User Privacy and Database Security using Context based Access Control in Android Devices: A Survey

Charuli Patil[1], Prof. Kiran K Joshi[2]

[1]M.Tech Student, Dept. of Computer Engineering and IT, VJTI, Mumbai, Maharashtra, India
[2]Assistant Professor, Dept. of Computer Engineering and IT, VJTI, Mumbai, Maharashtra, India

---***---

Abstract - *Existing projects are based on granting and retrieving permissions to applications. When a user installs an application it grants permission to it and once granted it can use it without any limit, unless the user retrieves the permission manually. This is the reason for security to privacy issues because a harmless permit can sometimes be harmful in some cases. Security risks are highly dependent on the current situation and, while accessing a particular service is safe in some contexts, while it may violate privacy in another context. One of the solutions to this contextual control problem is the level of consent that allows the granting of strong and denied consent depending on the specific situation. However, manually adjusting the policies becomes more complex for the user as there are a number of policies that will need for the application to run properly. Therefore, we try to propose a context-based application that can provide flexibility to provide and recover permissions based on different contexts. We use the Android platform to demonstrate our work. In this model, Android applications are assigned contexts, where the contexts grant or revoke different permissions and permission-related conditions [1].*

***Key Words*: Android, access control, context based access control, user privacy, database security, location.**

1. INTRODUCTION

The changing technology in mobile phone has introduced new smart technology that offers emerging yet sophisticated capabilities. In addition, smart phones are increasing the access to devices works with apps such as GPS, SMS, Wi-Fi etc. As a result, several challenges occur mainly in security issues. The most important feature is to limit the performance of users using apps and its services. In the meantime, the existing security and privacy control system hosts a built-in and imperfect security model. Most current applications work with device-level security that is the basic of each application, especially for installation. A user installing third-party applications thinks that the applications will not misuse the device resources. Similarly, if a user wants to use that program, he or she must grant all permission requests. This whole or empty decision leads to a built-in access control industry. In addition, as long as the user grants the permissions, there is no way to limit or revoke these permissions based on user activity. For example, a user may want to limit the amount of SMS sent on a day for saving a bill using status information such as access patterns. To address these issues, our model proposes an access control mechanism managed by the access control component i.e. Context-Related Role-Based Access Control (CtRBAC) [2]. Contexts are defined as a set of actions or obligations associated with a specific function. Policy restrictions are defined according to the resources, services, and permissions that are granted to apps.

The main issue with the latest operating system is that once permissions are granted to an application, it always has the right to access the services without taking prior permissions from the users [1]. Operating systems do not support a change in application permissions based on user context. This perspective of application permissions is very risky in case of mobile applications. The threat arises when an application acts as a virus and uses device resources to spy or leak the user's personal data without consent. Even when the users are carrying smartphones in public places, this may expose their data while being unaware of the harmful acts going on their devices [6].

Many mobile OSs, including Android, store personal information and provide application APIs, which can be used to access system-managed data or to manage its own database [7]. For Android, it stores data precisely, but the data may be unintentionally charged with several risks. The security of content information created by the application lies with the developer of the app, while the portable OS only provides some security features such as fragmentation and access control. Google introduced Full Disk Encryption (FDE) for Android having versions 3.0 and above it. It provides confidential encryption and decryption that works for the file system for all device read / write functions. But, FDE protects data only when data it is encrypted. When a user unlocks an Android device, all information on the device is decrypted and announced. That means, FDE protects data only on a locked device from offline attacks and not while being active. Once a device is unlocked, anyone with proper permissions can access, use and modify all the data in the database even if FDE is used [8].

1.1 Background

Mobile applications security focuses on preventing applications from accessing sensitive data and resources, but most of them have no effective means of enforcing those restrictions. Most of the work till now is focused on developing policies that apply only to the whole system and not on different contexts. Basically, a Context-based access control requires the identification of contexts in a system. These contexts have policy restrictions.

Policies are contextual and adjustable by the device user. The most common contexts defined are location and time. We have implemented CBAC policies on the Android operating system. When the user configures the device policies according to context, the policies will be applied whenever the user is using the application.

2. LITERATURE SURVEY

2.1 User Privacy

User grant permissions to applications. But these permissions granted are harmless at a time but can be harmful in some cases. Permission to access camera in normal areas (Home, Public Area etc.) is harmless but in a military or sensitive area it can be dangerous. These permission therefore should not be granted in such areas.

Applications shouldn’t be able to disclose the location or time of the device to third party sources. Applications should not allow third party apps to fake a location. Applications should not generate a fake time and gain access to certain content that is not allowed. [1]

Android up to marshmallow was like a mess, basically we had to accept a list of permissions and if we don't accept all, we won't be able to use the application. After marshmallow from Android Nougat (v8) onwards we got a granular control like we can turn off permissions one by one but the thing is lot of apps are using those permissions and a lot of users don't understand why it's important to grant extra permissions that the applications doesn't need. There is a fast growth in developing apps which are based on location detection. However, private data gets leaked due to untrusted location servers [4].

Nowadays it's important that we control our data so we can maintain the level of privacy we want. In Android we take privacy very seriously with each release. We’re building an increasingly privacy centered platform in which, privacy sensitive decisions must not lie entirely on our users instead it must be shared between the users and the Android system. Our goal is to help users understand that they're controlling their data, we help them make informed choices about the data they grant access to and whom they share the data within Android. In Android 11, developers have evolved the permission system to give our users even more control. They are now giving users the options to temporarily grant permission valid for only a single use by an application. The permission granted when the application is in foreground is either as a visible activity or as a foreground service. Granted permissions expire in a short period after the application is moved to background. [3]

Applications always check whether they hold a given permission before performing any action that requires it. This stage should not be cached or persisted to storage as it can get out of synchronization with system state. Furthermore applications should fail gracefully and offer an appropriate subset of their functionality in cases where the user declines a permission. In our application we must first obtain a program location permission that is one of Access Coarse location and Access Fine location [3].

If we request background location and any other permission at the same time the system shows error for apps targeting android 11. Application must clearly explain the features that require background location by an end context UI. This UI should explain why a particular feature needs background location access to the user to permit access. The user is then directed to the systems setting to complete the process. But even after an application is granted the background location permission, users can change their mind and deny access from the settings UI for the next useful statistic. The first improvement is to foreground services in Android 11, we're adding two new foreground service types for camera and microphone access. We're making these changes to better protect the sensitive data and to provide users data more transparently, about how this data is accessed if our applications is targeted. Any foreground service that accesses microphone or camera data must declare this types using the foreground service type manifest attribute [5].

2.2 Database Security

When we are trying to build an app, securing data is really important. The biggest issue is that it doesn't use Android key store which is a safe way to manage keys and basically prevent someone from getting access to the data that we've encrypted. Google has launched Jetpack security, which basically allows us to easily encrypt user’s data, files, and shared preferences without having to really understand the ins and outs of security. If we're building an applications that runs on a rooted or compromised device, it means the user is present on this device and the file system is unlocked. Once the file system is unlocked and even though we have full disk encryption, the data is now available to an attacker or someone that's getting the device. Secondly, attacker might actually have data that we don't want others to see. So if we have API keys, tokens, things like that to authorize services, we probably don't want those to get leaked, and if we probably don't want anyone to get access to that because technically, they could start pretending to be user, and start using our quota and getting free things that we have to pay for [8].

The first important thing is key management. So if we have like a car key, house, or apartment key, that's something we can keep on and kind of secure it. But with phones or with data, we can't really just physically make sure that it's safe. If our device is compromised in some or the other way, and if someone has rooted it, or if we've actually lost our device and it's been stolen, we don't really know a way of protecting that key. Even if it's in pocket, technically if something happened, that key could be used against us in some or the other way. So we get around this by using something called as the "Android keystore." It is hardware backed, that means it runs in a separate memory space on the device, which means even though our applications has access to the keys, we actually don't know what the key material is, so we can't get the key ourselves [7].

The Jetpack security, is built so that when we create a key, it creates sensible defaults so we don't have to understand all the ins and outs of security. We can also use features like strong box which is a separate piece of hardware on the device. That is a separate chip, so there will be a slight performance implication from using it because it is a little bit slower technically, but it's also a very safe way to manage our data. All the operations are going to happen in this separate memory space, which means that if we encrypt-decrypt signing, things, key is not leaked into our application. So if someone is spying on what you're doing in some way, you're going to be safe. So in Jetpack security, to get started, out of the box, we provide a master keys class which basically allows us to create a key and enter a keystore. And that means the default is using AES (Advanced Encryption Standard)-256. This is the normal sized key that we'd use. It's safe and it's well known.

We use the block mode GCM, and we do no padding. So if we want to encrypt a small amount of data that is less than equal to the key size, we don't really need any padding or blocking modes. One thing that's important here is that a lot of the attacks and challenges happen with encryption when we have data that's longer than the key length. So that's where padding and block modes come in place. Jetpack security takes advantage of this also. We just need to make sure we either have a key in the keystore or that it's been generated safely, and that we have an alias. So there's a string that's going to be the key to look this up and use later. If we do want to set more advanced settings, so that we want to use some sort of key authorization, to actually make sure that there's a user present, there's other options here. Currently, we enforce that the key must be 256 bit. We used GCM and no padding.

The other options we see below are like, user authentication, set strong box backed, and user required are features that are totally optional. We can have a key that is where the user has to unlock the device using the timed authentication. So the best way to implement this is by using biometric prompt. We're doing time based, so we're not going to pass a crypto or a signature there. We are just going to do this, and we have this authentication callback which we can see up here. If the authentication is successful, then we can use our key. If it's not, then we can prompt the user with the error. So once we have the key, then we can do stuff with it. So the first thing that we can do is use an encrypted file, it has an API similar to file input stream and file output stream, but it seamlessly encrypts and decrypts as you go along.

Other feature of Jetpack security other than files is encrypting shared preferences. So if we have sensitive data, like API keys, we can use the out of the box shared preferences and shared preferences editor interfaces which allows to basically understand most people well, most developers already know how shared preferences works, and this allows us to take advantage of that using keys and values [9].

Jetpack security uses a Google open source library called "Tink." Tink has cross-platform security for a lot of different mobile platforms, and it's something we use because it provides a way and a method of ensuring that-- there's a lot of different versions of Android, there's a lot of different OEMs out there. There's different versions of SSL that are implemented. Tink does provide a lot more features, and it does have an Android extension as we mentioned as well, because we actually use it. One thing to mention is that something called "queue rotation." So even though that we have a master key-- and we never explained why it was a master key-- the reason is because each file or shared preferences object, or each type of a primitive-- which a primitive could be an algorithm or something like that-- algorithm key size in Tink. And we create key sets of everything.

It create an AES-256 key to encrypt a file, it will automatically create a shared preferences key set under the covers. And what this allows us to do is every time we use Tink for some reason, if you're using it under the covers and we need to change or rotate a key. Let's say for some reason, the keys that are saved in shared preferences, we need to change them, they've been compromised in some way, we can rotate them which means it'll actually add a new key to the key set that's for that specific primitive or that specific file, and allow us to migrate all the files or data that's been saved into a new key set. So that's something called "queue rotation." Tink also supports features like message signing, MACs, hybrid encryption. If we don't know what hybrid encryption is, it's using both public and-- asymmetric and symmetric encryption, like an RSA key with an AES key to encrypt larger sets of data, but also be able to share that by sharing a public key with another party.

3. PROPOSED SYSTEM

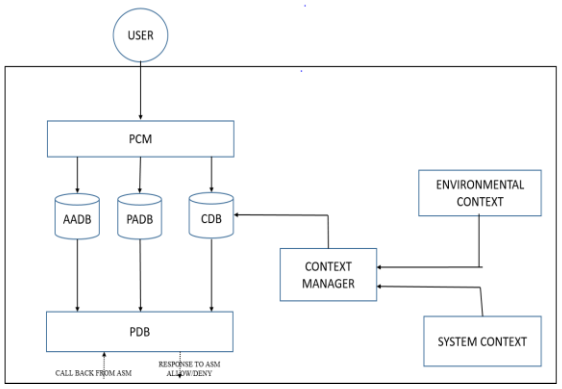

Fig 3.1 Context-aware Android role-based access control (CA-ARBAC) components

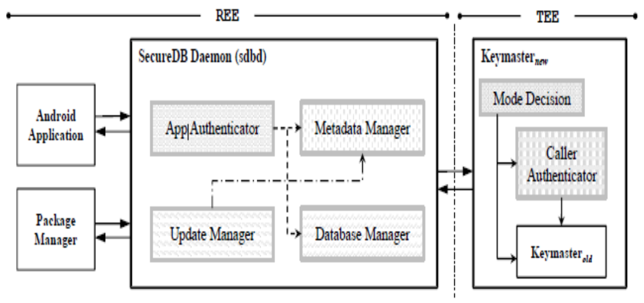

Fig 3.2 Database Architecture

4. CONCLUSION

We will develop an APS access control application to protect users' privacy. Our application allows permissions based on predefined contexts. Our system is a variant of CA-RBAC designed that will have a set of Android permissions, which are granted to applications. We will introduce the building of a secure data structure on Android. Our architecture will provide data privacy by allowing only the authorized business to access the associated database. Only sdbd can access a unique key in TEE and only an authorized application can request a purchase on sdbd. Therefore, for such a validation process, only the most patented applications can access that program.

REFERENCES

[1] B. Shebaro, O. Oluwatimi and E. Bertino, "Context-Based Access Control Systems for Mobile Devices," in *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 2, pp. 150-163, 1 March-April 2015.

[2] T. T. Yee and N. Thein, "Leveraging access control mechanism of Android smartphone using context-related role-based access control model," *The 7th International Conference on Networked Computing and Advanced Information Management*, Gyeongju, 2011, pp. 54-61.

[3] Saksham Chitkara, Nishad Gothoskar, Suhas Harish, Jason I. Hong, and Yuvraj Agarwal. 2017. Does this ApplicationsReally Need My Location? Context-Aware Privacy Management for Smartphones. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 1, 3, Article 42 (September 2017)

[4] Rohan J. Patil, Prof. K.K. Joshi and Prof. Sowmiya Raksha, "Analysis On Preserving Location Privacy," International Journal Of Advance Research In Computer Science And Software Engineering, vol. 3, no. 3, pp. 562-566, March 2015.

[5] E. Tomur, Department of Computer Engineering, ˙Izmir Institute of Technology, Urla, ˙Izmir, Turkey. , "CA-ARBAC: privacy preserving using context-aware role-based access control on Android permission system" , Published online 24 January 2017 in Wiley Online Library (wileyonlinelibrary.com)

[6] B. Ren, C. Liu, B. Cheng, S. Hong, J. Guo and J. Chen, "EasyPrivacy: Context-Aware Resource Usage Control System for Android Platform," in *IEEE Access*, vol. 6, pp. 44506-44518, 2018, doi: 10.1109/ACCESS.2018.2864992.

[7] Simone Mutti, Enrico Bacis, Stefano Paraboschi, 'An SELinux-based Intent manager for Android', DIGIP — Universit`a degli Studi di Bergamo, Italy fsimone.mutti, enrico.bacis, paraboscg @ unibg.it

[8] J. H. Park, S. Yoo, I. S. Kim and D. H. Lee, "Security Architecture for a Secure Database on Android," in *IEEE Access*, vol. 6, pp. 11482-11501, 2018.

[9] L. Hong-Yue, D. Miao-Lei and Y. Wei-Dong, "A Context-Aware Fine-Grained Access Control Model," *2012 International Conference on Computer Science and Service System*, Nanjing, 2012, pp. 1099-1102.