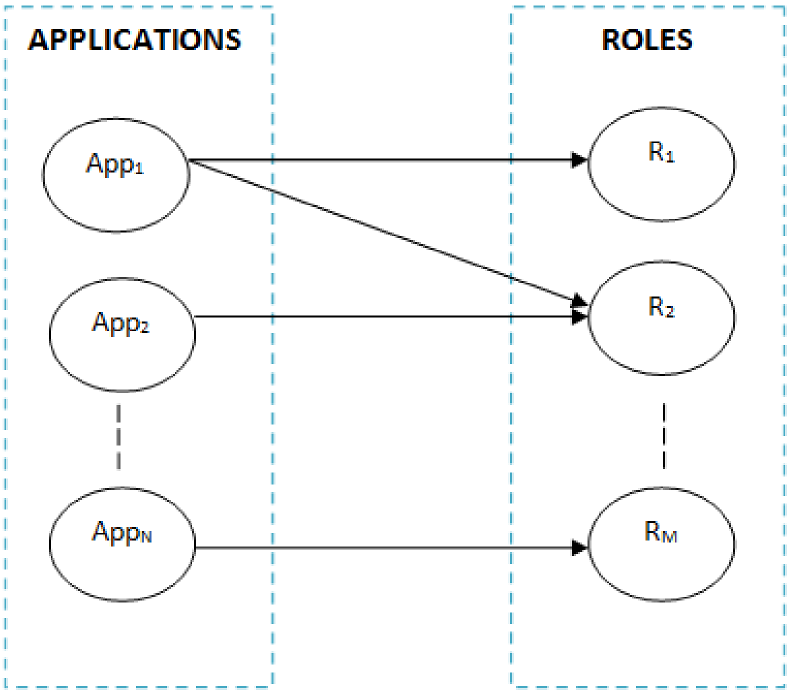


Figure 9. Application-to-role assignment relation

4.1.2. Permission Assignment (PA)

Permission assignment (PA) is a mapping that associates roles with an assigned permission. A role can be assigned multiple permissions and a single permission can also occur in many roles. Permission assignment can be classified into two based on the presence and absence of context: Static Permission Assignment and Dynamic Permission Assignment. When there is context data associated with permissions, the permission assignment is called Dynamic Permission Assignment. Otherwise, it is called Static Permission Assignment.

4.1.2.1. Static Permission Assignment without Context (PA)

In the absence of context associated with permissions, the permission set assigned to roles stays active for the role all the time i.e. all the permissions assigned to a role are allowed for the role all the time.

Definition 5 (Role Permission Mapping): Let RPM be the list containing the mapping between roles and permissions. The elements of RPM are duplets:

$<R_m, P_k>$ where $R_m \in R$ and $P_k \in P$.

A many-to-many mapping (role-to-permission assignment relation) exists between roles and permissions,

$RPM \subseteq P \times R$.

Role permission assignment relation is shown pictorially in Figure 10. As we can see from the figure, a role can be assigned to multiple applications at the same time. Similarly, an application can also have more than one role simultaneously

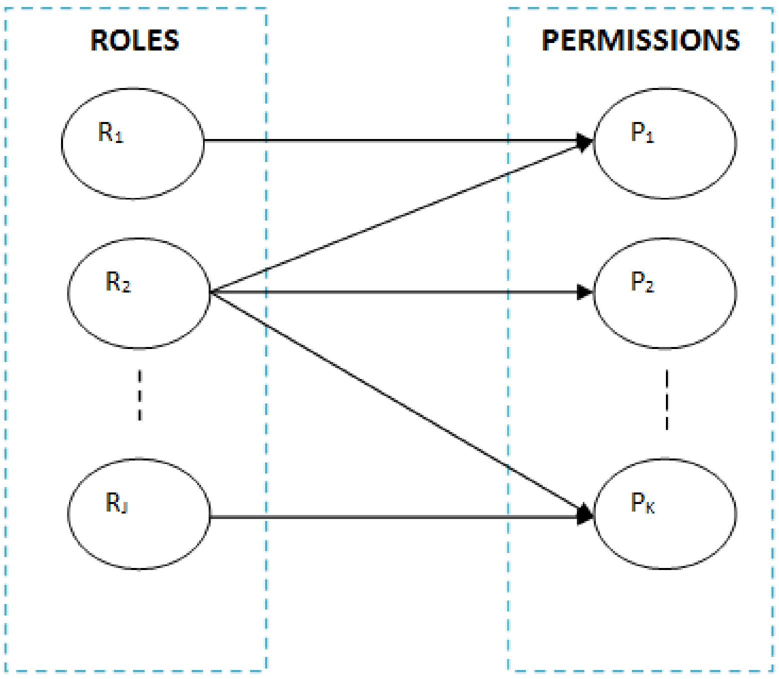


Figure 10. Role-to-permission assignment relation without context

4.1.2.2. Dynamic Permission Assignment in the Presence of Context

In CA-ARBAC, the usage of permissions inside a given role can be restricted by specifying the conditions under which the permission should or should not be allowed.

In this paper, we use two kinds of context sources: Environmental context and system context. Location of the user and surrounding temperature are types of environmental contexts. Some examples of system context include: Time, battery status, whether there is an ongoing phone call or not, whether the screen is locked or not etc.

Definition 6 (Context): Many kinds of contexts can be applied in our model. Each context is identified by its name and one or more attributes.

$$\text{Context} = <\text{ContextName}, \text{ContextAttributes}> \qquad (4.1)$$

For example, LOCATION is a context that is identified by two attributes: latitude and longitude; TIME is a context identified by single attribute: time of day.

Definition 7 (Context Policy): A context policy is a rule that specifies the condition under which a given permission should be allowed or not allowed. It consists of two parts: the context description and the action to take. ContextDescription is expressed as follows:

$$\text{ContextDescription}=[\text{ContextName},\text{Operator}, \text{AttributeValues}] \qquad (4.2)$$

The operator represents different kinds of key words used for comparison. It includes: EqualTo, GreaterThan, LessThan, GreaterThanOREqualTo, LessThanOREqualTo, INBetween and IN. AttributeValues is the set of values for each of the context attribute of the given context.

Let CD be a set of context descriptions and CP be the set of context policy rules configured in the system, then

$$CP_i = <CD_i, \text{Action}> \text{ where } CP_i \in CP \text{ and } CD_i \in CD \qquad (4.3)$$

When we configure context policy, we may need to specify multiple values for the context based on the range we want to include. For example, we may specify that some permission should be denied access from some starting time to some end time. Another case is we may specify that a given permission can be allowed on week days. The action attribute indicates the action to be taken when the context is satisfied. It is either allow or deny.

Definition 8 (Context Combination): Often context policies are made up of a combination of contexts combined together using logical conjunction operator. Let CCP (combined context policy) be the set of context policies containing combination of context descriptions, then

$$CCP_i = < (CD_1 \wedge CD_2 \wedge \dots \wedge CD_m), \text{Action}> \text{ where}$$
$$CCP_i \in CCP \text{ and } CD_i \in CD. \qquad (4.4)$$

The value of CCP is either true or false based on the value of the Action attribute and the return value of the combination of the context descriptions. When the Action attribute is set to allow, it returns true if all the contexts in the combination are satisfied. Otherwise, it returns false. When the Action attribute is set to deny, it returns false if all the contexts in the combination are satisfied. Otherwise, it returns true.

Definition 9 (Context List): Sometimes permissions are also associated with a list of combined contexts joined together using logical disjunction operator. Let CPL be the set of context policy lists, $CPL_i \in CPL$ where CPL_i is the context policy list associated with permission P_i.

$$CPL_i = < (CD_1 \lor (CD_2 \land CD_3) \lor CD_4 \ldots \lor CD_n), Action>$$
$$\text{where } CCP_i \in CCP. \qquad (4.5)$$

In this case, when the Action is set to allow, CPL returns false if all of the CDs in the list return false. Otherwise, it returns true. When the Action is set to deny, CPL returns true if all the CDs in the list return false. Otherwise, it returns false.

Definition 10 (Active Permissions): Not all the permissions assigned to a role are active for the role all the time. An application can only use active permissions. Whether permission is active or not for a given role is determined by the list of contexts associated with the permission. Active permissions are permissions for which the associated context is satisfied.

Figure 11 below illustrates our access control model graphically. In the figure, APP_x is assigned the role R_x. Role R_x is granted j permissions. Permission P_1 has no any context associated with it which means that it will be active for role R_x at all times. Permissions P_2, P_3 and P_J of role R_x on the other hand have contexts associated with them. P_2 is associated with single context. P_3 is associated with combination of two contexts and P_J is associated with a context list which contains two combined contexts. APP_x can access these permissions only if the contexts associated with them are satisfied.

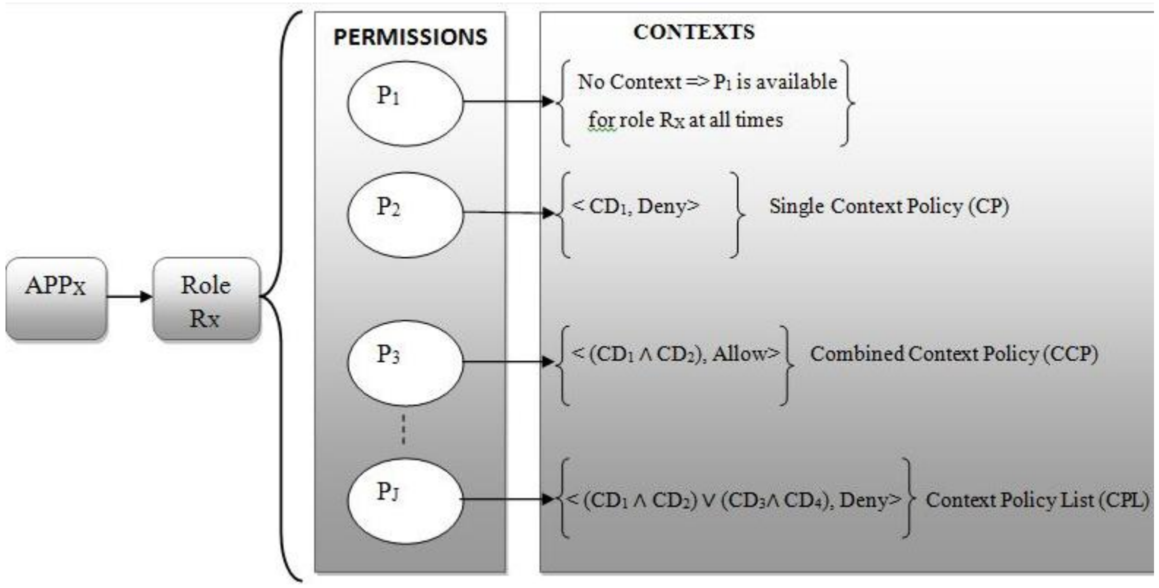

PERMISSIONS
CONTEXTS
APPx
Role Rx
P1
P2
P3
PJ
No Context => P1 is available for role Rx at all times
< CD1, Deny>
Single Context Policy (CP)
< (CD1 ∧ CD2), Allow>
Combined Context Policy (CCP)
< (CD1 ∧ CD2) ∨ (CD3 ∧ CD4), Deny>
Context Policy List (CPL)

Figure 11. Role-to-permission assignment relation with context

4.2. CA-ARBAC Architecture

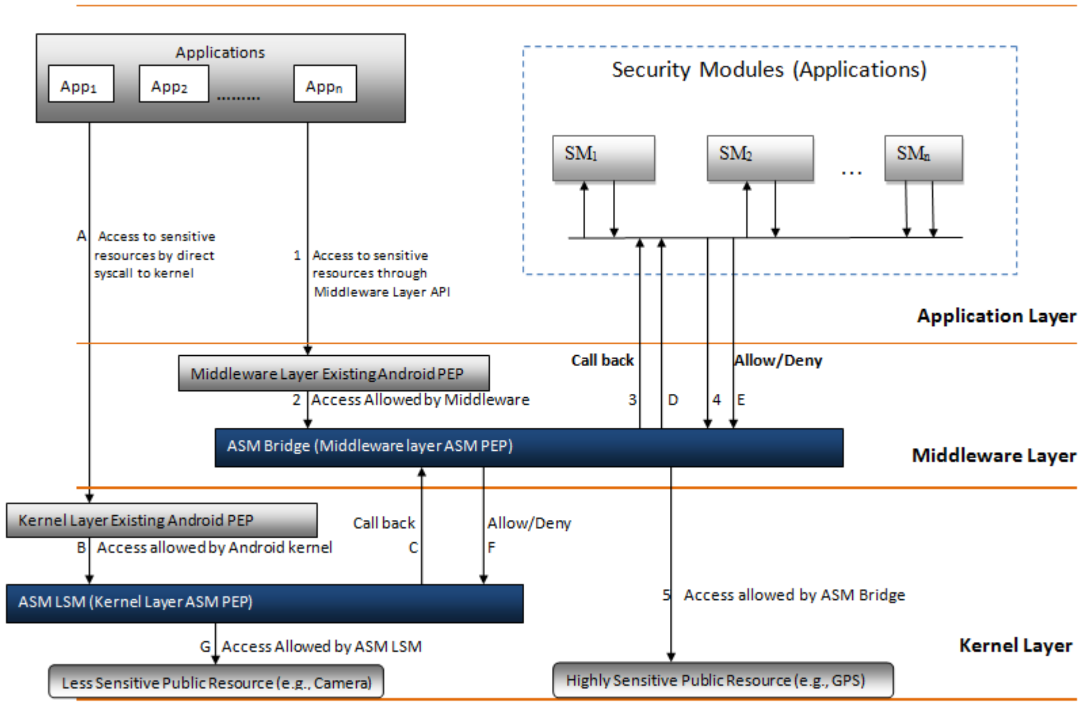
Because of the absence of comprehensive security API for the development and modularization of security extensions on Android, all of the earlier Android security system enhancements required modification to the android operating system. Consequently, these previous works are provided in one of two ways. Some of them are presented as separate model-specific patches to the Android software stack. Others are embedded into Android's mainline codebase and become integrated component of the Android OS design. Both of these lessen the effectiveness of the practical and theoretical aspects of security solutions. As noted by Android Security Framework (ASF) [28], first, there is in general no consensus on the *"right"* security model, as demonstrated by the broad range of Android security extensions. Thus, OS security mechanisms should not limit policy authors to one specific security model by embedding it into the OS design. Second, providing security solutions as *"security-model-specific Android forks"* impedes their maintainability across different OS versions, because every update to the Android software stack has to be re-evaluated for and applied to each fork separately. Apart from this, rebuilding the operating system to patch the security applications is very difficult for the majority of the user unless the user is highly skilled or device manufacturers include it in the operating system during device production.

Understanding this gap, ASF [28] and ASM [8] recently developed an extensible security framework for Android that provides a programmable interface for the development and integration of different kinds of security models in the form of code based security modules. ASF and ASM are two independent but similar systems developed by independent researcher groups. Both of these groups are dealing with Google to integrate their work to android operating system mainline codebase.

Taking the above two unsatisfactory situations into consideration, we decided to design CA-ARBAC as independent code based security module that does not require modification to the android operating system. Hence, we designed CA-ARBAC as a pure java application built on the application layer based on ASM. Before we explain about our design it will be helpful to have a small introduction about ASM.

4.2.1. ASM

The motivation behind the development of ASM is to provide a programmable interface that will enable security application developers to extend Android security without the need to change the operating system. Existing Android security enhancements define their own specific hooks in different ways. Such kinds of hooks only support the specific model they are designed for. They cannot provide general and complete support for others who wish to implement a different logic. To solve this problem, ASM provides a reference monitor interface for building new reference monitors (Security applications/Security modules). This allows reference monitor developers to focus on their novel security models and not worry about enforcement hooks. The reference monitors are called ASM apps and they are developed just like any other conventional Android applications. ASM apps implement the security logic. They use ASM hooks for policy enforcement. This is possible through registration for authorization hooks. Each ASM app registers for a unique set of hooks and will receive a callback from ASM when a sensitive resource is going to be accessed. The ASM on the other hand automatically invokes the callback in the ASM app. The part of ASM which contains the ASM reference monitor interface is the ASM Bridge shown in Figure 12. Besides interacting with ASM apps, the ASM Bridge also receives protection events from authorization hooks distributed all over the Android OS. All authorization hooks are not involved in this communication with ASM Bridge. Only those hooks that are enabled as a result of ASM apps making registration will notify the ASM Bridge when protected resources are to be accessed. ASM also supports authorization hooks within the Linux kernel. To achieve kernel authorization, a special ASM LSM (Linux Security Modules) performs up calls to the ASM Bridge, once more only doing so for hooks explicitly enabled. ASM framework looks like that in Figure 12.



Applications
App1
App2
Appn
Security Modules (Applications)
SM1
SM2
SMn
A Access to sensitive resources by direct syscall to kernel
1 Access to sensitive resources through Middleware Layer API
Application Layer
Middleware Layer Existing Android PEP
2 Access Allowed by Middleware
Call back
Allow/Deny
3
D
4
E
ASM Bridge (Middleware layer ASM PEP)
Middleware Layer
Kernel Layer Existing Android PEP
B Access allowed by Android kernel
Call back
C
Allow/Deny
F
ASM LSM (Kernel Layer ASM PEP)
5 Access allowed by ASM Bridge
G Access Allowed by ASM LSM
Kernel Layer
Less Sensitive Public Resource (e.g., Camera)
Highly Sensitive Public Resource (e.g., GPS)

Figure 12. ASM Framework

4.2.2. Architecture Overview

The design of CA-ARBAC which is based on ASM is shown in Figure 13. On the figure, there are three kinds of components. The lower light-grey colored parts belong to existing Android system components which participate in the resource access process. The two blue colored boxes represent ASM system extensions to Android operating system. The two (A to G) and (1 to 5) numbered arrows indicate the steps followed when an application needs to access a sensitive resource in Android system that is enhanced with ASM and CA-ARBAC. The big light-grey colored part on the top right corner is our CA-ARBAC system.CA-ARBAC is an ASM app. With ASM, security module developers do not need to worry about policy enforcement hooks and concentrate only on the logic that makes policy decision. Therefore, in our case, CA-ARBAC is only responsible for implementing the policy logic. It is concerned about policy decision making, policy configuration, context detection and policy storage. Policy enforcements are handled by ASM. CA-ARBAC receives call back from ASM when a sensitive resource is going to be accessed. CA-ARBAC consists of four components. Each components of CA-ARBAC are explained in the subsequent sections as follows:

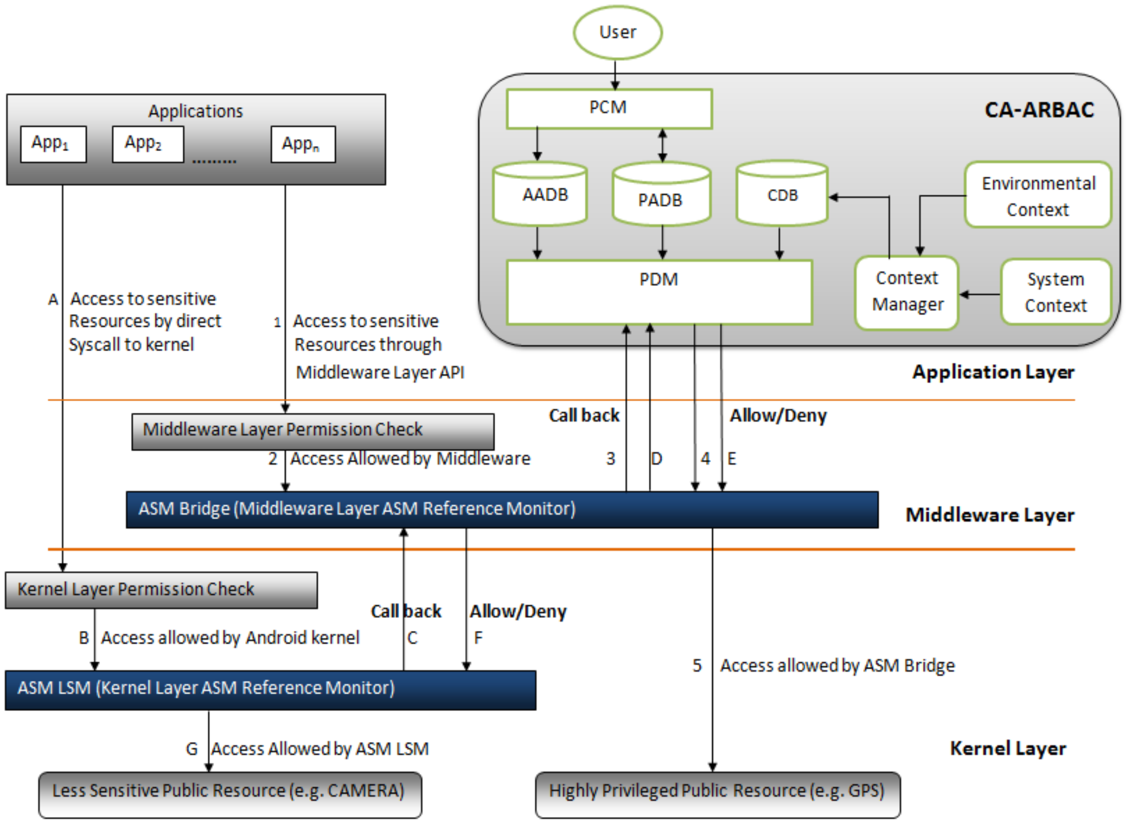

User
CA-ARBAC
PCM
AADB
PADB
CDB
Environmental Context
PDM
Context Manager
System Context
Applications
App1
App2
Appn
A Access to sensitive Resources by direct Syscall to kernel
1 Access to sensitive Resources through Middleware Layer API
Application Layer
Call back
Allow/Deny
Middleware Layer Permission Check
2 Access Allowed by Middleware
3 D 4 E
ASM Bridge (Middleware Layer ASM Reference Monitor)
Middleware Layer
Kernel Layer Permission Check
Call back
Allow/Deny
B Access allowed by Android kernel
C
F
5 Access allowed by ASM Bridge
ASM LSM (Kernel Layer ASM Reference Monitor)
G Access Allowed by ASM LSM
Kernel Layer
Less Sensitive Public Resource (e.g. CAMERA)
Highly Privileged Public Resource (e.g. GPS)

Figure 13. Architecture of CA-ARBAC based on ASM

4.2.3. Components of CA-ARBAC

Policy Decision Manager (PDM): The PDM is the core component of CA-ARBAC. It is the part that makes security decisions inside CA-ARBAC. CA-ARBAC is connected to ASM through PDM.

Policy Configuration Manager (PCM): Policies are configured by the user through the user interface component called PCM.

Context Manager: The PDM needs to get the current contextual information to make access decision i.e. it needs to check whether the pre-specified context associated with permissions is fulfilled or not. The PDM gets current context information from the Context Database (CDB). The CDB stores different kinds of context and their current values. The Context Manager is the part responsible for continuously receiving updates of contextual information from different context provider elements and updating the CDB with the new values.

CA-ARBAC Policy Databases: Access control policies are stored in two separate databases. Application Assignment Database (AADB) stores applications and their corresponding roles. Permission Assignment Database (PADB) is used to store roles and the permissions assigned to roles. PADB also contains context information for permissions that have associated contexts.

4.2.4. Working Principle of CA-ARBAC

Every time an application wants to access sensitive resource other than its private resource, it makes a call to either the Middleware Layer Android API or directly to the kernel as shown by steps (A and 1) of the two lines on Fig. 13. Permissions are also enforced at both of these points. In the existing Android system, when the two Permission Enforcement Points (PEPs) receive access request message, they will decide whether the application should be allowed access or not and they either allow access to resources or send back exception message.

In Android system enhanced with ASM, the ASM intercepts access request messages and sends callback to registered ASM applications. CA-ARBAC is an ASM application. Thus, it receives callback from ASM through the PDM interface. The message contains tuples {Application ID, Permission} i.e. the application that requests access and the requested permission. The PDM analyzes the request and decides on whether the request should be allowed or denied. The PDM then responds with allow/deny message to ASM based on the decision made. The steps (4 and E) in the lines going from PDM to ASM on Fig. 13 shows allow and deny responses.

The PDM performs the following actions to make decision. It first checks if there is a role assigned to the requesting application in the AADB. If there is no any role given for the application in AADB, the PDM automatically sends deny message to ASM. If it finds a role associated with that application, it retrieves the list of permissions allowed for that role from PADB. If the requested permission is not found in the list, PDM will again send back a deny message to ASM. If the requested permission is found in the list, there are two cases. Either there is context data associated with the permission or not.

Therefore, the PDM then goes on to checking if there are any contexts associated with the permission. If the permission is not accompanied by context, it will be allowed for the application automatically. If however, there are some contexts associated with the permission, the PDM gets the current value of the context from CDB and checks if the pre-configured context is equal to the current value of the context. If all the contexts are satisfied, an allow message will be sent to ASM.

Otherwise, deny message will be sent to ASM. Based on the response from PDM, the ASM either allows the application to access the requested resource or sends back an access denied exception to the application.

CHAPTER 5

IMPLEMENTATION

Currently, ASM is not integrated to Android operating system. Therefore, we could not use ASM; instead we simulated ASM with our own application called ASM Simulator. Figure 14 shows the architecture of CA-ARBAC after ASM is replaced with ASM Simulator. In the future, we have a plan to rebuild Android operating system with ASM and integrate our system to it. For the time being, we will explain the implementation of our system using ASM Simulator application.

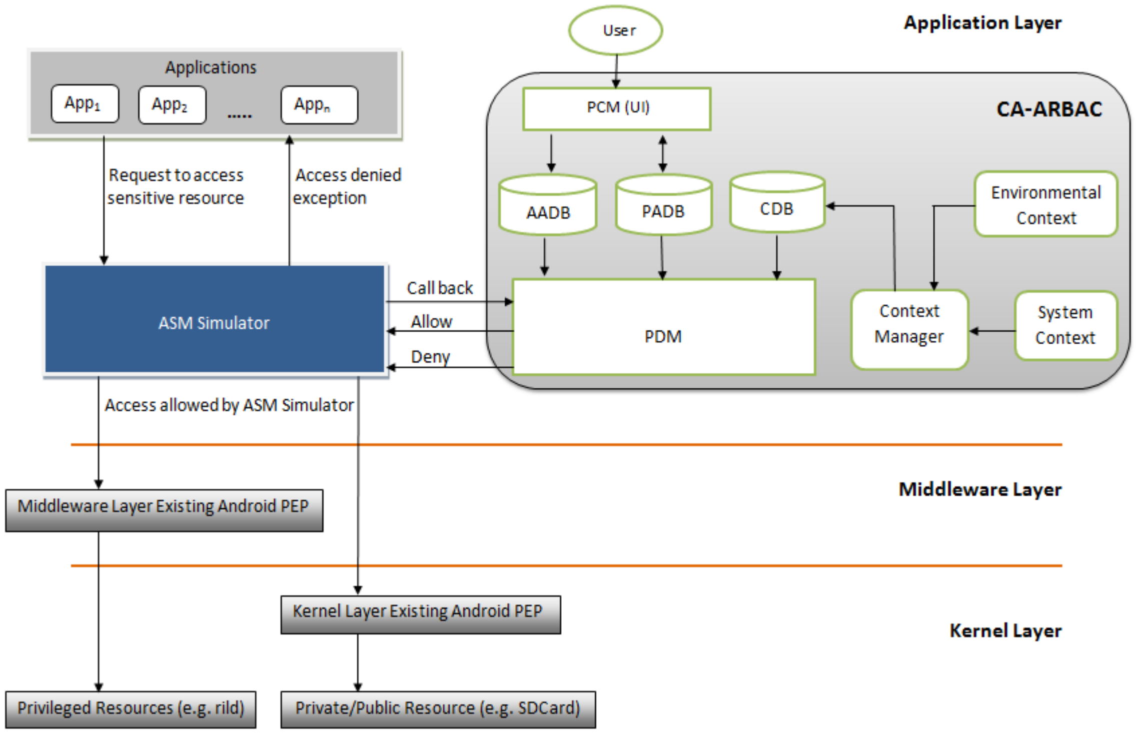


Figure 14. Architecture of CA-ARBAC based on ASM Simulator

5.1. ASM Simulator

ASM Simulator is an Android library application that controls resource access operations of other applications. ASM Simulator is different from ASM in that it lies in the application layer as opposed to ASM which is placed at two of the other layers, the middleware and kernel layers. In normal case in Android, applications directly communicate with either the Middleware Layer or the Kernel Layer PEPs to access sensitive resources. In this case, applications get access to sensitive resources through ASM Simulator.

When an application wants to access some resource, it calls the public method getCaarbacSystemService() of ASM Simulator by specifying the resource it needs to access.

The ASM Simulator checks whether the application has the necessary permission to access the requested resource by contacting CA-ARBAC. ASM Simulator calls the public method call() inside the PDM. The call() method in turn invokes the private method checkAppPermission() inside PDM itself. The arguments to both methods are tuples {Application ID, Permission}. The response to ASM Simulator is either ALLOW or DENY. If the response is ALLOW, ASM Simulator gets the resource from Android system on behalf of the application and passes the acquired resource to the requesting application. Otherwise, it sends back a security exception to the application.

5.2. CA-ARBAC Implementation

CA-ARBAC system is implemented using four of the Android application components: Activities, Services, Content Providers and Broadcast Receivers. It consists of various components that altogether perform these four main operations: policy configuration, context detection, policy decision and storage.

Policy Configuration:

PCM is the component of CA-ARBAC that provides user interfaces for policy configurations. As such, it is made up of many Android Activity classes which allow the user to perform various activities. It consists of classes used for role creation, role assignment and role modification. Role creation involves giving appropriate name to the role, assigning one or more permissions to the role and associating context with the permissions (in the case where the user is interested to associate context with permissions).

When a new role is created, it is stored in PADB. PADB and all other databases in our system are implemented using SQLite database. Role assignment is assigning roles to applications. When a role is assigned to an application, the data is saved in AADB. Role modification enables the user to modify existing roles.

Context Detection:

This part consists of the Context Manager and CDB. The list of contexts defined in the system and their current values is kept in CDB. The Context Manager is an Android service class that works continuously in the background. It constantly collects current context information from different context sources and updates the values in CDB. To be able to do so, it implements different kinds of listeners such as Android LocationListener. Context collection does not affect the performance of the other parts of our system since the service runs on a separate process independent of the other components. Moreover, not to harm the overall performance of the mobile device, it is possible to adjust the frequency at which the Context Manager collects context data.

The Context Manager service is started at boot time by the ContextManagerStarter class that extends Android Broadcast Receiver class. Broadcast Receivers can register for Intent.ACTION_BOOT_COMPLETED system intent that tells the device has completed booting. Our ContextManagerStarter class is also registered for this intent. Hence, it starts the Context Manager service when it receives the Intent.ACTION_BOOT_COMPLETED intent.

Policy Decision: Upon the arrival of request message from ASM Simulator, the PDM performs policy decisions based on the entries in the AADB, PADB and CDB. The PDM extends Android Content Provider class. It consists of various methods such as checkAppPermission(), getAppRoles(), getRolePermissions() and checkContext(). checkAppPermission() is the main method that checks whether the Application should be currently granted a given permission or not. It uses getAppRoles() to get the roles of the application from AADB and getRolePermissions() to retrieve the permissions assigned to the roles of the Application from PADB. Finally, checkContext() is used to check if the preconfigured context is satisfied or not.

Sequence Diagram: Figure 15 represents the sequence diagram for CA-ARBAC system. It shows the interaction between various components involved in the process of resource access through CA-ARBAC system.

Entity-Relationship Diagram: The various entities in our database and their relationships are also shown in the entity relationship diagram in Figure 16.

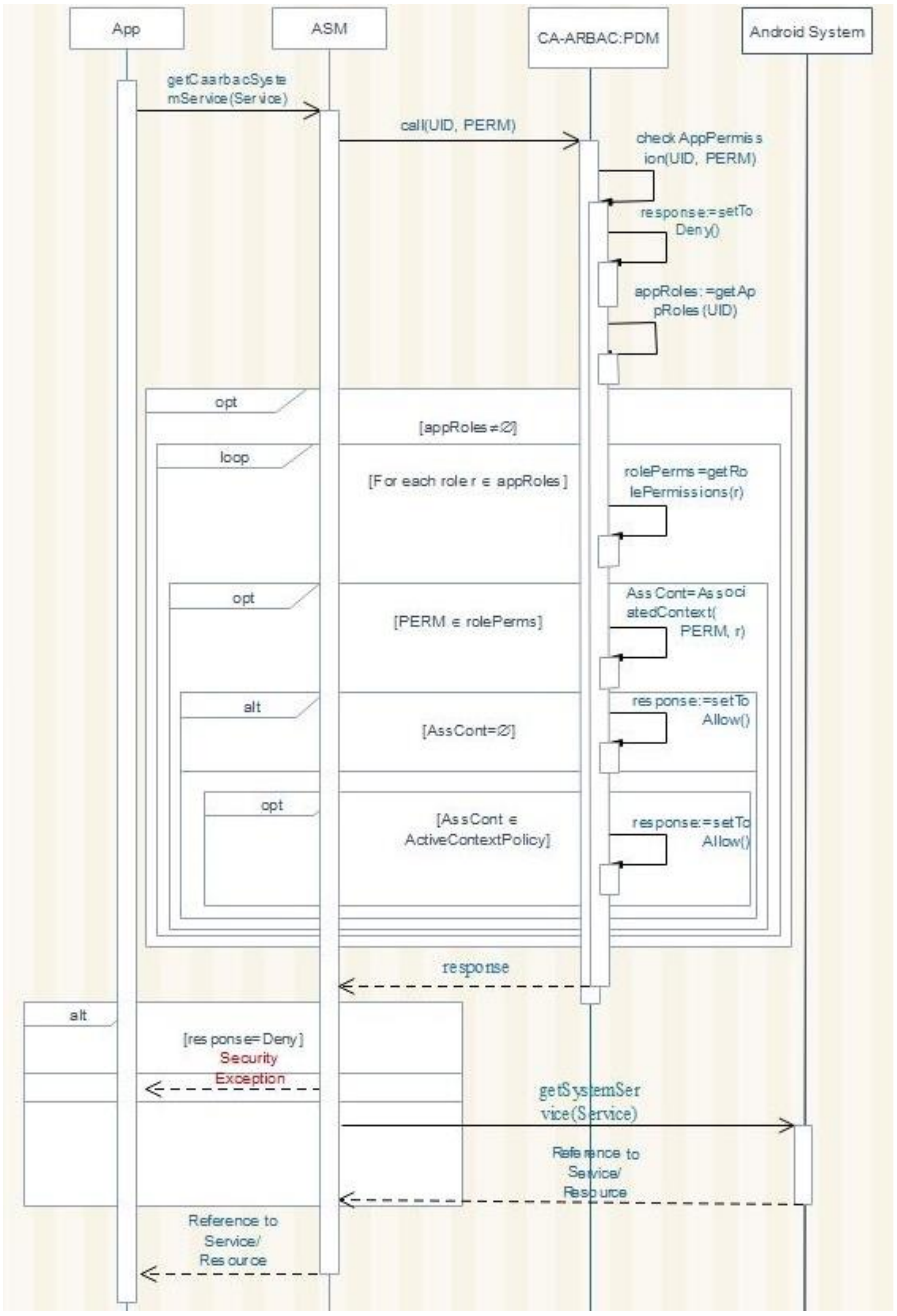

App
ASM
CA-ARBAC:PDM
Android System
getCaarbacSystemService(Service)
call(UID, PERM)
checkAppPermission(UID, PERM)
response:=setToDeny()
appRoles:=getAppRoles(UID)
opt
[appRoles≠∅]
loop
[For each role r ∈ appRoles]
rolePerms=getRolePermissions(r)
opt
[PERM ∈ rolePerms]
AssCont=AssociatedContext(PERM, r)
alt
[AssCont=∅]
response:=setToAllow()
opt
[AssCont ∈ ActiveContextPolicy]
response:=setToAllow()
response
alt
[response=Deny]
Security Exception
getSystemService(Service)
Reference to Service/ Resource
Reference to Service/ Resource

Figure 15. CA-ARBAC System Sequence Diagram

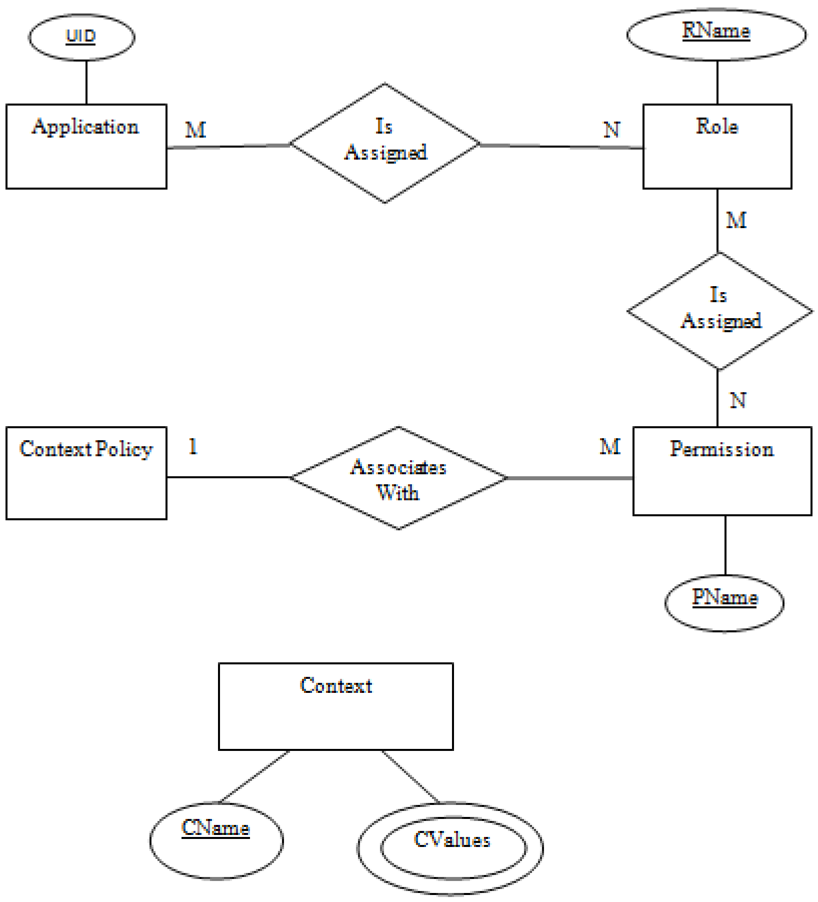
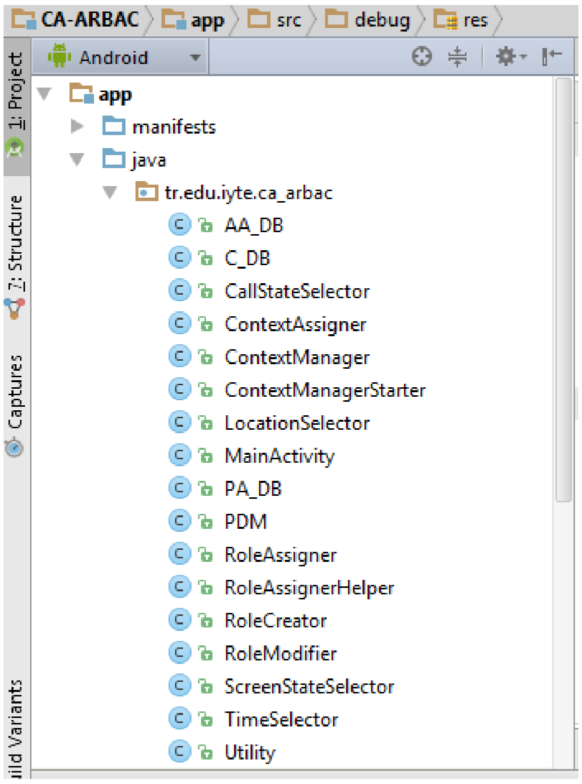


Figure 16. Entity Relationship Diagram

The Source code for our CA-ARBAC application is shown in Figure 17 below. CA-ARBAC application's user interface looks like that in Figure 18. The user interface allows users to create roles, assign created roles to applications and or modify existing roles later on. We show how these operations are performed step by step in the following sections.

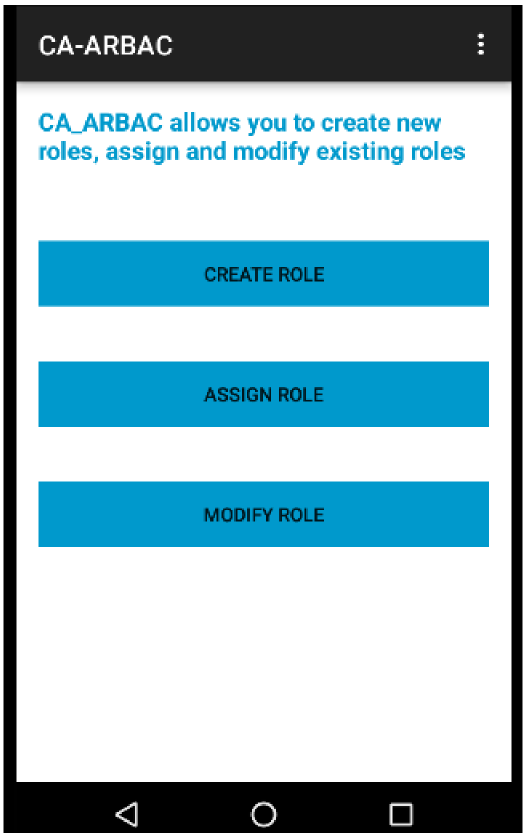


CA-ARBAC › app › src › debug › res

Android

- app
  - manifests
  - java
    - tr.edu.iyte.ca_arbac
      - AA_DB
      - C_DB
      - CallStateSelector
      - ContextAssigner
      - ContextManager
      - ContextManagerStarter
      - LocationSelector
      - MainActivity
      - PA_DB
      - PDM
      - RoleAssigner
      - RoleAssignerHelper
      - RoleCreator
      - RoleModifier
      - ScreenStateSelector
      - TimeSelector
      - Utility

Figure 17. Source Code of CA-ARBAC



CA-ARBAC

CA_ARBAC allows you to create new roles, assign and modify existing roles

Figure 18. User Interface of CA-ARBAC system

CHAPTER 6

EXPERIMENT AND ANALYSIS

6.1. Experimental Tests

In this section, we further demonstrate our system by using some real examples. We test the working of our system using three applications, three roles and four kinds of contexts. We developed three test applications for this purpose. The three test applications are shown in Figure 19. The first one is a messenger application called MyMessenger that allows phone call, SMS sending and audio recording. The second one is called PhotoEditor. It is a photography application that allows taking photos and editing them. The last application is a simple application that gets the users current location and displays it. We named it LocationGetter. The three roles we created for our test are: MESSENGER, PHOTOGRAPHY and TRAVEL roles. Four types of contexts namely LOCATION, CALL_STATE, SCREEN_STATE and TIME are used for this test.

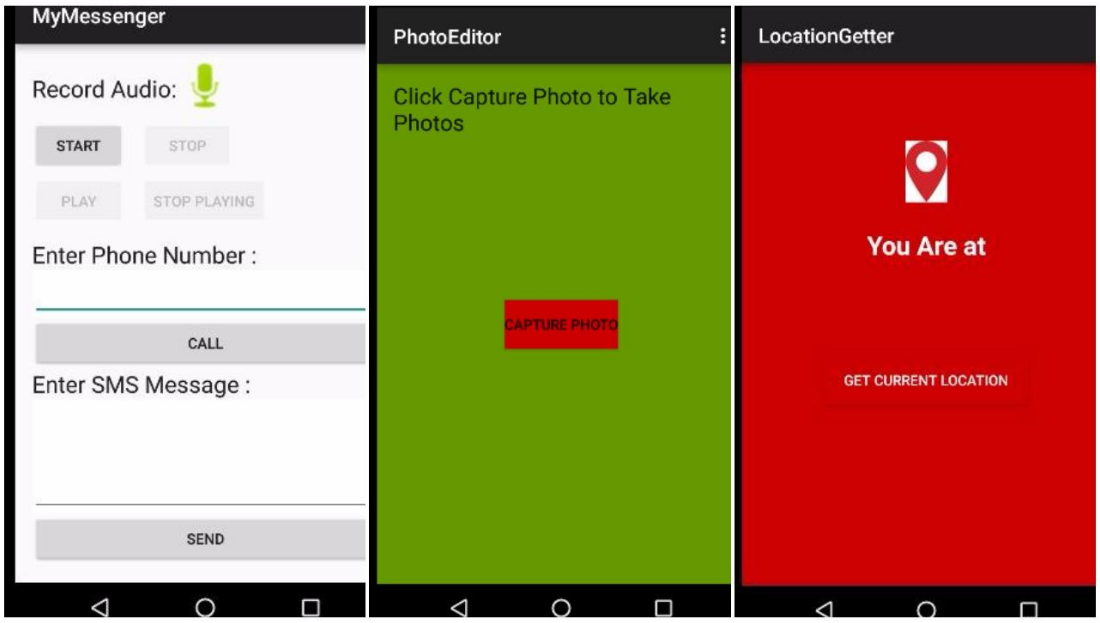


Figure 19. Applications Used for Experimental Test

6.1.1 Creating Roles

Before we look at examples, it is essential to discuss the method we used to generate roles for applications and determine permissions to be assigned for roles. Different techniques may be used to do this. As mentioned earlier, we assign roles to applications based on their functional group. Applications are developed to give some kind of services to users. Different applications provide different kinds of functions. Some applications like games are designed for simple entertainment purposes. Others are intended to be used for critical functions like financial transactions. As a result of this, not all applications need the same kind of permissions. There are permissions which are necessary for one type of application and prone to misuse for other type. For instance, allowing sending SMS messages is quite normal for messengers but very suspicious for games. Therefore, to discover sample roles, we followed a method of classifying applications into logical groups based on the functionality they are designed for. Therefore, we categorize applications into functional groups and create different roles which are appropriate for each functional category.

To give more insight into the process of role creation and assignment in our system, we look at some examples. For instance, “PHOTOGRAPHY” role can be created for photography applications and a “MESSENGER” role can be created for messenger applications. These roles will be allocated different number and types of permissions. For example, permissions such as CALL_PHONE, SEND_SMS, RECIEVE_SMS, RECORD_AUDIO, CAMERA, WRITE_CONTACTS or READ_CONTACTS are normal for messenger applications but most of them are suspicious if requested by photography applications. Thus, out of these mentioned permissions, PHOTOGRAPHY role may be assigned only CAMERA permission whereas MESSENGER role could be granted all of them.

Moreover, using the context awareness functionality of the system, we can impose restrictions on permission usage for already granted permissions. For instance, we can set the precondition that RECORD_AUDIO permission is not allowed for MESSENGER role during the time that the user is talking on the phone or the user is in meeting room. We can also say that PHOTOGRAPHY role is forbidden from using CAMERA permission if the user is at home or in meeting room. One of the common attacks by hackers is calling and or sending SMS messages to premium numbers when the phone is locked. In such cases, we may have a context policy that forbids the application from using permissions such as CALL_PHONE, SEND_SMS and RECIEVE_SMS if the screen is locked. Another common privacy attack is tracking users' location. The user can have a policy that denies location permission to applications when the user is at secure locations such as home. As mentioned earlier, in Android version 6, all applications have internet permission by default. This is not what users really want. We believe that limiting the internet permission depending on context is the better way to do it which is possible in our system.

When we come to the implementation of this method of categorizing applications into logical groups, it is not as simple as it seems. Being able to create roles which contain optimum number of permissions is one of the challenges of our system. For skilled users, we believe that deciding which permissions to assign to which roles is completely up to the user. However, as most users of mobile devices are ordinary users who have difficulty in creating roles, there is a need to create default system roles that can be used as needed. Currently, there is no any reference standard that states which kinds of applications should use which kind of permissions. There is also no satisfactory system that can identify the permissions appropriate for the different categories of applications. This topic by itself is a new research area that needs further study.

However, there are few works such as [29], [30], [31], [32] and [33] which have done limited researches on this topic. Most of these studies used this methodology as a way of detecting malicious Android applications. Among them we found [29] to be more convenient for our work. We used their open source application called "SuspiciousAppsChecker" [29] to find sample application roles and corresponding permissions. "SuspiciousAppsChecker" is an application that analyzes Android applications for over-privilege. It checks Android applications for over-privilege by comparing the permissions used by the applications with a pre-defined permission list allowed for the category that the application belongs. We identified sample applications roles and permission lists for each role by using the data we get from SuspiciousAppsChecker.

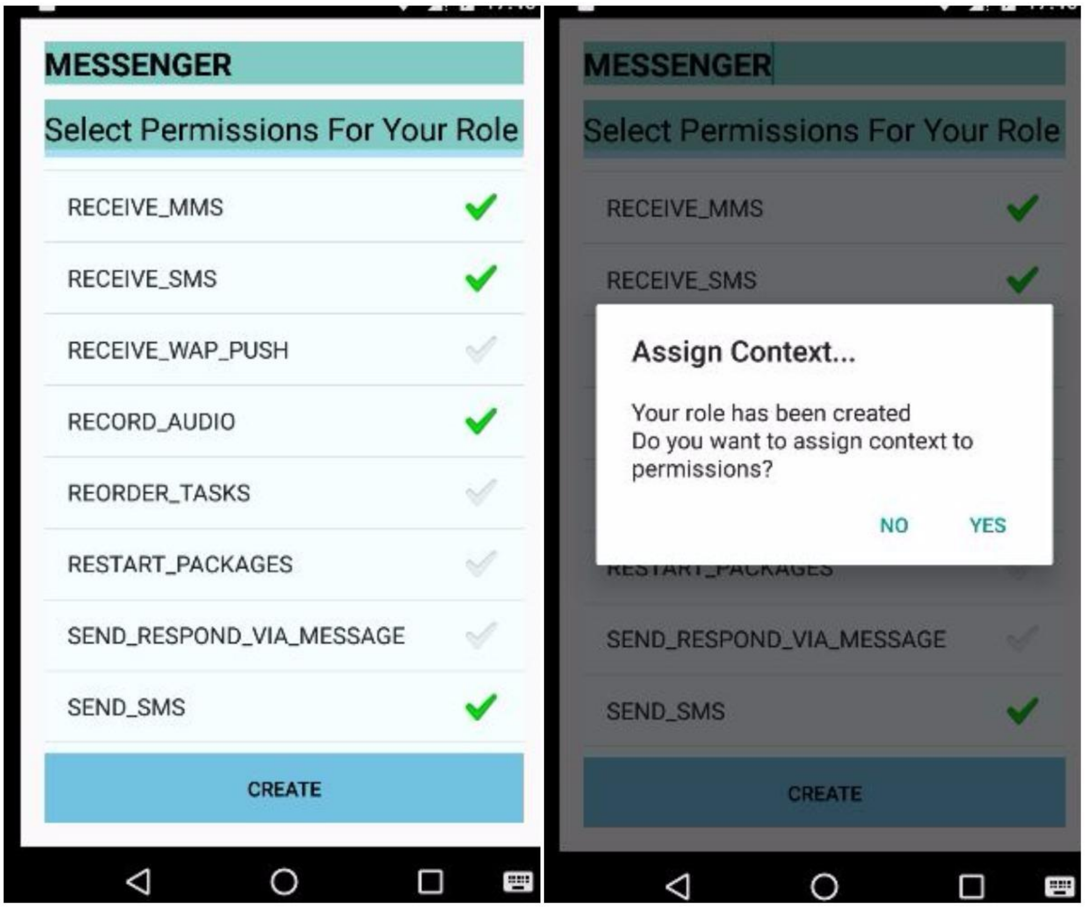
We recognize that this is not sufficient way of generating roles for applications. First of all, the methodology by itself may not be taken as a good means of dealing with this problem. Secondly, SuspiciousAppsChecker is not yet mature and has limitations. To mention one, the categorization is too general and not fine-grained. For instance, the system assumes that all messenger applications belong to the same category and believes that all messenger applications should be given the same set of permissions. In reality, there are various kinds of messenger applications such as text messaging applications and voice messaging applications. For example, RECORD_AUDIO permission is not necessary for text messaging applications but is must for voice messaging applications. So it is wrong to group all messenger applications into one category and assign them the same set of permissions.

In the future, we have a plan to develop a system that can automatically analyze applications, determine appropriate permissions for applications and suggest appropriate roles to users. We also hope that a better automated technique may be discovered by other researchers. Nonetheless, for the purpose of explaining our model, we believe that it is adequate to use simple samples developed with the help of SuspiciousAppsChecker because our main goal in this thesis is not identifying roles and equivalent permissions but rather showing that Context Aware Role Based Access Control can be used to provide usable privacy preserving permission system.

Helped by SuspiciousAppsChecker application, we derived three roles that we use for test purpose. The three roles, the permissions they contain and the contexts associated with them are shown in Table 4 below. Roles are created in CA-ARBAC system as shown in Figure 20. During role creation, a popup window is displayed and asks the user if he wants to configure context for one or more of the permissions selected for the role. We present how context is configured in the next section.

Table 4. Example Application roles, permissions and contexts

Role	**Permissions Assigned to the Role**	**Context Policy**	**Context Policy Type**
MESSENGER	RECORD_AUDIO	USER IN MEETING (LOCATION +TIME)	DENY
		USER TALKING ON PHONE	DENY
		PHONE IS LOCKED	DENY
	READ_CONTACTS		
	WRITE_CONTACTS		
	CALL_PHONE	PHONE IS LOCKED	DENY
	SEND_SMS	PHONE IS LOCKED	DENY
	RECEIVE_SMS		
	READ_SMS	PHONE IS LOCKED	DENY
	⋮	⋮	⋮
TRAVEL	INTERNET		
	ACCESS COARSE LOCATION	USER NOT AT HOME	ALLOW
	ACCESS_FINE_LOCATION	USER NOT AT HOME	ALLOW
	⋮	⋮	⋮
PHOTOGRAPHY	CAMERA	USER NOT AT HOME	ALLOW
	WRITE_EXTERNAL_STORAGE		
	READ_EXTERNAL_STORAGE		
	⋮	⋮	⋮


MESSENGER

Select Permissions For Your Role

RECEIVE_MMS	✓
RECEIVE_SMS	✓
RECEIVE_WAP_PUSH	
RECORD_AUDIO	✓
REORDER_TASKS	
RESTART_PACKAGES	
SEND_RESPOND_VIA_MESSAGE	
SEND_SMS	✓

CREATE

Assign Context...

Your role has been created
Do you want to assign context to permissions?

NO YES

Figure 20. Role Creation in CA-ARBAC System

6.1.2 Associating Contexts

Once the user created a role by selecting one or more permissions, CA-ARBAC system asks the user if he wants to associate contexts with the selected permissions. If the user is willing to configure contexts, he is forwarded to a context configuration screen. Figure 21 shows CA-ARBAC context configuration screen. It allows the user to select policy type (allow/deny), context types (location, time, call state, screen status) and the permission for which the context should be applied.

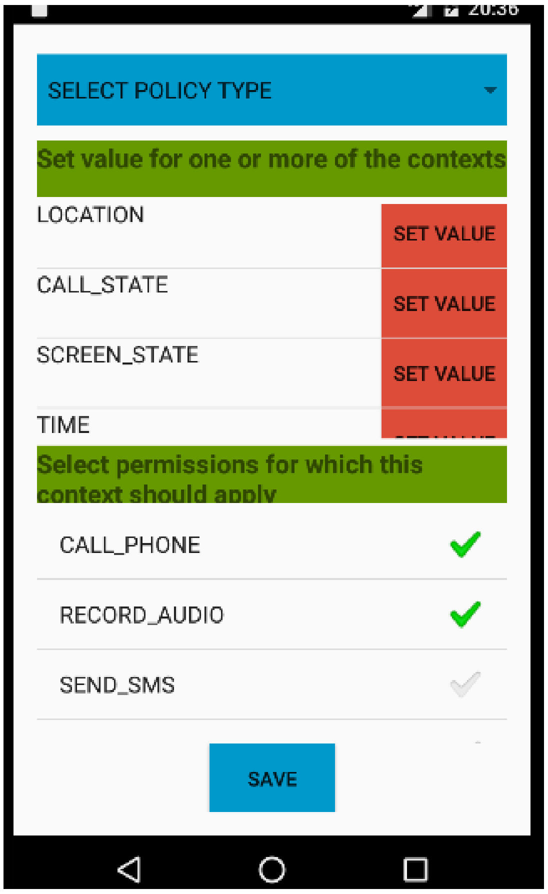


Figure 21. Context Configuration Screen

We associated contexts with three of the permissions assigned to our roles as shown in Table 4. For example, RECORD_AUDIO permission is a desirable permission for messenger applications. However, it may be very dangerous at some conditions such as when the user is in a meeting, or if the user is talking on phone and if the phone is locked. Therefore, we can set a policy that says RECORD_AUDIO permission is not allowed if the user is in one of these situations. To know that the user is in a meeting, we may need to know the meeting place and time. Hence, it is expressed using a combination of LOCATION and TIME contexts. For instance, let say user John has meeting at IYTE computer engineering department every Monday and Friday from 2:30 pm to 4:30 pm and he doesn't want applications to record audio while he is in a meeting. The snapshot in Figure 20 shows how location and time context policy can be configured in our system. For location context, the user sets a circular area by selecting two points on the map. Configuring time context involves selecting time range and days. The context policy for user John's context requirement is represented in our system as:

<([LOCATION=38.32099966466455;26.64043352007866;
38.321032544732574;26.640723198652267] ∧
[TIME=1430;1630;MONDAY,FRIDAY]),DENY>.

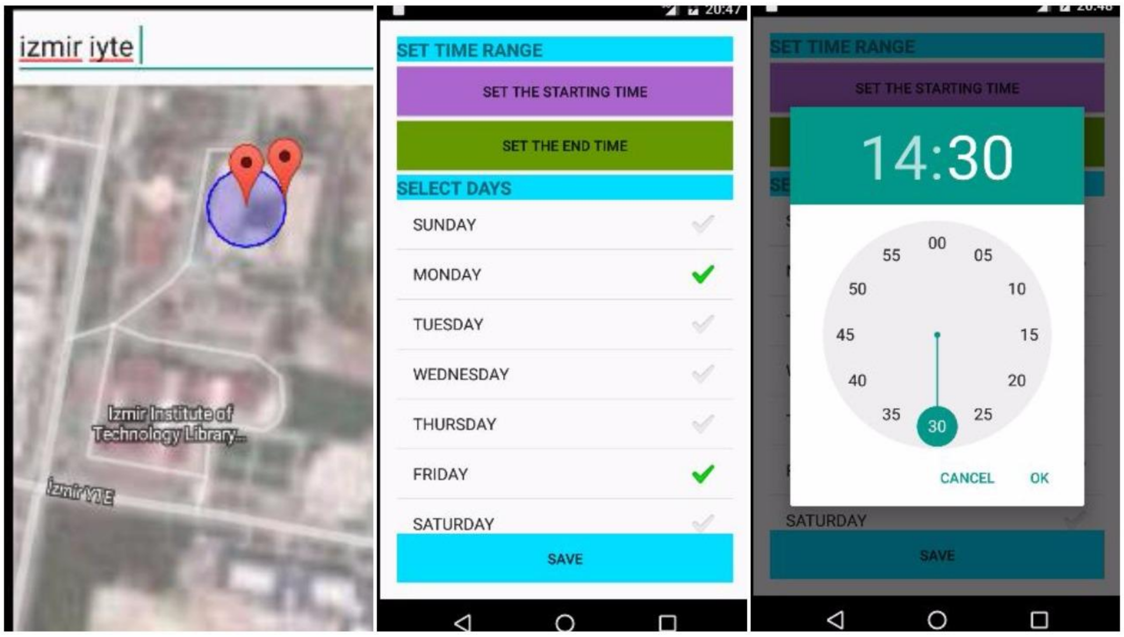


Figure 22. Location and Time Context Configuration

Moreover, John also does not want applications to record audio if he is talking on phone or if his phone is locked. Figure 21 below shows how these two contexts are configured. The context policy for these two situations looks like this in our system: <[CALL_STATE=CALL_STATE_OFFHOOK],DENY> and <[SCREEN_STATE=SCREEN_STATE_OFF],DENY> respectively.

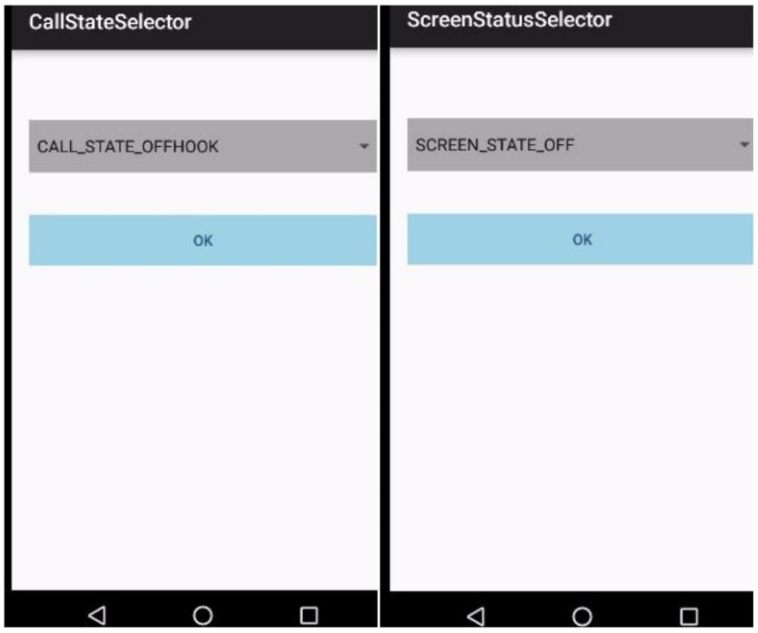


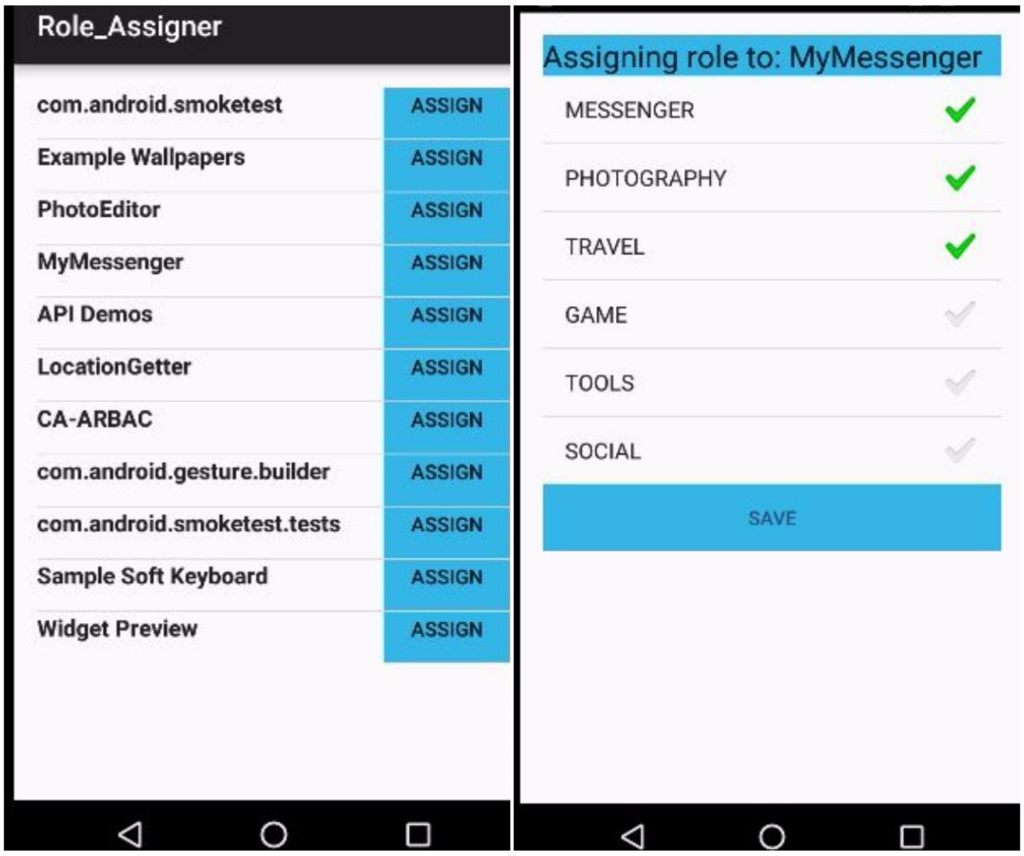
Figure 23. Call State and Screen State Context Configuration

Similarly, the user John specified a context rule which says that CALL_PHONE, SEND_SMS and READ_SMS permissions are not allowed if his phone is locked. Finally, John stated that ACCESS_FINE_LOCATION and CAMERA permissions are allowed only if he is out of his home such as in cafeteria, markets and work place. Let us assume that John’s work place is at IYTE computer engineering department and his home is at Inciralti Ataturk Students Dormitory.

6.1.3 Assigning Roles to Applications and Making Experimental Tests

In this section, we first see role assignment to our test applications and then we conduct an experiment by running our test applications in different contexts.

Let say John who has configured the prior contexts has installed our three applications. John then has to assign roles to the applications based on the permissions they require. Our test applications require different number of permissions. The MyMessenger application which represents a messenger application requires the highest number of risky permissions. It usually requires most of the permissions in MESSENGER role. Moreover, messengers may require CAMERA permission which is in the PHOTOGRAPHY role. Messengers also usually need access to location. Location permission is contained in TRAVEL role. Hence, John assigned MyMessenger all the three roles in the system. The other two applications each is assigned one role; PhotoEditor is assigned PHOTOGRAPHY role and LocationGetter is assigned TRAVEL role. Role assignment for MyMessenger application is performed as shown in Figure 22.


Role_Assigner

com.android.smoketest	ASSIGN
Example Wallpapers	ASSIGN
PhotoEditor	ASSIGN
MyMessenger	ASSIGN
API Demos	ASSIGN
LocationGetter	ASSIGN
CA-ARBAC	ASSIGN
com.android.gesture.builder	ASSIGN
com.android.smoketest.tests	ASSIGN
Sample Soft Keyboard	ASSIGN
Widget Preview	ASSIGN

Assigning role to: MyMessenger

MESSENGER	✓
PHOTOGRAPHY	✓
TRAVEL	✓
GAME	
TOOLS	
SOCIAL	

SAVE

Figure 24. Role Assignment for MyMessenger Application

We did various tests based on John's policy configurations as follows:-

Firstly, we performed the following four tests using MyMessenger application.

TEST1:

We tried to record audio using MyMessenger at IYTE computer engineering department meeting room on Monday and Friday between 2:30pm and 4:30pm.

We also checked if we can record audio at some other contexts.

RESULT1:

The result shows that we are not able to record audio on the given days and time. MyMessenger application crashes. Have a look at left side of Figure 23. Moreover, a security exception is thrown by ASM Simulator as shown in Figure 24. However, we can record audio at other contexts (right side of Figure 23).

TEST2:

We tried to record audio while there is an ongoing phone call. We also tried to record audio while the phone is idle.

RESULT2:

We cannot record audio on the first case but we are able to record audio for the second scenario.

TEST3:

We again tried to record audio while the phone's screen is on. We also tried to record audio while the phone is locked.

RESULT3:

It is possible to record audio for the former case but not for the latter case.

TEST4:

We checked if it is possible to call a phone, send and read SMS while the phone is locked. We also tested if the same thing may happen when the phone is unlocked.

RESULT4:

The result shows that it is possible to call a phone, send and read SMS when the phone is unlocked but not possible for the opposite case.

Secondly, we made a single test using PhotoEditor application

TEST5:

We tried to take photos at Inciralti Ataturk Students Dormitory and also at IYTE computer engineering department.

RESULT5:

PhotoEditor can take photos when we are at IYTE computer engineering department but not at Inciralti Ataturk Students Dormitory.

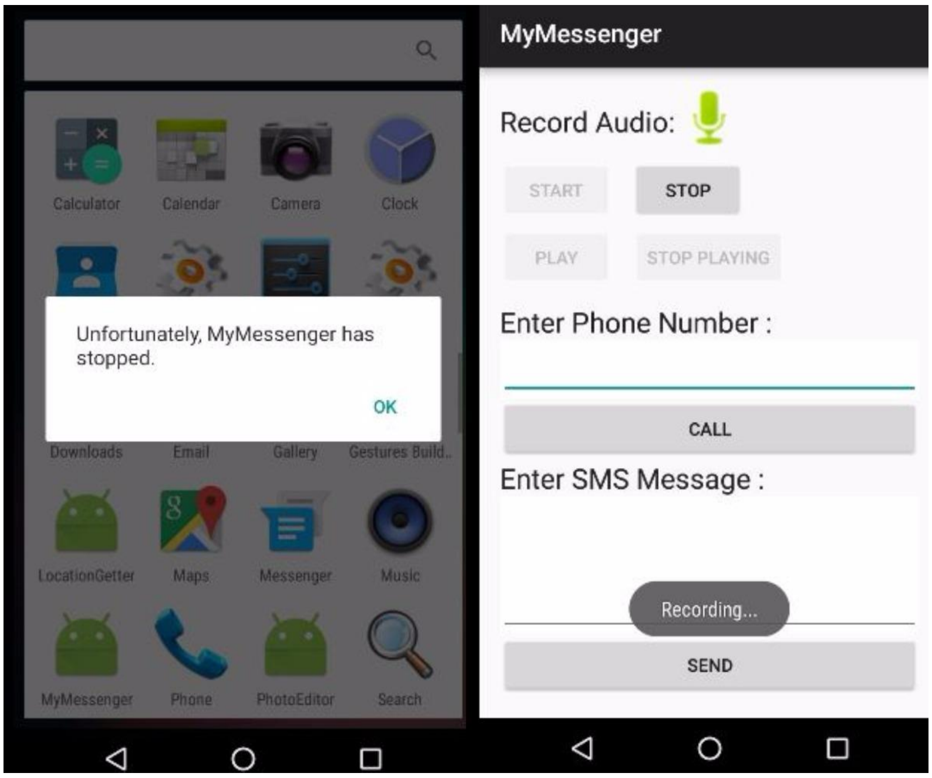
Finally we made a test using LocationGetter application.

TEST6:

We tried to get the current location of the user at IYTE computer engineering department and outside of it.

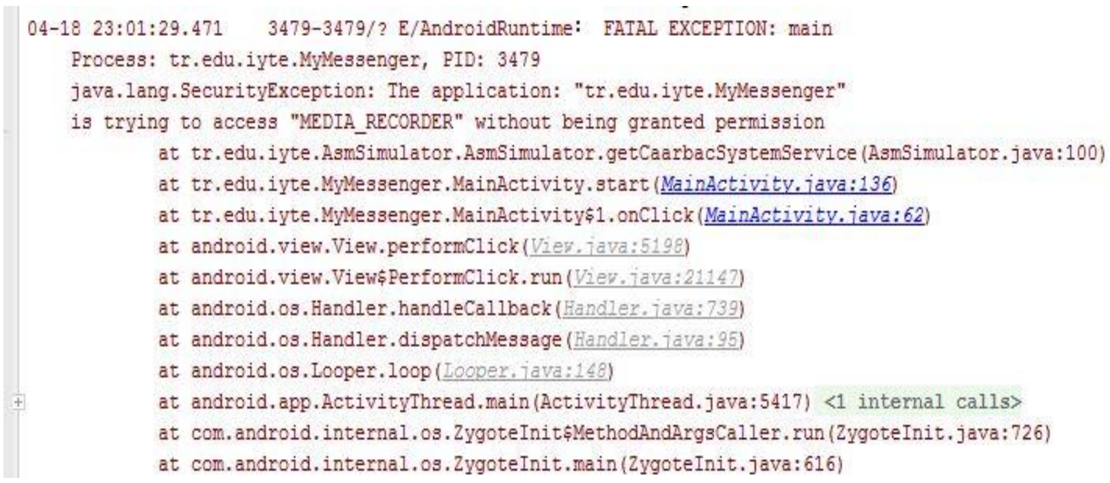
RESULT6:

We can get the location of the user outside of IYTE computer engineering department but not inside it.



Unfortunately, MyMessenger has stopped.

Figure 25. MyMessenger Application Crashing and Recording Cases



```
04-18 23:01:29.471     3479-3479/? E/AndroidRuntime:  FATAL EXCEPTION: main
    Process: tr.edu.iyte.MyMessenger, PID: 3479
    java.lang.SecurityException: The application: "tr.edu.iyte.MyMessenger"
    is trying to access "MEDIA_RECORDER" without being granted permission
            at tr.edu.iyte.AsmSimulator.AsmSimulator.getCaarbacSystemService(AsmSimulator.java:100)
            at tr.edu.iyte.MyMessenger.MainActivity.start(MainActivity.java:136)
            at tr.edu.iyte.MyMessenger.MainActivity$1.onClick(MainActivity.java:62)
            at android.view.View.performClick(View.java:5198)
            at android.view.View$PerformClick.run(View.java:21147)
            at android.os.Handler.handleCallback(Handler.java:739)
            at android.os.Handler.dispatchMessage(Handler.java:95)
            at android.os.Looper.loop(Looper.java:148)
            at android.app.ActivityThread.main(ActivityThread.java:5417) <1 internal calls>
            at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:726)
            at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:616)
```

Figure 26. Security Exception Thrown by ASM Simulator

6.2. Formal Verification

In the previous sections, we showed how our system enhances user privacy in APS by enabling dynamic permission granting and revoking. However, this is not enough to show that the system is valid. One of the important criteria for access control policies is the ability to prevent unauthorized access to resources. An access control policy shouldn’t also deny access to resources for actors which possess the right privilege. In this section, we present the formal expression of our access control policy to prove that our system allows only authorized applications to get access to permissions. In our model, an application is authorized to use a given permission if all of the three properties below are fulfilled:-

- If there exists a role that the application is assigned and
- If that role contains the requested permission and
- If the context (if any) associated with the given permission for that role is satisfied and
- If there does not exist any other role containing the same permission and also assigned to the same application but the context associated with the permission is not satisfied.

The last rule is required because a single permission can appear in different roles and hence can have different context policies associated with it for different roles. For example permission P can have context policy CP_1 for role R_1. The same permission may be associated with another context policy CP_2 when assigned to role R_2. Hence, for instance, if application A is assigned the two roles R_1 and R_2, allowing permission P for application A requires that both of the two contexts CP_1 and CP_2 be satisfied. Otherwise, if application A is allowed to use permission P based on the satisfaction of only one of the contexts, it leads to contradiction.

We demonstrate the formal expression of our model based on the previous definitions covered in Chapter 4: - A: set of applications, R: set of roles, P: set of permissions, CP: set of context policies, ARM: application role mapping and RPM: role permission mapping. We also introduce more definitions in this section as follows:-

- $\text{AssignedApps}(r:\text{Role}) \rightarrow 2^A$ is the mapping of a set of applications to role r. i.e.

$$\text{AssignedApps}(r) = \{a \in A \mid (a, r) \in ARM\} \tag{6.1}$$

- $\text{AssignedPerms}(r:\text{Role}) \rightarrow 2^P$ is the mapping of a set of permissions to role r. i.e.

$$\text{AssignedPerms}(r) = \{p \in P \mid (r, p) \in RPM\} \tag{6.2}$$

- Permission Context Mapping: Let PCM be the list consisting of the mapping between permissions and associated context policies. It contains triplets <P_i, R_i, CP_i > where P_i ∈ P, R_i ∈ R and CP_i ∈ CP. CP.
- AssociatedContext(p:Permission, r: Role)→CP is the mapping of permission p to context policy cp for role r. i.e.

$$\text{AssociatedContext}(p, r) = \{cp \in CP \mid (p, r, cp) \in PCM\} \tag{6.3}$$

- ContextState = {1, 0}, is the set containing the possible outcome of a context policy rule. At any given time, the context policy rule evaluates to either true or false. If it evaluates to true, the ContextState is set to 1, otherwise it is set to 0.
- Context State Mapping: Let CSM be the list consisting of the mapping between context policy rules and their states. It contains duplets <CP_i, CS> where CP_i ∈ CP and CS ∈ ContextState.

$$\text{ActiveContextPolicies}=\{cp \in CP \mid (cp, 1) \in CSM\} \tag{6.4}$$

$$\text{InactiveContextPolicies} =\{cp \in CP \mid (cp, 0) \in CSM\} \tag{6.5}$$

Hence, the run time authorization decision in our system is governed by the following formal expression:-

$$\begin{aligned}
&(\forall a : Application)(\forall p : Permission):\\
&allow(a, p) \Rightarrow\\
&(\exists r : Role)(\exists p : Permission):\\
&\left[a \in AssignedApps(r) \wedge p \in AssignedPerms(r)\right]\\
&\wedge \begin{bmatrix} (AssociatedContext\,(p,r) = \varnothing) \vee \\ (AssociatedContext\,(p,r) \in ActiveContextPolicies) \end{bmatrix}\\
&\wedge \neg \begin{bmatrix} (\exists r' : Role) : (r' \neq r) \wedge a \in AssignedApps(r') \wedge p \in AssignedPerms(r') \wedge \\ AssociatedContext\,(p,r') \in InactiveContextPolicies() \end{bmatrix}
\end{aligned}$$

Figure 25 also explains the run time authorization decision pictorically by example.

In Figure 25, A_1 cannot use P1 because even if A_1 ∈ AssignedApps(R_1) ∧ P_1 ∈ AssignedPerms(R_1) ∧ AssociatedContext(P_1, R_1) ∈ ActiveContextPolicies, there is another contradicting rule. i.e. A_1 ∈ AssignedApps (R_2) ∧ P_1 ∈ AssignedPerms (R_2) ∧

AssociatedContext (P_1, R_2) ∈ InactiveContextPolicies. Similarly A_1 cannot use P_4 because A_1 ∈ AssignedApps (R_2) ∧ P_4 ∈ AssignedPerms (R_2) ∧ AssociatedContext (P4, R_2) ∈ InactiveContextPolicies. However, A_1 can use P_2 and P_5 because A_1∈ AssignedApps (R_1) ∧ P_2 ∈ AssignedPerms (R_1) ∧ AssociatedContext (P_2, R_1) = Ø and also A_1∈ AssignedApps (R_2) ∧ P_5 ∈ AssignedPerms (R_2) ∧ AssociatedContext (P_5, R_2) = Ø. Moreover, A_1 can use P_3 because A_1∈ AssignedApps (R_1) ∧ P_3 ∈ AssignedPerms (R_1) ∧ AssociatedContext (P_3, R_1) ∉ ActiveContextPolicies.

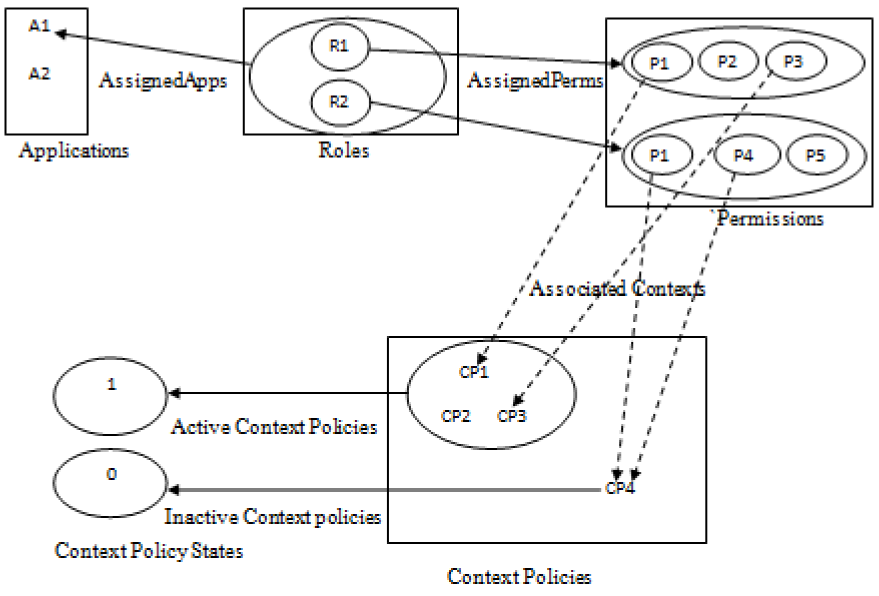


Figure 27. CA-ARBAC run time authorization decision by example

CHAPTER 7

DISCUSSION AND FUTURE WORKS

The motivation behind this thesis was the privacy problems caused by the manual nature of permission granting/revoking in mobile systems and particularly in Android permission system. As a solution to this problem, we proposed a context-aware role based access control where the context is associated with permissions. Others researchers have also suggested solutions close to ours. Some suggested context-aware role based access control where context is associated with roles. Others suggested pure context aware access control which links permissions to context but does not include role based access control. The former may result in unnecessary creation of large number of roles. Moreover, the former also does not provide permission level fine grained context policy while the later creates usability problems by requiring the user to tackle with a lot of policy configurations. Our approach is different from the two in that it can protect user's privacy without affecting the usability and also without the requirement to create excess roles.

The various steps we passed through in the previous chapters show that our approach is feasible and can be realized. We designed our access control model policy model in a novel way. We also contributed a new architecture on Android and implemented the prototype on Android. Based on the prototype implementation, we did various experiments to see if our system gives the desired result. Experimental results gave us promising results. All of the outputs were as expected. Furthermore, we showed how formal verification is done in our model. Nevertheless, because of the various restrictions we have, there are some issues which we are forced to leave aside for now. The following are some directions in which this thesis research can be improved:-

Active Context Management: In our current system, we use passive context management. In passive context management, once an application is granted permission, it can use it irrespective of changes in context information. However, for realistic situations, the application should be revoked access if the context changes during the time that the application is using the permission. We could not use Active Context Management in CA-ARBAC as the underlying Android security policy enforcement framework does not support context management. ASM framework also does not support context management. For the future, we suggest the integration of security APIs similar to ASM that support Active Context Management.

Integrating CA-ARBAC System to Android: There are two choices to implement security applications such as CA-ARBAC on Android. The first one is by modifying the operating system. This way is not effective as already explained in Chapter 4. The second and better way is implementing it on a security framework that provides API. Currently, in Android, there is no security framework that provides security API that allows the development of independent security applications such as CA-ARBAC system. ASM is one such security framework project aimed to solve this problem. We designed CA-ARBAC system based on ASM. However, ASM is not yet integrated to Android. Therefore, we used a simulation for ASM. In the future, CA-ARBAC system implementation should be tested based the real ASM or any other framework similar to ASM.

Usability Test: We argue that our system can be better in usability than the existing Android permission system. But, this claim needs to be confirmed with user studies that measure usability.

Default System Roles: Skilled users can easily create roles on their own. This might not be an easy task when it comes to naive users. The situation becomes more difficult if the user has to configure context for permissions. One of the solutions for this problem can be having system default roles. Creating default roles requires that the roles should contain optimum number of permissions and context configuration. As explained earlier, we used a rough method of grouping applications into functional groups to create roles. We believe that this is not the only way to so do. For example, we can think of an automated system that can analyze applications and suggest roles to the user.

CHAPTER 8

CONCLUSION

In this thesis paper, we proposed a new access control model to protect user's privacy on mobile systems. Our model allows dynamic granting and revoking of applications' permissions based on predefined contexts. Even though the proposed model can be used on other platforms, our work in this thesis focuses more on Android permission system. Our system is a variety of CA-RBAC designed in such a way that roles which will consist of a set of Android permissions are assigned to applications and contexts are associated with permissions. The proposed model is different from most traditional CA-RBAC models in that it associates contexts with permissions in contrary to the classical one which associates contexts with roles. We designed a novel architecture for our proposed system based on ASM. ASM is a new project that is aimed to provide security enforcement framework API that enables others to develop their own security applications without worrying about the low level security policy enforcement. However, as ASM is not currently integrated to Android, we simulated ASM. We also developed and implemented a prototype application called CA-ARBAC (Context-Aware Android Role Based Access Control) on Android. Based on our prototype application, we made various experimental tests using three test applications, three roles and four kinds of contexts. The experimental results are all as expected. Both the Role based access control and the context access control part works well. Moreover, we tried to show the formal verification of our access control model.

We have also identified some important future works such as Active context management, usability test, creating default systems roles and integrating our system to Android. One of the challenges was creating system default roles that can be used by ordinary users. We believe that our proposed system can provide better privacy without significant effect on the usability of the permission system. However, to reach on full conclusion about the usability aspect of our system, a usability test is required which we leave it to future works.

REFERENCES

[1] "Smartphone Users Worldwide Will Total 1.75 Billion in 2014," eMarketer, 16 01 2014. [Online]. Available: http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536. [Accessed 20 04 2015].

[2] "Worldwide Smartphone Shipments Edge Past 300 Million Units in the Second Quarter; Android and iOS Devices Account for 96% of the Global Market," IDC, 14 08 2014. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS25037214. [Accessed 20 04 2015].

[3] "Kaspersky Lab KSN Report mobile cyberthreats," Kaspersky Lab & INTERPOL Joint Report, 10 2014. [Online]. Available: http://media.kaspersky.com/pdf/Kaspersky-Lab-KSN-Report-mobile-cyberthreats-web.pdf. [Accessed 20 04 2015].

[4] J. Leyden, (Apr. 2013). Your phone may not be spying on you now—but it soon will be. [Online]. Available: http://www.theregister.co.uk/2013/04/24/kaspersky_mobile_malware_infosec/

[5] R. Templeman, Z. Rahman, D. J. Crandall, and A. Kapadia, "Placeraider: Virtual theft in physical spaces with smartphones," in Proc. 20th Annual Netw. Distrib. Syst. Security Symp. (NDSS), Feb. 2013.

[6] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound trojan for smartphones," in Proc. 18th Annu. Netw. Distrib. Syst. Security Symp., Feb. 2011, pp. 17–33.

[7] A.P.Felt, S. E. E. Ha, A. Haney, E. Chin and D. Wagner, "Android Permissions: User Attention, Comprehension, and Behavior," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*, New York, ACM, 2012, pp. 3:1--3:14.

[8] S.Heuser, A.Nadkarni, W.Enck and A.R.Sadeghi, "A Programmable Interface for Extending Android Security," in *Proceedings of the 23rd USENIX Security Symposium*, San Diego, CA, 2014.

[9] M.Backes, S.Gerling, C.Hammer, M.Maffei and P.Styp-Rekowsky, "AppGuard - Fine-grained Policy Enforcement for Untrusted Android Applications," in *Proc.\ 8th International Workshop on Data Privacy Management (DPM 2013)*, Springer, 2013, pp. 213-231.

[10] M.Nauman, S.Khan and X.Zhang, "Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, Beijing;, ACM, 2010, pp. 328-332.

[11] "BlurSense: Dynamic fine-grained access control for smartphone privacy," in *Sensors Applications Symposium (SAS), 2014 IEEE*, 2014, pp. 329-332.

[12] C.D.Stelly, "Dynamic User Defined Permissions for Android Devices," M.S.thesis, Dept. Com.Sci., University of New Orleans, LA, USA, 2013.

[13] S.Bugiel, S.Heuser and A.Sadeghi, "Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies," in *Proceedings of the 22Nd USENIX Conference on Security*, Washington, D.C., USENIX Association, 2013, pp. 131-146.

[14] Y.Zhou, X.Zhang, X.Jiang and V.W.Freeh, "Taming Information-stealing Smartphone Applications (on Android)," in *Proceedings of the 4th International Conference on Trust and Trustworthy Computing*, Berlin, Springer-Verlag, 2011, pp. 93-107.

[15] A.R.Beresford, A.Rice, N.Skehin and R.Sohan, "MockDroid: Trading Privacy for Application Functionality on Smartphones," in *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, New York, ACM, 2011, pp. 49-54.

[16] J.Jeon, K.K.Micinski, J.A.Vaughan, A.Fogel, N.Reddy, J.S.Foster and T.Millstein, "Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications," in *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, New York, ACM, 2012, pp. 3-14.

[17] B. Shebaro, O. Oluwatimi and a. E. Bertino, "Context-Based Access Control Systems for Mobile Devices," in *Dependable and Secure Computing, IEEE Transactions on*, 2015, pp. 150-163.

[18] G.Bai, L.Gu, T.Feng, Y.Guo and X.Chen, "Context-Aware Usage Control for Android," in *Security and Privacy in Communication Networks*, Singapore, Springer Berlin Heidelberg, 2010, pp. 326-343.

[19] M.Conti, B.Crispo, E.Fernandes and Y.Zhauniarovich, "CRêPE: A System for Enforcing Fine-Grained Context-Related Policies on Android," in *Information Forensics and Security, IEEE Transactions*, IEEE, 2012, pp. 1426-1428.

[20] S. W. M.Miettinen, A. Sadeghi and N. Asokan, "ConXsense: Automated Context Classification for Context-aware Access Control," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, New York, NY, USA, ACM, 2014, pp. 293-304.

[21] T.Guo, H.Liang, P.Zhang and S.Shao, "Enforcing Multiple Security Policies for Android System," in *2nd International Symposium on Computer, Communication, Control and Automation*, 2013.

[22] F.Rohrer, Y.Zhang, L.Chitkushev and T.Zlateva, "DR BACA: Dynamic Role Based Access Control for Android," in *Proceedings of the 29th Annual Computer Security Applications Conference*, New York, NY, USA, ACM, 2013, pp. 299-308.

[23] T.T.W.Yee and N.Thein, "Leveraging access control mechanism of Android smartphone using context-related role-based access control model," in *Networked Computing and Advanced Information Management (NCM), 2011 7th International Conference on*, IEEE, 2011, pp. 54-61.

[24] J.H.CHOI, H.JANG and Y.G.I.EOM, "CA-RBAC: Context Aware RBAC Scheme in Ubiquitous Computing Environments," *JOURNAL OF INFORMATION SCIENCE AND ENGINEERING*, vol. 26, no. 5, pp. 1801-1816, 2010.

[25] K. a. S.Park, "Context-Aware Role Based Access Control Using User Relationship," *International Journal of Computer Theory and Engineering*, vol. 5, no. 3, 2013.

[26] Y.Zhang, F. Rohrer, L.Chitkushev and T.Zlateva, "Role Based Access Control For Android (RBACA)," Boston University, MA USA, 2012.

[27] Android developers, Android Overview, Manifest Permissios, [Online]. Available: https://developer.android.com/reference/android/Manifest.permission.html

[28] M.Backes, S.Bugiel, S.Gerling and P.S.Rekowsky, "Android Security Framework: Enabling Generic and Extensible Access," in *Proceedings of the 30th Annual Computer Security Applications Conference*, New York, NY, USA, ACM, 2014, pp. 46-55.

[29] V. Juraj and P. Muska, in *Presenting Risks Introduced by Android Application Permissions in a User-Friendly Way*, Tatra Mountains Mathematical Publications, 2015, pp. 85-100.

[30] Mylonas.Alexios, T. Marianthi and Gritzalis.Dimitris, "Assessing Privacy Risks in Android: A User-Centric Approach," in *Risk Assessment and Risk-Driven Testing*,

Springer International Publishing, 2014, pp. 21-37.

[31] M. Z. Qadir, A. N. Jilani and H. U. Sheikh, "Automatic Feature Extraction, Categorization and Detection of," International Journal of Information & Network Security (IJINS), 2014, pp. 12-17.

[32] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *Security and Privacy (SP), 2012 IEEE Symposium on*, Ieeexplore, 2012, pp. 95-109.

[33] D. Barrera, H. G. Kayacik, P. C. van Oorschot and A. Somayaji, "A Methodology for Empirical Analysis of Permission-based Security Models and Its Application to Android," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, New York, NY, USA, ACM, 2010, pp. 73-84.

[34] C.Perera, A.Zaslavsky, P.Christen and D.Georgakopoulos, "Context Aware Computing for The Internet of Things: A Survey," *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, 2013.

[35] G. Abowd, P. B. A. K. Dey, N.Davies, M.Smith and P.Steggles, "Towards a Better Understanding of Context and Context-Awareness," in *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, London, UK, UK, Springer-Verlag, 1999, pp. 304-307.

APPENDICES

APPENDIX A. NORMAL PERMISSIONS

As of Android Version 6 (API level 23), the following permissions are categorized by as PROTECTION_NORMAL:

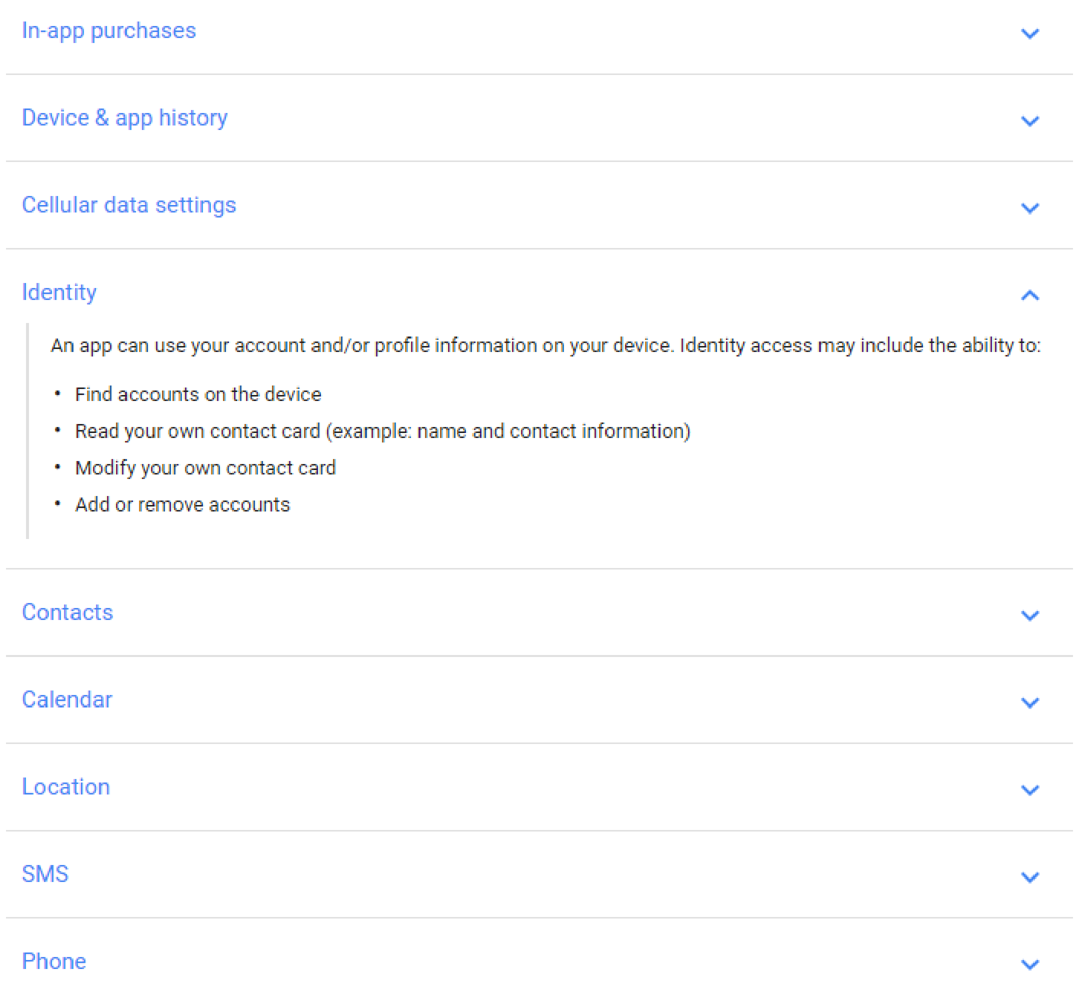
- ACCESS_LOCATION_EXTRA_COMMANDS
- ACCESS_NETWORK_STATE
- ACCESS_NOTIFICATION_POLICY
- ACCESS_WIFI_STATE
- BLUETOOTH
- BLUETOOTH_ADMIN
- BROADCAST_STICKY
- CHANGE_NETWORK_STATE
- CHANGE_WIFI_MULTICAST_STATE
- CHANGE_WIFI_STATE
- DISABLE_KEYGUARD
- EXPAND_STATUS_BAR
- GET_PACKAGE_SIZE
- INSTALL_SHORTCUT
- INTERNET
- KILL_BACKGROUND_PROCESSES
- MODIFY_AUDIO_SETTINGS
- NFC
- READ_SYNC_SETTINGS
- READ_SYNC_STATS

- RECEIVE_BOOT_COMPLETED
- REORDER_TASKS
- REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
- REQUEST_INSTALL_PACKAGES
- SET_ALARM
- SET_TIME_ZONE
- SET_WALLPAPER
- SET_WALLPAPER_HINTS
- TRANSMIT_IR
- UNINSTALL_SHORTCUT
- USE_FINGERPRINT
- VIBRATE
- WAKE_LOCK
- WRITE_SYNC_SETTINGS

APPENDIX B. DANGEROUS PERMISSIONS

Dangerous Permission Group Name	Dangerous Permissions Under the Group
STORAGE	READ_EXTERNAL_STORAGE
	WRITE_EXTERNAL_STORAGE
SMS	SEND_SMS
	RECEIVE_SMS
	READ_SMS
	RECEIVE_WAP_PUSH
	RECEIVE_MMS
SENSORS	BODY_SENSORS
PHONE	READ_PHONE_STATE
	CALL_PHONE
	READ_CALL_LOG
	WRITE_CALL_LOG
	ADD_VOICEMAIL
	USE_SIP
	PROCESS_OUTGOING_CALLS
MICROPHONE	RECORD_AUDIO
LOCATION	ACCESS_FINE_LOCATION
	ACCESS_COARSE_LOCATION
CONTACTS	READ_CONTACTS
	WRITE_CONTACTS
	GET_ACCOUNTS
CAMERA	CAMERA
CALENDAR	READ_CALENDAR
	WRITE_CALENDAR

APPENDIX C. ANDROID PERMISSION GROUPS



In-app purchases

Device & app history

Cellular data settings

Identity

An app can use your account and/or profile information on your device. Identity access may include the ability to:

- Find accounts on the device
- Read your own contact card (example: name and contact information)
- Modify your own contact card
- Add or remove accounts

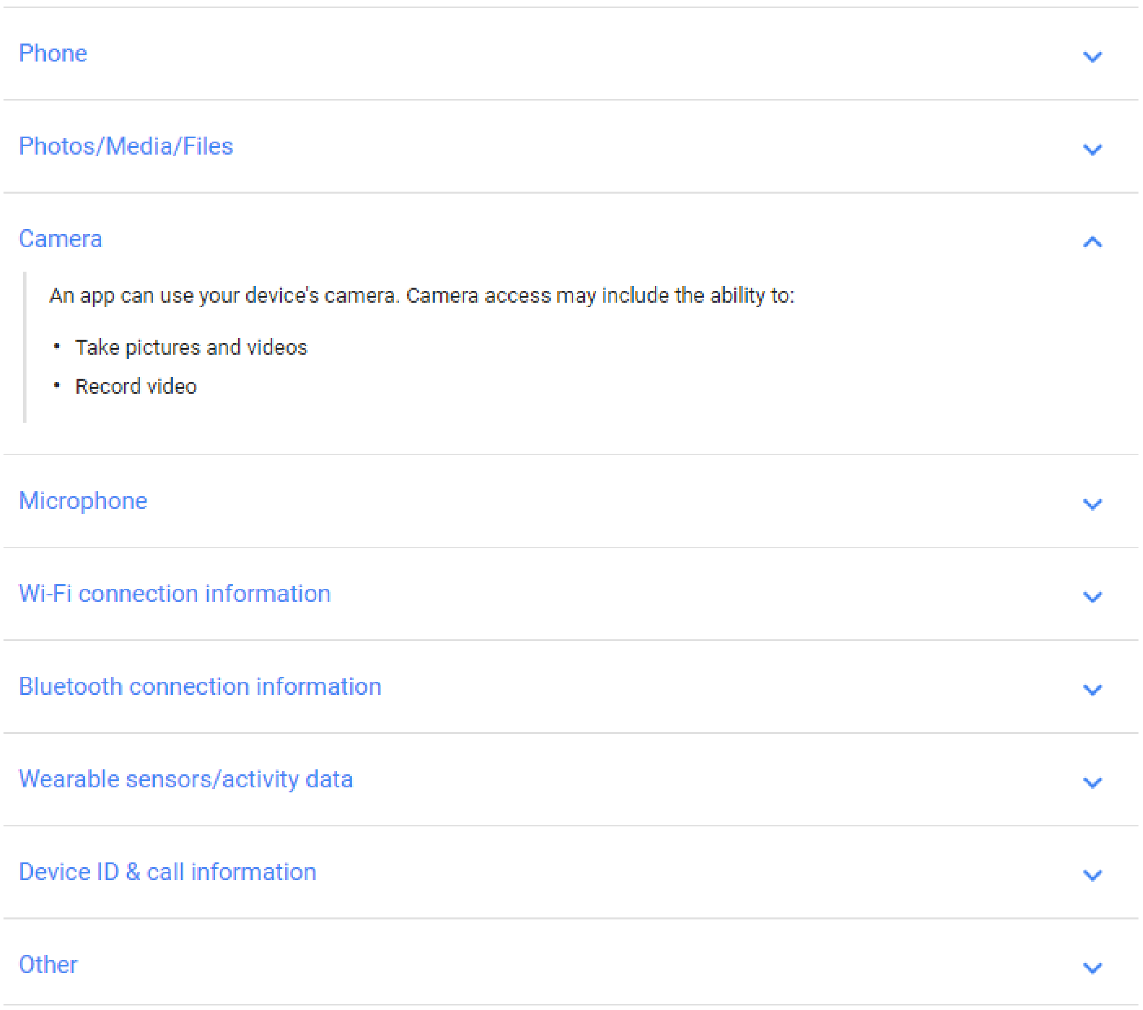
Contacts

Calendar

Location

SMS

Phone



Phone

Photos/Media/Files

Camera

An app can use your device's camera. Camera access may include the ability to:

- Take pictures and videos
- Record video

Microphone

Wi-Fi connection information

Bluetooth connection information

Wearable sensors/activity data

Device ID & call information

Other