IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 1, Ver. IV (Jan – Feb. 2016), PP 51-57
www.iosrjournals.org

Development of Security Based Reserved Agreement Blocker for Smartphone

Ms. Shirin Ayisha Maryam. M[1], S. Arogya Swarna[2]
[1]*Assistant Professor/CSE S. Veerasamy Chettiar College of Engineering Puliangudi*
[2]*Assosciate Professor and HoD/PG CSE S. Veerasamy Chettiar College of Engineering Puliangudi*

***Abstract**: Smart phones are very effective tools for increasing the productivity of business users. With their increasing computational power and storage capacity, smartphones allow end users to perform several tasks and be always updated while on the move. In Existing system, the location-based policy systems (LPS) are not accurate enough to differentiate between nearby locations without extra hardware or location devices. In most cases, such systems assume the context as given without providing or evaluating context detection methods of mobile devices. In order to overcome this issue, context-based access control (CBAC) mechanismhave been proposed. By means of CBAC, many privileges can be dynamically granted or revoked to applications based on the specific context of the user. The CBAC framework utilizes context sensing and machine learning to automatically classify contexts according to their security and privacy-related properties. It provides protection against device misuse using a dynamic device lock and protection against sensory malware. We have performed several experiments to assess the efficiency of our access control mechanism and the accuracy of context detections.*

***Keywords**: CBAC-Context Based Access Control, LPS-Location based policy system*

I. Introduction

Android mobile devices are becoming a popular alternative to computers [4]. Our daily lives become more and more dependent upon smartphones due to their increased capabilities [5]. Smart phones are used in various ways from payment systems to assisting the lives of elderly or disabled people[3]. Security threats for these devices become increasingly dangerous since there is still a lack of proper security tools for protection.

Android emerges as an open smartphone platform which allows modification even on operating system level. Therefore, third-party developers have the opportunity to develop kernel-based low-level security tools which is not normal for smartphone platforms [4]. Android quickly gained its popularity among smartphone developers and even beyond since it bases on Java on top of "open "Linux in comparison to former proprietary platforms which have very restrictive SDKs and corresponding APIs The rise in the number of tasks performed on smartphones means sensitive information is stored on the devices.

II. Related Works

Yury Zhauniarovich,Giovanni Russello,Bruno Crispo, Earlence Fernandes[2014] **"MOSES: Supporting and Enforcing Security Profiles on Smartphones"** IEEE transactions on dependable and secure computing, vol. 11, no. 3,pp 211 - 223

Smartphones are very effective tools for increasing the productivity of business users. With their increasing computational power and storage capacity, smartphones allow end users to perform several tasks and be always updated while on the move.
Companies are willing to support employee-owned smartphones because of the increase in productivity of their employees. However, security concerns about data sharing, leakage and loss have hindered the adoption of smartphones for corporate use. In this paper we present MOSES, a policy-based framework for enforcing software isolation of applications and data on the Android platform. In MOSES, it is possible to define distinct Security Profiles within a single smartphone. Each security profile is associated with a set of policies that control the access to applications and data.
[2] Jaeyeon Jung Seungyeop, David[2012]**"Enhancing Mobile Application Permissions with Runtime Feedback and Constraints"** Published by ACM Article , pp 45-50

They report on a field study that uses a combination of OS measurements and qualitative interviews to highlight gaps between user expectations with respect to privacy and the result of using the existing permissions architecture to install mobile apps. Most of our participants expected advertising and analytics behavior, yet they were often surprised by applications' data collection in the background and the level of data sharing with third parties that actually occurred. Given participant feedback, we propose platform support to reduce this "expectation gap" with transparency of data usage and constrained permissions.

[3] ShaikhSarim Asib1 and Prof. Anuja K. Pande[2014] **"Study on Enhancing User Privacy on Android Mobile Device via Permission Removal"** VOLUME-2, SPECIAL ISSUE-1,pp 276-279

Android mobile devices are becoming a popular alternative to computers. The rise in the number of tasks performed on mobile devices means sensitive information is stored on the devices. Consequently, Android devices are a potential vector for criminal exploitation. Existing research on enhancing user privacy on Android devices can generally be classified as Android modifications. These solutions often require operating system modifications, which significantly reduce their potential. This paper proposes the use of permissions removal, wherein a reverse engineering process is used to remove an app's permission to a resource. The repackaged app will run on all devices the original app supported. These findings that are based on a study of seven popular social networking apps for Android mobile devices indicate that the difficulty of permissions removal may vary between types of permissions and how well-integrated a permission is within an app.

III. Existing system

Existing System work has focused on developing policy systems that do not restrict privileges per application and are only effective system-wide. Also, existing policy systems do not cover all the possible ways in which applications can access user data and device resources. Finally, existing location-based policy systems (LPS) are not accurate enough to differentiate between nearby locations withoutextra hardware or location devices. In most cases, such systems assume the context as given without providing or evaluating context detection methods of mobile devices. User-specified policies have the potential to correctly reflect the user's true security and privacy preferences, but the amount of work required to set up and maintain a comprehensive set of context-dependent policies is high .A study concerning location sharing policies showed that the initial accuracy of location disclosure rules specified by users was only 59% and improved to 65% after users modified the rules based on a review of concrete enforcement decisions resulting from these rules.

Disadvantages Of Existing System

- Not sufficient to capture the highly dynamic and highly personal nature of a user's context.
- Need External devices to locate devices.
- User need to share their privacy.
- Threat arises when a device application acts maliciously and uses device resources to spy on the user or leak the user's personal data without the user's consent.

Proposed Research

A context-based access control (CBAC) mechanism has been proposed for Android systems that allow smartphone users to set configuration policies over their applications' usage of device resources and services at different contexts. We present a context model used to extract context features reflecting the familiarity of contexts and the persons in the context. The context features are input for the Classifier and used for classifying contexts as having high or low privacy exposure and/or risk of time management. Here we use a scheduled protection for smartphone. The access permission of application is control by means of time. The access permission can denied or enabled by time. The contexts are not predefined in this project. Contexts are user defined in context based scheduled lock

Advantages' Of Proposed System

- Time Based Access control provided us the environmental profits.
- No need to share the user details.
- Applications should not be able to fake the location or time of the device.
- Can develop securer and more acceptable applications for end users.

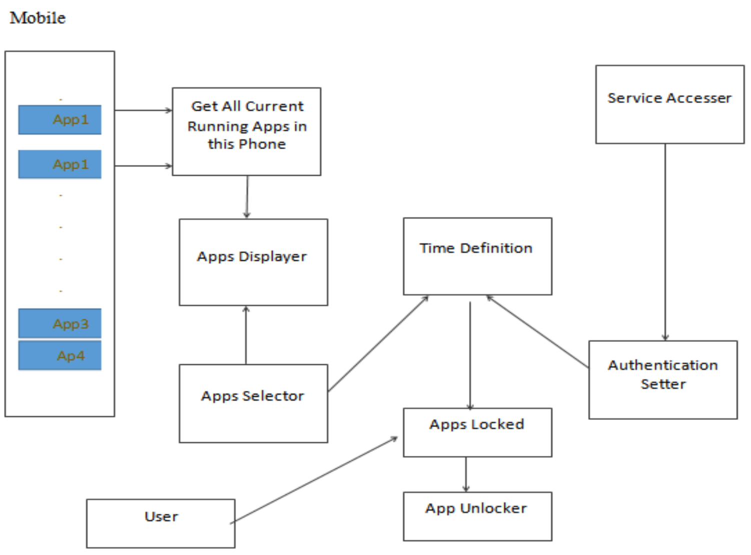


Fig 4.1 system architecture

Data Flow Diagram
Dfd Level 0

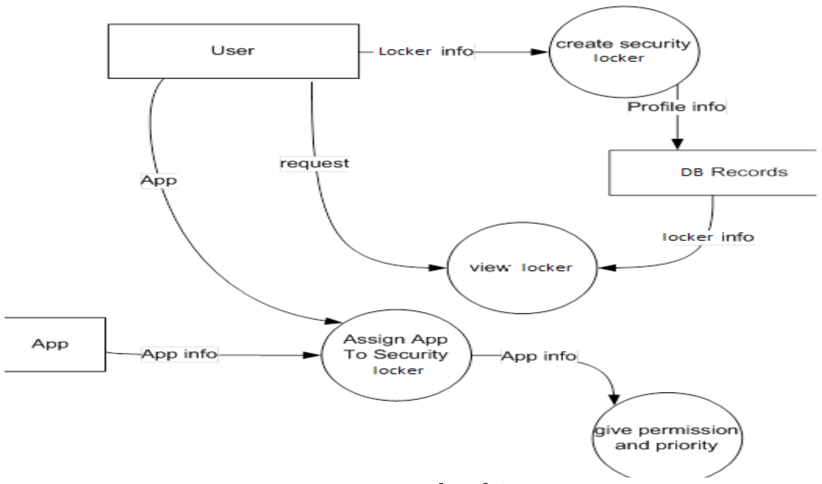


Fig 5.1 DFD level 0

A. Dfd Level 1

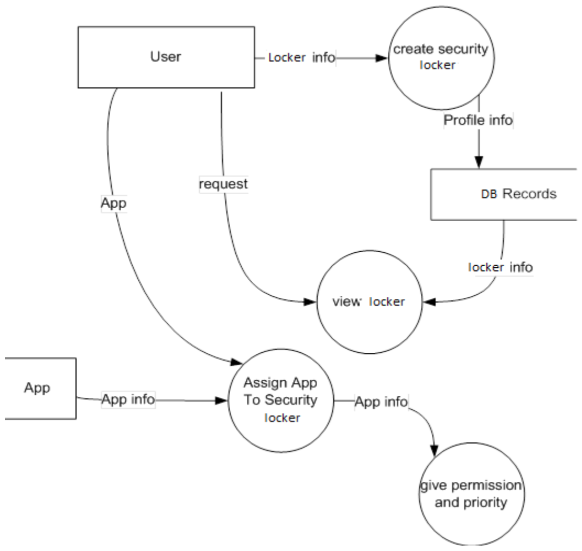


Fig 5.2 DFD level 1

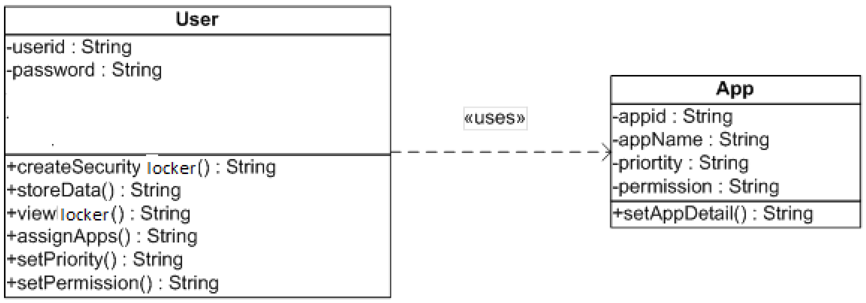
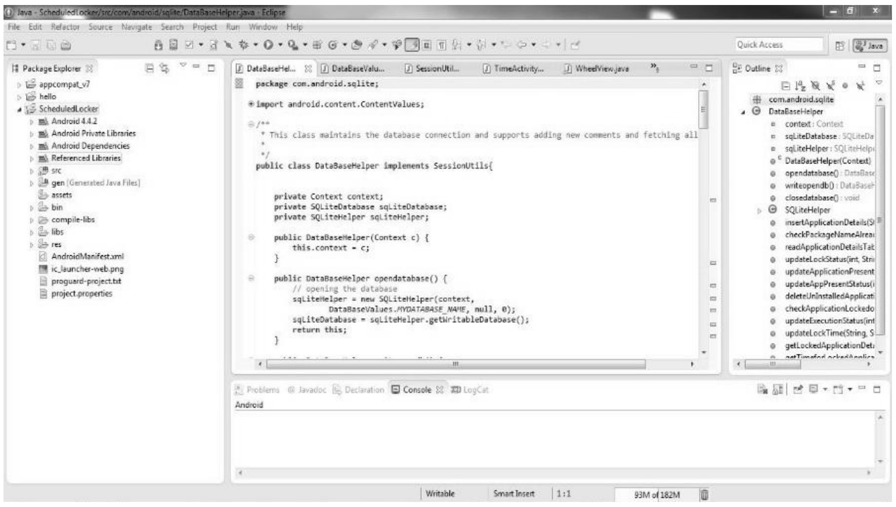


Fig 5.3 class diagram

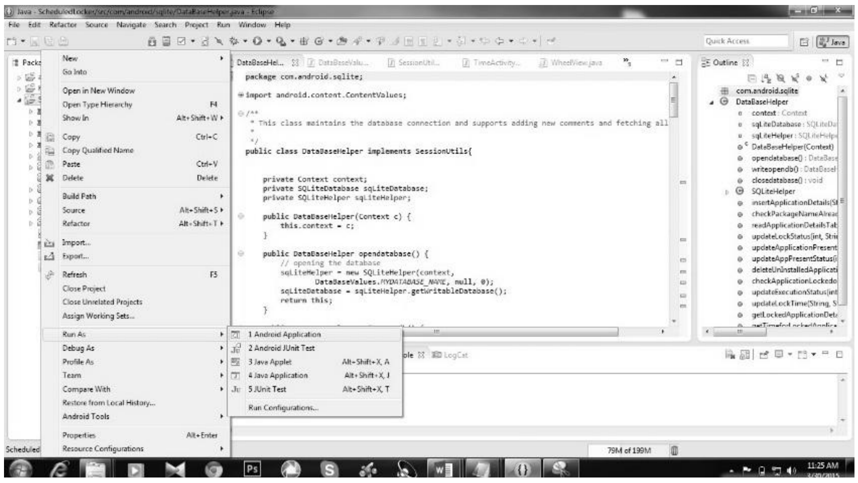
Experimental Environment And Output

The Proposed CBAC Mechanism is implemented in JAVA 1.7.The experiment is initially tested in the andriod 4.2.2 Operating system on mobile.It can develop the securer and more acceptable applications for end user



```
package com.android.sqlite;

import android.content.ContentValues;

/**
 * This class maintains the database connection and supports adding new comments and fetching all
 *
 */
public class DataBaseHelper implements SessionUtils{


    private Context context;
    private SQLiteDatabase sqLiteDatabase;
    private SQLiteHelper sqLiteHelper;

    public DataBaseHelper(Context c) {
        this.context = c;
    }

    public DataBaseHelper opendatabase() {
        // opening the database
        sqLiteHelper = new SQLiteHelper(context,
                DataBaseValues.MYDATABASE_NAME, null, 0);
        sqLiteDatabase = sqLiteHelper.getWritableDatabase();
        return this;
    }
```

Fig 6.1.eclipse sdk



```
package com.android.sqlite;

import android.content.ContentValues;

/**
 * This class maintains the database connection and supports adding new comments and fetching all
 *
 */
public class DataBaseHelper implements SessionUtils{


    private Context context;
    private SQLiteDatabase sqLiteDatabase;
    private SQLiteHelper sqLiteHelper;

    public DataBaseHelper(Context c) {
        this.context = c;
    }

    public DataBaseHelper opendatabase() {
        // opening the database
        sqLiteHelper = new SQLiteHelper(context,
                DataBaseValues.MYDATABASE_NAME, null, 0);
        sqLiteDatabase = sqLiteHelper.getWritableDatabase();
        return this;
    }
```

Fig 6.2 Running project as android application

Fig 6.3 Android emulator in eclipse

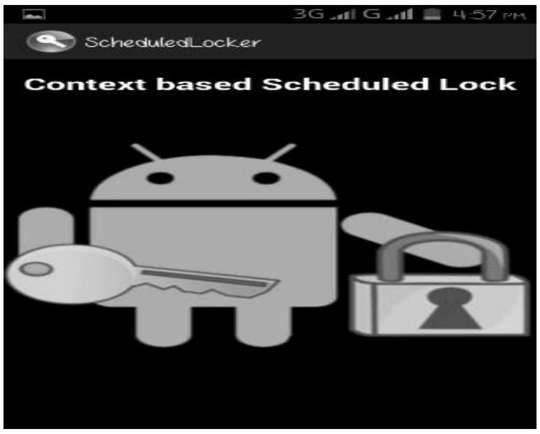


Fig 6.4Running of application

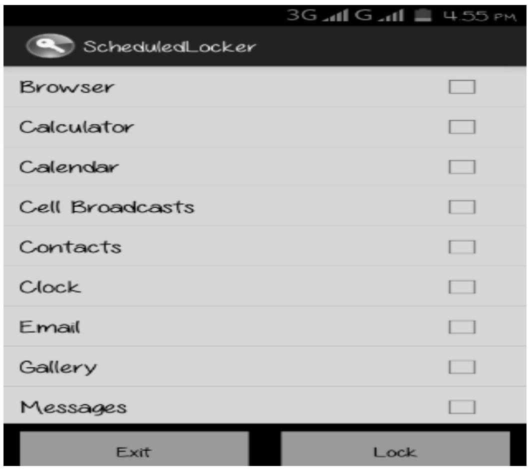


Fig 6.5 App displayer

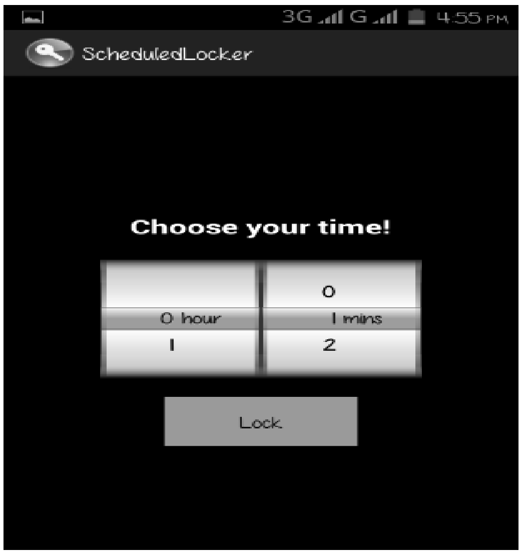
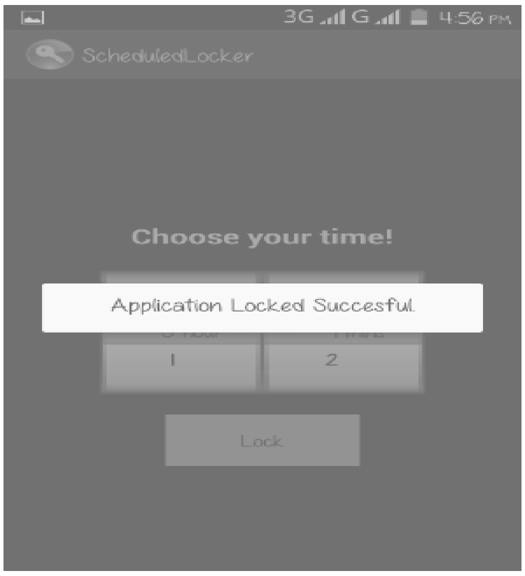


Fig 6.6.Time setter



Application Locked Succesful

Fig 6.7. Apps permission denier

Cbac Evaluation

In this section, we report on the thorough experiments we ran to evaluate the performance of CBAC. For all the experiments, we used a Google Nexus S phone

Energy Overhead

To measure the energy overhead produced by CBAC, we performed the following tests. We charged the battery of our device to the 100 percent. Then, every 10 minutes we run four system applications (sequentially) via a monkey runner
[7] Script: Calculator, Browser, Contacts and Email. For each of them the script performed common operations representative for the applications (multiplication of numbers in case of Calculator, browsing several webpages in case of Browser, calling a number and creating an account in case of Contacts, and composing and sending a email in case of Email application). Each experiment lasted for a total of 120 minutes. We executed this experiment for three types of systems: Stock Android, CBAC without SP changes, and CBAC with SP changes (the system switched between two profiles every 20 minutes). During each experiment, every 10 seconds, our service measured the level of the battery and wrote this value into a log file. For each of the three considered systems, we executed the test 10 times and averaged the obtained values. The results of this experiment are reported in Fig. 6.7. We note that the curves for the three considered systems behave similarly. This shows that the fact that CBAC is just running, or even switching between context does not incur a noticeable energy overhead.

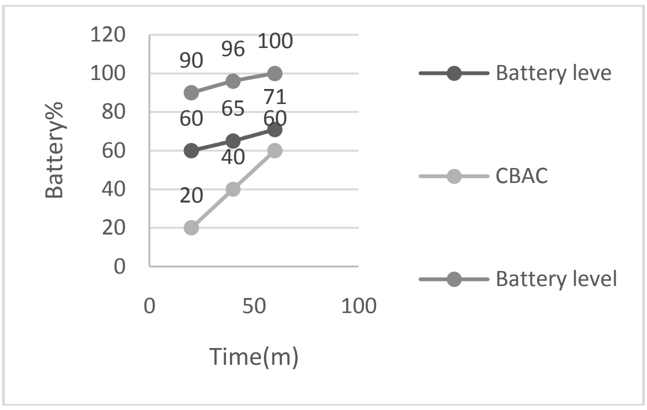


Fig7.1 Comparative analysis of Energy Overhead between LPC and CBAC System

During each experiment, every 10 seconds, our service measured the level of the battery and wrote this value into a log file. For each of the three considered systems, we executed the test 10 times and averaged the obtained values. The results of this experiment are reported in Fig. 7.1. We note that the curves for the three considered systems behave similarly. This shows that the fact that CBAC is just running, or even switching between contexts does not incur a noticeable energy overhead

IV. Conclusion

From these results, it can be determined that it is indeed possible to remove permission requests from apps via reverse engineering and result in a usable and privacy friendly app. Removing the READ_PHONE_STATE permission from Face book and LinkedIn had a similar result to removing location access from them. With Tango Text, which required this permission in order to identify the phone, the app would instead display an error message. This means that successful removal of this permission would depend on the app in question.

Future Work

All the research undertaken in this paper on permissions removal was manually completed; therefore future research recommendations include automating this process. By using heuristic methods, it may be possible to locate sections within the source code where permissions resources are used automatically. The decompilation, modification and recompilation process could then be completely autonomous. This leads onto a further future work or research that could be undertaken where this automated system would be implemented onto an Android device. This would result in a self-functioning system that could enhance the privacy of apps on the device.

References

[1]. MOSES: Supporting and Enforcing Security Profiles on Smartphones(2014)Yury Zhauniarovich, Giovanni Russello Mauro Conti, Member, IEEE,Bruno Crisp and Earlence Fernandes. ieee transactions on dependable and secure computing, vol. 11, no. 3, may-june 2014

[2]. M. Oleaga, 2013, "OS vs. Android Market Share 2013: Google Mobile Platform Dominating Apple Worldwide in March Figures", http://www. latinospost.com/articles/15039/20130322/ios-vs-android-market-share-2013-google- mobile-platform-dominating.htm, accessed 25 March 2013.

[3]. O.Hou, 2012, "A Look at Google Bouncer", http://blog. trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/, accessed 14 April 2013

[4]. FY. Rashid, 2012, "Researchers upload dangerous app to Google Play store", http://www.scmagazine.com.au/News/310192,blackhat-researchers- upload-dangerous-app-to-google-play-store.aspx, accessed 14 April 2013

[5]. Y. Zhou, X. Zhang, X. Jiang & V. Freeh, "Taming information-stealing smartphone applications (on Android)", TRUST 2011, pp. 93-107.

[6]. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev & C. Glezer, "Google Android: A Comprehensive Security Assessment", IEEE Security & Privacy Magazine, Vol. 8, no. 2, 2012, pp. 35-44.

[7]. Monkey runner, http://developer.android.com/tools/help/ monkeyrunner_concepts.html, 2014