International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE)
ISSN: 0976-1353 Volume 24 Issue 7 – APRIL 2017 (SPECIAL ISSUE)

GS LAUNCHER

M Indumathy [#1], M Parveen Banu [*2] and K Kadambari [*3]

[#] *Asst. Professor, Dept. Of Information and Technology, Rajiv Gandhi College Of Engineering And Technology, Puducherry, India*

[*] *6th semester, Dept. Of Information and Technology, Rajiv Gandhi College Of Engineering And Technology, Puducherry, India*

***Abstract*— Smartphones are very popular among the users because of its multiple smart features related to education, entertainment, business etc. Companies design professional smartphones to complete most of the official works on travel like Windows phone with all MS Office feature. However, security concerns about data sharing, leakage and loss have hindered the adoption of smartphones for corporate use. In APP HIDING, 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. Profiles are not predefined or hardcoded, they can be specified and applied at any time. One of the main characteristics of APP HIDING is the dynamic switching from one security profile to another. We run a thorough set of experiments using our full implementation of APP HIDING. Smartphones allow end users to perform several tasks while being on the move. As a consequence, end users require their personal smartphones to be connected to their work IT infrastructure. More and more companies nowadays provide mobile versions of their desktop applications. Studies have shown that allowing access to enterprise services with smartphones increases employee productivity [2]. An increasing number of companies are even embracing the BYOD: Bring Your Own Device policy [3], leveraging the employee's smartphone to provide mobile access to company's applications. Several device manufacturers are even following this trend by producing smartphones able to handle two subscriber identification modules (SIMs) at the same time.**

***Index Terms*—About four key words or phrases in alphabetical order, separated by commas.**

I. OBJECTIVE

Objective of this project is to create a profile based application to the user interface for its security purpose and to create profile based application access for users.

II. PROBLEM DEFINITION

A. EXISTING SYSTEM

A solution could be implemented by means of virtualization technologies where different instances of an OS can run separately on the same device. Although virtualization is quite effective when deployed in full-fledged devices (PC and servers), it is still too resource demanding for embedded systems such as smartphones. Another approach that is less resource demanding is paravirtualization. Unlikely full virtualization where the guest OS is not aware of running in a virtualised environment, in paravirtualization it is necessary to modify the guest OS to boost performance. Paravirtualization for smartphones is currently under development and several solutions exist (e.g.,Trango, VirtualLogix, L4 microkernel, L4Android).

B. DISADVANTAGES:

- All the virtualization solutions suffer from having a coarse grained approach (i.e., the virtualised environments are completely separated, even when this might be a limitation for interaction).
- Other limitation is the hardcoding of the environment specification. Environments cannot be defined by the user/company according to their needs but they are predefined and hardcoded in the virtual machine.
- Furthermore, the switching among environments always requires user interactions and it could take a significant amount of time and power. While researchers are improving some of these aspects, the complete separation of virtual machines and the impossibility to change or adapt their specifications remain an open issue.

C. PROPOSED SYSTEM

In this paper, we propose APP HIDING provides an abstraction for separating data and apps dedicated to different contexts that are installed in a single device. For instance, corporate data and apps can be separated from personal data and apps within a single device. Our approach provides compartments where data and apps are stored. APP HIDING enforcement mechanism guarantees data and apps within a compartment are isolated from others compartments' data and apps. These compartments are called Security Profiles in APP HIDING. Generally speaking, a SP is a set of policies that regulates what applications can be executed and what data can be accessed.

D. ADVANTAGES

- One of the features introduced in APP HIDING is the automatic activation of SP depending on the context, in which the device is being used.
- APP HIDING can be used for realising a Mobile Device Management solution to manage remotely the security settings of a fleet of mobile devices

E. HARDWARE REQUIREMENTS:

- System : Pentium IV 2.4 GHz.
- Hard Disk : 40 GB.
- Floppy Drive : 1.44 Mb.
- Monitor : 15 VGA Colour.
- Mouse : Logitech.
- Ram : 512 Mb.
- MOBILE : ANDROID

F. SOFTWARE REQUIREMENTS:

- Operating system : Windows XP/7.
- Coding Language : Java 1.7
- Tool Kit : Android 2.3 ABOVE
- IDE : Eclipse

III. DESIGN MODULES

A. SYSTEM DESIGN

SYSTEM ARCHITECTURE:

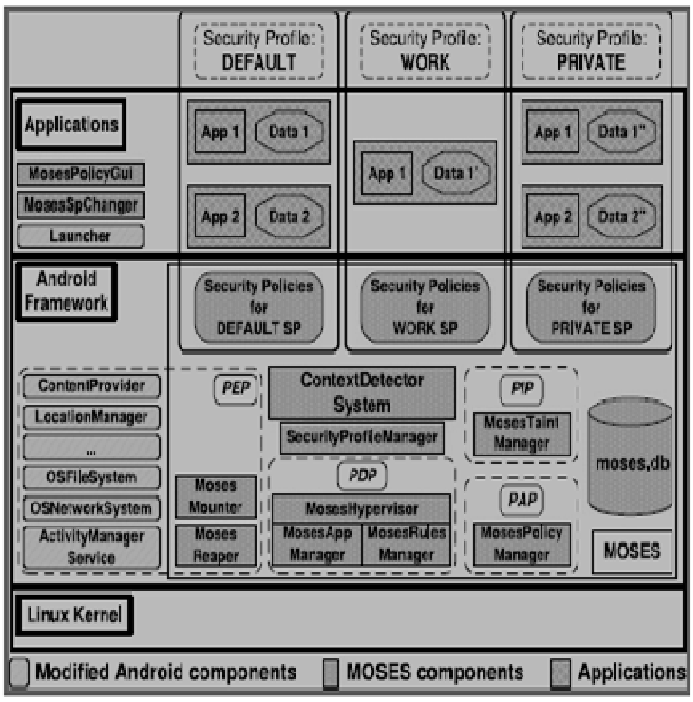


Fig. 1. MOSES architecture.

APP HIDING consists of the components presented in Fig. 1. Central to APP HIDING is the notion of Context. The component Context Detector System is responsible for detecting context activation/deactivation. When such an event happens, the Context Detector System sends a notification about this to the Security Profile Manager. The Security Profile Manager holds the information linking a SP with one or more Context. The Security Profile Manager is responsible for the activation and deactivation of SPs. The Security Profile Manager implements the following logic: _ If a newly activated Context corresponds to the active SP then the notification is ignored; _ If the SP corresponding to a newly active Context has a lower or equal priority to the currently running SP, then the notification is ignored; _ In all other cases, a SP switch has to be performed.

This means that the currently running SP has to be deactivated and the new SP becomes active. In the latter case, the Security Profile Manager sends a command to the APP HIDING Hypervisor informing which is the new SPs that needs to be activated. The APP HIDING Hypervisor is the component that acts as a policy decision point (PDP) in APP HIDING. The APP HIDING Hypervisor provides a central point for APP HIDING security checks against the policies defined for the active SP to regulate access to resources. The APP HIDING Hypervisor delegates the policy checks to its two managers: the APP HIDING App Manager and the APP HIDING Rules Manager. The former is responsible for deciding which apps are allowed to be executed within a SP. The latter takes care of managing Special Rules. The APP HIDING Policy Manager acts as the policy administrator point (PAP) in APP HIDING. It provides the API for creating, updating and deleting APP HIDING policies. It also allows a user to define, modify, remove monitored Contexts and assign them to SPs. Moreover, this component also controls access to APP HIDING policy database (APP HIDING.db) allowing only applications with special permissions to interact with this component. The APP HIDING Taint Manager component manages the “shadow database” which stores the taint values used by Taintdroid. We have extended the functionality of Taintdroid to perform more fine-grained tainting. In APP HIDING, we can taint specific rows of a content provider: to be able to perform per row filtering when an app access data in the content provider. For instance, it is possible to filter out from the query result data the rows which contain the information about device identifiers or user contacts. Given the fact that the enforcement of policies depends on the information provided by the APP HIDING Taint Manager, this component acts as a policy information point (PIP).

The decisions taken by the APP HIDING Hypervisor need to be enforced by the policy enforcement point (PEP). APP HIDING affects several components within Android middleware where decisions need to be enforced. For this reason, the PEP includes several Android components offering system services such as Location Manager and Activity Manager Service. Moreover, some Android core classes (such as the OS File System and OS Network System) are modified to enforce decisions regarding the access to the file system and network, respectively. The enforcement of separated SPs requires special components to manage application processes and file system views. When a new SP is activated, it might deny the execution of some applications allowed in the previous profile. If these applications are running during the profile switch, then we need to stop their processes.

The APP HIDING Reaper is the component responsible for shutting down processes of applications no longer allowed in the new SP after the switch. In APP HIDING, applications have access to different data depending on the active profile. To separate data between profiles different file system view are supported. This functionality is provided by the APP

HIDING Mounter. To allow the user of the device to interact with APP HIDING, we provide two APP HIDING applications: the APP HIDING Sp Changer and the APP HIDING Policy Gui. The APP HIDING Sp Changer allows the user to manually activate a SP. It communicates with the APP HIDING Hypervisor and sends it a signal to switch to the profile required by the user. The APP HIDING Policy Gui allows the user to manage SPs.

B. FUNCTIONAL REQUIREMENTS

1) NUMBER OF MODULES:

The system after careful analysis has been identified to be presented with the following modules:

1. App Building Block
2. Profile Creation
3. Application Control

2) MODULE DESCRIPTION

1. App Building Block

In this module we create an application vessel which can be installed in android platform devices. The application will be generated using Eclipse which would be installable in any android OS devices.

2. Profile Creation

Profiles like home, office, etc will be created in the form of switches. The different applications associated with the concerned profiles will be attached.

3. Application Control

With the concerned profile switch the priority will be provided with the applications. Whenever the profiles are called it will allow only the high priority applications to run.

IV. APP HIDING IMPLEMENTATION

This section describes implementation details of some keyaspects of APP HIDING. In particular, the version described hereis based on the Android Open Source Project (AOSP) [46] version 2.3.4_r1. Moreover, APP HIDING incorporates the functionality of Taintdroid [4] to taint sensitive data.

A. Context Detection

One of the contributions of APP HIDING is that it can automatically switch SPs based on the current Context. The Context- DetectorSystem is responsible for monitoring Context definitions and for notifying the listeners about the activation or deactivation of a Context. The SecurityProfileManager component, which is one of these listeners, is notified about the change through the callback functions onTrue (context_id) and onFalse(context_id), which correspond to activation and deactivation of a Context respectively. The context_id parameter represents a Context identifier. So as APP HIDING context detection functionality is decoupled from the rest of the system, it may be easily extended by integrating other context detection solutions [47], [48]. When the system starts up, APP HIDING selects from the database information about all Contexts and corresponding SPs. APP HIDING preserves this information in a runtime map in the form of hCi; ðSPk; prtkÞ I i, where Ci is the identifier of Context and ðSPk; prtkÞ i is a tuple, which corresponds to the Context Ci and consists of SP identifier SPk and the priority prtk that corresponds to this profile. When the ContextDetectorSystem detects that a Context Ci becomes active (meaning the Context definition is evaluated to true), we select from this map the corresponding tuple ðSPk; prtkÞ i and put it in the list of active SPs. Because more than one Contexts might be active at the same time, there may be more than one SP to switch to. In this case, from the list of active SPs the one with the highest priority is selected. If the selected SP identifier differs from the identifier of the currently running SP, the Context Detector System sends a signal to the APP HIDING Hypervisor to switch to the new profile. Similarly, when Context Detector System detects that a Context Ci becomes inactive, the tuple ðSPk; prtkÞ i is deleted from the list of active SPs. After that the selection procedure of a SP with the highest priority is repeated.

B. File system Virtualization

To separate data between different SPs, we use a technique called directory polyinstantiation [49]. A polyinstantiated directory is a directory that provides a different instances of itself according to some system parameters. In brief, for each SP APP HIDING creates a separate mount namespace [50]. The Android filesystem structure is quite stable, i.e., the system forces an application to store its files in the application's "home" directory that is /data/data/ <package_name>/ (<package_name> is the package name of the application). During the installation of an application, Android creates this "home" folder and assigns it Linux file permissions to allow only the owner of the directory (in this case the application) to access the data stored in it. To provide applications with different data depending on a currently running SPs, polyinstantiation of "data" folder may be used, i.e., for each SP a separate mount namespace, which points to different "physical" data folder depending on the identifier of a SP, may be created. In APP HIDING the described approach is used with two modifications. The first modification let the system to store all "physical" data directories under one parent directory (/data/APP HIDING_private/). The second modification creates the bindings not between the whole data folder and its "physical" counterpart, but bindings for separate application folders. The former modification allows APP HIDING to control direct access to the "physical" directories, while the latter permits to decrease storage overhead, because the usage of some apps is prohibited in some SPs. The APP HIDINGMounter component is responsible for providing the above functionality. In particular, it receives the list of applications' package names that are allowed to execute in a SP. For each package name, the APP HIDING system builds the paths to the application "home" directory and to its APP HIDING "physical" counterpart, using the information of the identifier of a newly activated SP. These two paths are passed to the APP HIDINGmounter native tool. This tool at first checks if APP HIDING "physical" directory exists. If not, then it creates this folder and copies there the initial application data

from the corresponding "home" directory. Then the APP HIDINGmounter mounts the "physical" directory to a "home" directory using the Linux command mount(target, mount_point, "none", MS_BIND, NULL) [50], where mount_point corresponds to the path of the "home" directory and target corresponds to the path of the "physical" folder. Thus, the "home" directory always contains the initial copy of the application data, which are created by the Android system during the application installation. If a new SP is created, these initial data are copied to the "physical" directory providing the application with a fresh copy of its initial data as if the application has just been installed. If this process finishes successfully, the APP HIDING Mounter stores the name of this package in the list of mounted points. Thus, the process of polyinstantiation is completely transparent for the applications: after the mounting the applications work with the same paths as usual, although these paths point to another "physical" locations. Thus, there is no need to modify the applications to support the separation of data between different SPs. Before switching to a new SP, the APP HIDINGMounter has to unmount all previously mounted points using the values stored in the list of mounted points. Similarly to the mounting, the APP HIDING Mounter passes the path to a mounted point (from the list of mounted points) to the APP HIDING mounter tool, which performs unmounting. During this operation it is possible that some processes hold some files opened. In this case, the unmount command will fail. To overcome this problem, APP HIDING sends a SIGTERM signal to the process and repeats the amounting. If after this the unmounting is still unsuccessful, APP HIDING will send a SIGKILL signal to the process and once again will perform the unmounts operation.

C. *Dynamic Application Activation*

Each SP is assigned with a list of application UIDs that are allowed to be run when this profile is active. As it was discussed in Section 2, each application during the installation receives its own UID. APP HIDING uses these identifiers to control which applications can be activated for each SP.
It should be mentioned that some packages can share the same UID. This happens if the developer of these applications have explicitly assigned the same value to shared User Id property in the manifest files of the applications, and signed these packages with the same certificate. Thus, during the installation of these applications, the Android system assigns them the same UID. In this case, APP HIDING cannot distinguish these applications and if one of them is allowed in one profile the other will be allowed as well. During the SP switching, the APP HIDING App Manager selects from the APP HIDING database the list of UIDs, which are allowed in the activated profile, and stores it into the set of allowed UIDs. To control the launch of applications' services and activities.

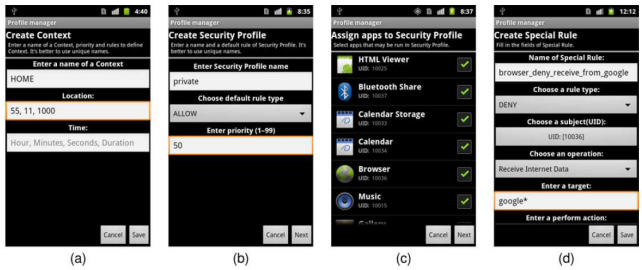


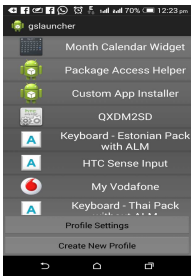
Fig. 2. Screenshots of APP HIDING Profile Manager application: (a) Context creation, (b) security profile creation, (c) application assignment to a security profile, (d) ABAC rule creation.

Retrieve Service Locked and start Activity May- Wait methods of the Activity Manager Service and the Activity Stack classes correspondingly are put. These hooks communicate with the APP HIDING App Manager and check against the set of allowed apps if a component of an application can be launched. Additionally, the APP HIDING App Manager controls the appearance of application icons in Android's Launcher application. When a new SP is activated, only the icons of the allowed applications for this profile will be displayed.

D. *SCREEN SHOTS*

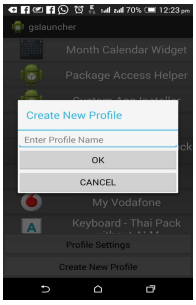
App Building Block

- In this module we create an application vessel which can be installed in android platform devices.
- The application will be generated using Eclipse which would be installable in any android OS devices.



Profile Creation

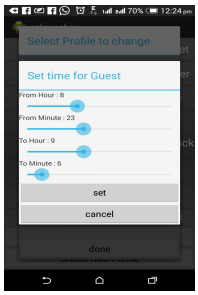
- Profiles like guest, home, office, etc will be created in the form of switches.
- The different applications associated with the concerned profiles will be attached.



Time Constrain

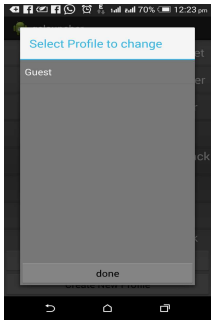
- For each profile setting a time will be their active period in the smartphone

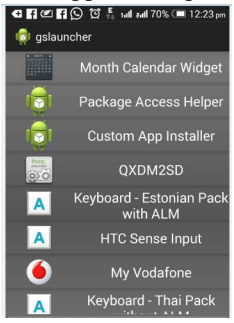
- Thus automatically change the profile according to the time constrain.
- Example: professionals in office daily routine time(9 AM TO 6PM)
- If someone don't want the time basis for the profile they can set 0 hour , minutes for from and to timing



Application Privacy

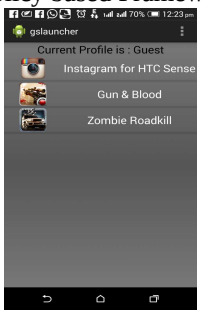
- After creating the profile with time constrain, add the application which are all want to be enable in that profile.
- By selecting that application will add to that profile.
- Other applications are not shown in that profile when it is active.
- Even though we can provide privacy on contacts, message, gallery , mail etc. which are all basic application want to be protected from others.
- Thus we can create a application privacy



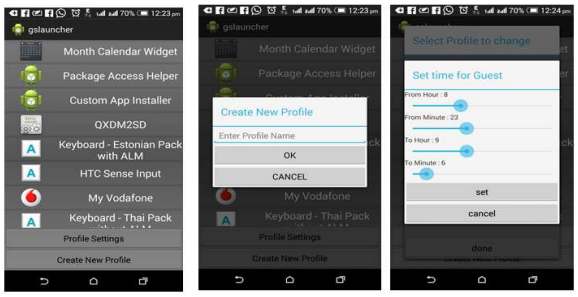


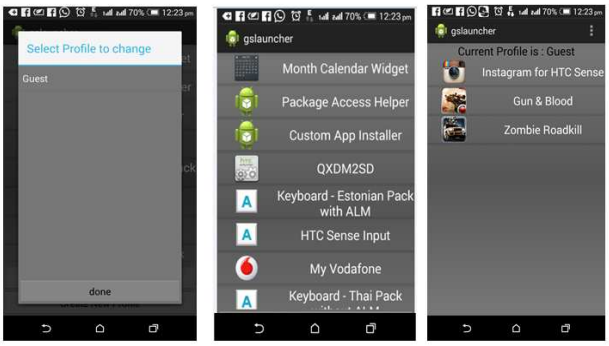
Secure And Privacy Profile

Thus finally secure and privacy for application in smartphone is provided by multiple profiles Creation using policy based Framework-APP HIDING.



Steps





CODING

```
public void openDataBase() throws SQLException{

    //Open the database
    String myPath = DB_PATH + DB_NAME;
    myDataBase = SQLiteDatabase.openDatabase(myPath, null,SQLiteDatabase.OPEN_READONLY);

}

@Override
public synchronized void close() {

    if(myDataBase != null)
        myDataBase.close();

    super.close();

}

@Override
public void onCreate(SQLiteDatabase db) {

}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

}

// Add your public helper methods to access and get content from the database.
// You could return cursors by doing "return myDataBase.query(....)" so it'd be easy
```

```
        // to you to create adapters for your views.

    }
```

V. CONCLUSION

APP HIDING is the first solution to provide policy-based security containers implemented completely via software. By acting at the system level we prevent applications to be able to bypass our isolation. However, at the present moment APP HIDING has also some limitations. At first, fine-grained policies and allowed applications are specified using the UID of an application. Meanwhile, in Android it is possible that some applications share the same UID. Thus, if we apply APP HIDING rules and restrictions to one application they automatically will be extended to the other ones with same UID. Furthermore, some fine-grained policies in APP HIDING are built on top of Taintdroid [4] functionality. Thus, APP HIDING inherits the limitations of Taintdroid explained in Section 3. It should be also mentioned that the applications that have root access to the system can bypass APP HIDING protection. Thus, APP HIDING is ineffective in combating with the malware that obtains root access, e.g., rootkits. APP HIDING can also be improved in several aspects. For instance, to make the policy specification process easier, a solution could be to embed into the system policy templates that can be simply selected and associated to an application. It should be also mentioned that currently APP HIDING does not separate system data (e.g., system configuration files) and information on SD cards. In the future we plan to add this functionality to the system. Moreover, performance overheads are also planned to be reduced considerably in the future versions.

REFERENCES

[1] S. Chen, B. Mulgrew, and P. M. Grant, “A clustering technique for digital communications channel equalization using radial basis function networks,” *IEEE Trans. on Neural Networks*, vol. 4, pp. 570-578, July 1993.

[2] J. U. Duncombe, “Infrared navigation—Part I: An assessment of feasibility,” *IEEE Trans. Electron Devices*, vol. ED-11, pp. 34-39, Jan. 1959.

[3] C. Y. Lin, M. Wu, J. A. Bloom, I. J. Cox, and M. Miller, “Rotation, scale, and translation resilient public watermarking for images,” *IEEE Trans. Image Process.*, vol. 10, no. 5, pp. 767-782, May 2001.