ISSN:0975-9646

Dynamic Profile Activation for Enhancing Security in Android Devices

Venkata Sai Abhishek Gogu, Venkata Sai Teja and Jahnavi Agepati

Students, Sree Vidyanikethan Engineering College,Tirupati, Andhra Pradesh, INDIA -517501.

***Abstract*— It's been only a decade with its advent yet smartphone technology is so successful that business and their staff have trouble imagining a day without them. It helps them stay connected and updated everywhere. These smartphones support tens of thousands of apps and still growing every day. Companies allow employee owned smartphone devices as their performance can be further uplifted with such applications. But it induces security worries about data breach, scams, theft that concerns the use of smartphone devices. Here in this paper we try to overcome that by developing a policy based framework pertaining to necessary separation of applications and data on the Android platform and define a set of separate profiles in a single smartphone device. Each and every profile has set of policies associated with it. The rules associated with the profile can be altered at our convenience and they automatically switch from one security profile to another based on our location. The result of the full implementation of the security profiles confirms the reliability of the system**

Keywords— Activity Manager Service,Authentication, Context Detection, Data isolation.

I. INTRODUCTION

As the whole world is going into the new phase of technological performance, our needs have become more sophisticated. One of the finest and biggest technological advancements is the creation of smart phone. Smart phones are changing the way people around the world live, work, study, communicate and share information Everyone try to have a smart phone as having a smart phone is beneficial we can do company work on MS Office and can go through our e-mails anytime and anywhere and many more things. As we look for convenience, we respect the devices which combine multiple features and which give us more mobility. The Android OS is such most popular platform with 82 percent market share. On the other hand, we need security, quality, and effectiveness of these smartphones to be maintained to the highest level possible. Companies these days allow their employees to bring their own device policy. More and more companies nowadays are providing mobile versions of their desktop applications as allowing access to enterprise services with smartphones increases employee productivity. Many manufacturers these days are following this trend by producing smartphones that support two subscriber identification modules (SIMs) at the same time. Inspite of the advantages of using a smartphone in the company environment there are certain security issues regarding their usage which may lead to company losses. For example, malicious applications may grant permissions to MMS,emails and SMS. In such scenarios data stored in the smartphone containing company confidential data is the only reason for huge losses. This poses serious security concerns to sensitive corporate data, exceptionally when the generic security mechanisms offered by the platform are not sufficient to protect the users from such attacks. A possible solution to the above specified problem is isolation. By having applications and data related to work separated from recreational applications and private/personal data. Within the same device, we can have separate security environments. A security environment will be restricted to sensitive/corporate data and trusted apps only. A second security environment could be used for entertainment where third-party games can be allowed for home purposes. But the key concept here is applications from the second environment should not able to access data of the first environment thus the risk of leakage of sensitive information can be greatly diminished. This application can be done by means of the concept Of virtualization.

Through virtualization different instances of an OS can run separately on the same device. But the concept of virtualization is more effective in full-fledged devices (PC and servers), it is too resource demanding for embedded systems such as smartphones. So we try another approach of Para virtualization. Unlike full virtualization where the guest OS is not aware of running in a virtualized created environment, unlike in para-virtualization it is necessary to modify the guest OS to boost performance. Yet the Para virtualization concept i still under development for smartphone devices. Even if it existed we cannot fully make use of virtualization concept to create isolation as it suffers from having a coarse grained approach. We cannot define the environment and security policies as we need, and also the switching among environments always require user interactions and it could take a significant amount of time and power.

1.1 TERMS OF ANDROID SECURITY:

As we are trying to develop secure profile environments there are certain terms and concepts which we should be aware of. Activities represent a user interface; Services execute background processes; Broadcast Receivers are mailboxes for communications within components of the same application or belonging to different apps; Content Providers store and share applications data. Application components communicate through messages called Intents.

Android implements two levels of enforcement. One at the Linux kernel level and the other at application framework level. At the Linux kernel level Android is a multi-process system. During installation, an application is assigned with a unique Linux user identifier (UID) and a group identifier

(GID). Thus, in the Android OS each application is executed as a different user process within its own isolated address space In Android, by default, all the files in the user's home directory can be read, written and executed by the owner and the users from the same group as the owner. All other users cannot work with these files. So as different applications by default have different user identifiers files created by one application cannot be accessed by another. In the application framework level, Android provides access control through the inter-component communication (ICC) reference monitor. It gives mandatory access control (MAC) enforcement on how applications access the different components of it. In the simple words, protected features are assigned with unique security labels known as permissions.

II. PROCEDURE FOR PAPER SUBMISSION

This section provides an outline of the related work. In particular, we discuss approaches for smart phones enriching android security by projecting solutions based on virtualization and secure container.

2.1 SECURE CONTAINER:

Creation of an isolated environment at an application layer level is done by the special mobile client application called as Secure Container. It provides authentication for an enterprise administrator to generate the policies which in turn helps to control the isolated environment. It does not help to control the actions of the user outside the container. This kind of technique doesn't need the modification of the image of the system and also widely analyses in the research community.

App security system popularly known as App Guard is a Java application that disassembles apk files. Inline security checks dangerous instructions as per the instructed policy and reassembles by signing packages. Henceforth, at runtime before compiling the dangerous code Appsecurity does a security check and as per the policy, if the instruction is not allowed then an exception is thrown. The functions in these applications are implemented in a standalone android services, which performs the additional checks. Many solutions use security container implemented as a user application in there solution.NitroDesk TouchDown and Good offer solution with prefixed set of business functionality in the container. Some more solutions, offer a set of basic applications and an SDK that will used as to develop new apps, if needed.

2.2 MOBILE VIRTUALIZATION:

Virtualization provides environments which are isolated from one another, and from the OS point of view,that are indistinguishable from the bare hardware. For isolation and co-coordinating the virtual machines activities, the hypervisor is responsible. Virtualization has been widely used in computers as it can:

(i) enhance security, and
(ii) decline the cost of applications deployment.

With the improvement in production of mobile devices and with the increment of their performance capabilities the main problem of porting virtualization to mobile platforms became actual. Virtualization for smartphones shows specific advantages like

(i)The probability to separate communication subsystems high-level application code

(ii)An prospect to afford license separation

(iii)A outlook to increase the security of communication stack.

However, there are still several barriers for the adoption of virtualization in mobile devices. The main one is that ARM architecture, which is the most popular architecture for mobile devices, has a non-virtualisable instruction set architecture (except Cortex-A15 design, which adds hardware-assisted virtualization capabilities). So as efficiency is a major concern in embedded virtualization, full virtualization approaches (emulation and binary translation) are not yet applicable for these devices because they are computational expensive.

III. PROPOSED SYSTEM

Our system helps to give an abstraction for isolating data and applications dedicated to many contexts that are to be installed in a single smartphone device. For particular, corporate data and applications can be isolated from personal data and applications within a single smart phone device. Our concept provides chamber where data and applications are stored. Our system provides enforcement mechanism that guarantees data and applications within a chamber are isolated from others chambers data and applications. These chambers are called Security Managers in our system. Generally, a SM is a set of policies that classify what applications can be executed and what data can be accessed. One of the important feature introduced in our system is the automatic activation of SM depending on context and also the location of the system, in which the smartphone is being used. SMs are joined with one or more interpreted of Context.

A context is defined as a Boolean expression determined over any instruction that can be gained from the smartphone's sensors (for e.g., GPS sensor). Logical sensors are functions which collaborate raw data from physical sensors such that to capture peculiar user performance. When a context solution performs true, SP correlated with such a context is triggered. It is an achievable situation when definite contexts, which are linked with various SMs, may be active at same time. To resolve such contests, each SM is also accredited such that allowing our system to activate the SM with the maximum priority. If SMs have the same priority, the SM, which activated first, will retain as active.

Our system allows a user to manually switch to a designed SM. To this end, our system presents a system applications that the user will be able to exploit for driving our system to trigger a given SM. Nonetheless, this conduct can be blocked to dodge that the user activates the useless or undesirable SM in a given context (for example, switching to a private SM when at home). Each SM is correlated with an owner of the profile and can be encrypted with a password. A SM can be generated/revise locally through an application installed on the mobile phone. Additionally, our system supports remote SM management. The latter

possibility may be used and by use of the mobile for monitoring his/her personal SM, while the latter may be engaged by an enterprise administrator to monitored the work SM. To prevent that the user tinkers with the work SM, the security administrator conserve the work SM with a password. In this manner, our system can be used for attaining a Mobile Device Management to manage remotely the security settings of a fleet of smart phone devices

IV. IMPLEMENTATION

One of the endowments of our system is that it can automatically shift SMs based on the current Context. Context Detector System is accounted for authorizing Context definitions and for advising the audience about the triggering or shutting off a Context. The Security Profile Manager peripheral, which is one of these audience, is triggered about the vary through
the bring back functions on two values such as True and False of the context_id, which in return correspond to triggering and shutting off a Context discretely. The context id specify sum to a Context id. Hence detection utility in our system context is detached from the remaining part of the system, it may be freely stretched by collaborating remaining solutions of context detection. When the system reboots, our system picks from db information about every Contexts and respective SM. Our system preserves this data in a compile time map in the form of <Ci;<SMk;prt>>, where Ci is the id of Context and (SPk;prt) is tuple, which is related to Context Ci and also contains of SM id. The priority prtk that corresponds to this profile. When the Context Detector System recognise that Context Ci becomes live i.e., the Context definition is changed to true, we pick from this map the respective tuple (SMk;prt) and mention it in the set of current SMs.As more than a Contexts would be active at the same time, there will be more than one SM to switch to. In this case, from the set of active SMs the system with the maximum priority is selected. If the listed SM id varies from id of the presently running SM, the Context Detector System projects a signal to Moses Hypervisor to shift to the newly created profile.

V. PERFOMANCE DISCUSSION

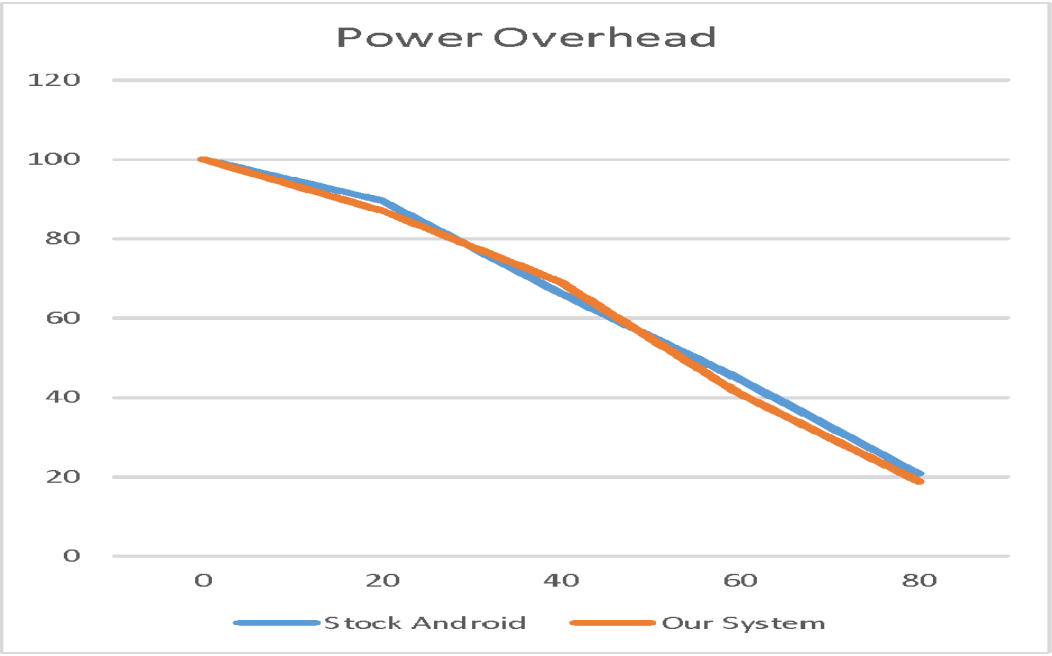
In this section, we address on the detailed experiments we ran to evaluate the performance of our system. For all the analysis, we used a Google Nexus S phone.

5.1 POWER OVERHEAD

To calculate the power overhead produced by our system, we performed the following tests. We charged the battery of our device to the 100 percent. Then, every 10 minutes we run five system applications (steadily) via a monkey runner script: Browser, Clock, Contacts, Calculator and Email applications. For each of the application, the script executed prevailing operations representative for the apps (for example, additions of numbers in Calculator application, browsing several web pages in Browser Application, changing time in clock application, calling a number in contacts application and creating an account in Contacts application, and composing and sending a email in Email app).
Each experiment remained for a sum of 180 minutes. We carried this experiment for two types of systems: Stock Android, and mobile with our system pre-installed in it and it supports SM switch changes (the system changes and switch the two profiles every 30 minutes).During each operation, every 1 minute, our system gauged the level of the battery and save this value into a log file. For each of the two studied systems, we executed the test 10 times and mean the resultant values. The outcome of this experiment is also reported. We remarked that the curves for the two designed systems behave alike. This proves that the fact that our system is just running, or even switching between the context does not earn a predictable power overhead.

5.2 REPOSITORY OVERHEAD

One of the most cogent overheads produced by our system is the repository overhead. Indeed, the isolation of data for



different SMs reminds that some application data will be replicated in some other different profiles.
In generic terms, the repository size utilized by a system can be determined by the following equation:
size = size (OS) + Application Executable files size (AEj) + App data files (ADj)

where AEj and ADj , are the application executable files and the application data files of the jth application, and OS is the operating system of the mobile system. In the specific case, in our system,
(ADj)in our system = $\Sigma\Sigma$ (ADij)where i in 1 to (n+1) and j in 1 to k
where size of (ADij), is the size of the data of the jth application in the ith SM, k is the number of installed applications, and n is the number of SMs. One additional copy of application data (i.e., the (n+1)th one) is required to store initial information of all applications. If a new SM is constructed, we need a clear copy of app data to be duplicated into this freshly created profile. Thence, our system stock a copy of app data just after the installing the application, this copy is used for later duplicates when a new SM is created. It should be specified that for our system only the antecedent data of apps are replicated.
The data generated by applications during compilation time are not duplicated between SMs. Secondly, the data of apps, which are not authorize in a profile, are not photocopied into the profile. When comparing our system with rivalry approaches, our system produces fewer repositories overhead. For particular, in case of mobile virtualization, not only app data are replicated (as for system), but also app executable files and an OS (sometimes fully or may be partially). Thus, our system sums less overhead analyzing to this set of ways as it only works with a copy of app executable files.

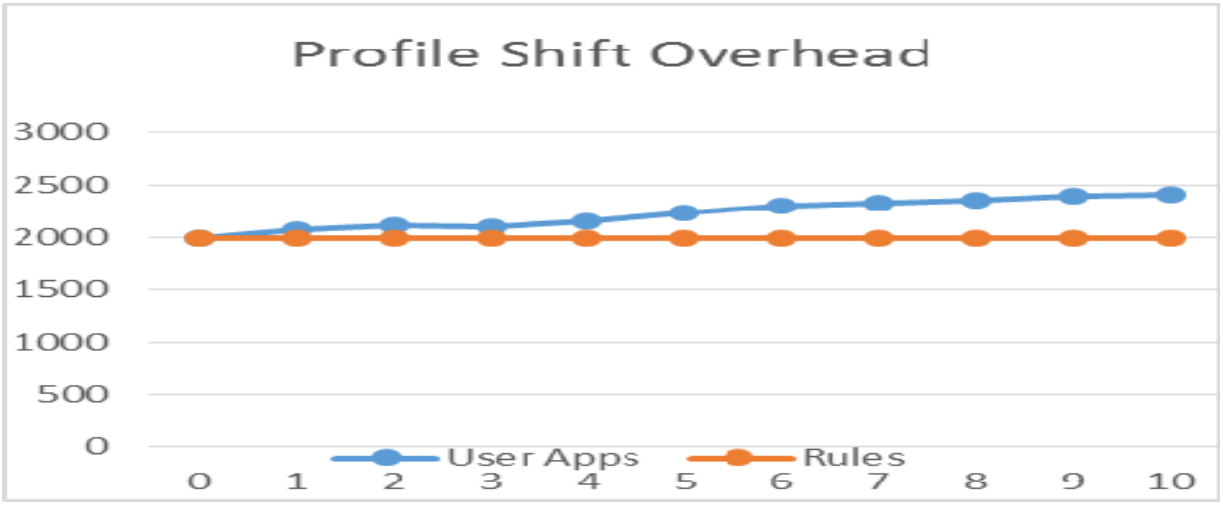
5.3 SWITCHING PROFILES OVERHEAD
In this part, we present the outcome of the experiments gauging the time needed to switch between SMs. We recall that in the course of the profile switch (from an old to a newly created profile), our system performs the following operations:

1. The unmounting of the document folders of the old security profile,
2. The mounting of the document folders of the new profile,
3. The discharge of the old and the charging of the newly specified Special Rules.

Henceforth, the time to shift between SMs should depend on the sum of the number of specified Special Rules and the number of user applications. To check out the dependency between the time and these parameters we ran a sets of experiments.
Firstly, we measured the time essential to transform SMs varying the total number of user apps. Then, we did the same measurement while changing the number of Special Rules. To calculate this time, we take a call function i.e., SystemClock.elapsedRealtime () by putting this function before and after the operations such as switching, and measured the difference between the values produced by this function. To delve into the dependency between the number of apps we changed the number of user apps from 0 to 10 and the time. For each number of apps, our system was used in a clean and clear manner (i.e., the system had been deployed on the mobile device just before the experiment). Then, a SM was created allowing all apps to be launched. Then, we gauged the time of shift between this newly created profile and DEFAULT SM. For every number of apps we recursively done the shift for 30 times and then measured the mean time of the shift. For entire attempts we made, have the same set of 10 apps was used.

From this figure, we noticed that the shifting time increments with the number of apps: moving from 1,000 ms for 0 applications to 3,000 ms for 30 apps. Time for profile shift as a function of the number of: (a) User applications, (b) special rules. 3,496 ms for 10 applications. The increment of the time is correlated with the enhancement of mounting and unmounting operations that our system behaves during the shift. Moreover, we marked that the time is not consistently rising with the increment of the number of apps.

Secondly, our examination was conducted equivalently to the first one, but here, in this case we discrete the number of specified Special Rules assigned to a new SM: from 0 to



50, incrementing by 10 rules every time. The consequence of this research are examined in above figure. At that time we can see, the time of the shift gradually raising with the enhancement of the number of rules assigned to the user apps. We can also mark that the function standard deviation for the examined values is noteworthy. We also observed that the time for the early change of profiles is appreciably mightier than for the following shifts. For particular, for four apps the time taken for the first switch is 2143 ms while for the second shift is just 1,431 ms. Indeed, during the first shift, for each app our system has to replicate the initial information of an app to a newly created profile.

VI. SECURITY MODEL

Our System consists of the several units and Base fundamental part of Our System is the approach of Context. The component CDS(Context Detector System) is authoritative for recognizing context activation/deactivation. When such an act happens, the Context Detector System sends a bulletin about this to the SPM(Security Profile Manager). The Security Profile Manager clutch the information associating a SM with one or more number of Context. The Security Profile Manager is authorized for the enlivening and demilitarize of SMs. The Security Profile Manager enforces the following logic:

If any new Context that is activated resembles to active SM then the proclamation is overlook; If the SM equivalent to a new Context that is active at present, has a lessened or balanced priority to the currently running SM, then the bulletin is ignored. The Our SAM (System App Manager) and the Our SRM(System Rules Manager). The erstwhile is responsible for authorizing which applications are allowed to be compiled within a SM. The closing one takes care of managing Special Rules.The Our SPM(System Policy Manager) enact as the policy administration point (PAP) in Our System. It covers the API for producing, renewing and evacuating all our System policies. It also permits a user to designate, customize, evacuate audited Contexts and accredit them to SMs. However, this part also curbs approach to Our System policy database also called as system.db file permitting only apps with exceptional acknowledgement to get around with this component. The imposition of separated SMs lack special factors to monitorize application processes and file system views. When a new SM is triggered, it might deny the compilation of some apps allowed in the last profile. If these apps are functioning during the profile shift, then we should stop their processes. Our System Reaper is the peripheral authorized for enclose processes of apps i.e., no longer allowed in the new SM after the shift. In our System, apps have authority to different data relying on the active profile. I It supports different file system view to isolate data between profiles. This functionality is considered by our System Mounter. To permit the user of the smartphone to interact with Our System, we provide two System applications: i) the System SM Changer ii)the System Policy Gui. The System SM Changer permits the user to manually trigger a SM. It interacts with the System Hypervisor and dispatch a signal to shift the profile enforced by the user. The System Policy Gui permits the user to monitor SMs.

VII. CONCLUSION

Security is vulnerable and valuable; resistance to security breaches is a valuable asset to the company. This perfect software based profile security enhances implementation of security in smart phone devices. Since it acts at system level we will be able prevent applications not to escape the security definitions. However sometimes android may assign same UID to some applications so in such applications we must restrict to same rules to be defined over them. We utilize the functionality of Taintdroid , virtualization by overcoming their limitations. A major drawback of the system is the separation of data for different SMs means that some application information will be duplicated in different profiles which induces storage overhead. We can further improve the performance of our system by developing system policy templates that can be simply selected and associated to the application.

REFERENCES

[1] Android Open Source Project (AOSP), http://source.android. com/, 2014.

[2] By the authors Y. Zhou, X. Zhang, X. Jiang, and V. Freeh, “Taming InformationStealing Smartphone Applications (on Android),” mentioned in Proc. Fourth Int’l Conf. Trust and Trustworthy Computing (TRUST ’11), pp. 93107

[3] “Understanding Android Security,”,by W. Enck, M. Ongtang, and P. McDaniel in IEEE Security and Privacy, vol. 7, no. 1, pp. 50-57, feb

[4] G. Russello, B. Crispo, E. Fernandes, and Y. Zhauniarovich, “YAASE: Yet Another Android Security Extension,” Proc. IEEE Third Int’l Conf. Social Computing and Privacy, Security, Risk and Trust (SocialCom/PASSAT), pp. 1033-1040, 2011.

[5] D. Kramer, A. Kocurova, S. Oussena, T. Clark, and P. Komisarczuk, “An Extensible, Self Contained, Layered Approach to Context Acquisition,” Proc. Third Int’l Workshop Middleware for Pervasive Mobile and Embedded Computing (M-MPAC ’11), pp. 6:1-6:7, 2011

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

[7] B. van Wissen, N. Palmer, R. Kemp, T. Kielmann, and H. Bal, “ContextDroid: An Expression-Based Context Framework for Android,” Proc. PhoneSense ’10, pp. 1-5, 2010.

[8] Divide Webpage, http://www.divide.com/, 2014

[9] OKL4 Microvisor, http://www.ok-labs.com/products/okl4-microvisor, 2014.

[10] S. Smalley and R. Craig, “Security Enhanced (SE) Android: Bringing flexible MAC to Android,” Proc. 20th Ann. Network and Distributed System Security

[11] NitroDesk TouchDown, http://www.nitrodesk.com/ TouchDown.aspx, 2014.