SE-Persona: Secure-Enhanced Containers on Mobile Platform

Mohamed El-Serngawy
École de Technologie Supérieure
Dept of Software Engineering
Montreal, QC, Canada
Email: Mohamed.elserngawy.1@ens.etsmtl.ca

Chamseddine Talhi
École de Technologie Supérieure
Dept of Software Engineering
Montreal, QC, Canada
Email: Talhi.Chamseddine@etsmtl.ca

***Abstract*—Shortly smart devices especially tablets will be the primary computing device in the business world. The smart device helps the employees to access client's information easily and quickly. Moreover, individuals use these smart devices as a common device for business and personal purposes. The overlap between personal and business usage attracts the companies attention to secure their data that correspondingly restrict the devices usages to business needs. However, employees are uncomfortable with losing control of their private data through having restrictions on their smart device. Mobile virtualization is a solution for/to the overlapping problem in which the enterprise secures its data without controlling the employees personal data. However, the mobile virtualization inherently shares the underlying system hardware resources without enforcing policy on their interfaces that may result in side channel attacks. For instances, a side channel attack could easily track the employee location through his personal profile although there is a restriction on the location service of the business profile. Specifically, there is a need for mutual exclusion policies for these hardware interfaces and sensors. In this paper, we investigate the security risks of the mobile virtualization technology. Focusing on the operating system level virtualization on Android platform and propose a new middle-ware policy model to protect the user privacy.**

I. INTRODUCTION

The range of using smart devices, specifically phones and tablets, is getting more predominant in the individuals life. Many companies and organizations are trying to adapt Bring Your Own Device BYOD model [47] with their enterprise networks and systems to help their employees get the maximum benefits of the smart device's usage. The employees use their own smart devices to connect to their company's enterprise systems and do their work with greater flexibility and freedom [36]. Other companies have security concerns, so they provide their employees with the company smart devices and prevent the employees from using their own smart devices. In both cases, companies need a policy management system to ensure security and overcome employees' misuse [40]. Mobile virtualization technology is one of the solutions that achieves data isolation and policy management in the both models. In fact, mobile virtualization offers flexibility with the smart devices usage and addresses the concerns over the data security and privacy. Therefore, the employee can have her/his personal mobile platform VM (Virtual Machine) side-by-side with the business mobile platform VM in the same device. The separation of the mobile platform VMs (personal and business) let the employees have full control over their private data without affecting the enterprise system requirements of controlling their data security and policies management. Different virtualization technologies such as KVM [14], Xen [18] and LXC [15] are being adapted to mobile platforms considering multiple factors such as the hardware specification, the availability of software applications and the back-end management systems. However, the limitation of the smart device's hardware compared with the desktop or server computers challenges the virtualization technologies to adapt to the mobile platforms. Fortunately, the lightweight OS-level virtualization technologies such as LXC [15] or Cells [5] are well adapted to the Android platform due to the minimum hardware requirements and the kernel sharing mechanism between containers that increases the system performance. In the OS-level virtualization, the virtual machine named as container due to the kernel sharing mechanism that runs different operating systems of the same type simultaneously. Also in mobile virtualization technology, the container could be named as Persona due to the different personal usage of the containers. The OS-level virtualization provides a virtual environment (for the Linux kernel like operating systems) that has its process and network communication, instead of creating a full-stack operating system as a virtual machine. Furthermore, the kernel sharing mechanism incurs less CPU and memory usage and network overhead that is important due to the performance and hardware limitations in the smart device. However, the virtualization technologies were established and used on desktop and server platforms, did not address the different behavior and context of the smart device. Due to the mobility and context changes in the smart devices, the virtualization technologies on mobile platforms could lead the user data (personal or business) to privacy escalation without running a malicious execution. For example ˝consider a mobile accountant agent who has a smart device running her/his business and personal mobile platform containers. The business container has a mobile enterprise application to record transactions and the GPS-coordinate as the accountant meets with customers and distributes money. Typically, any third party application in the personal container (which is running in the background) has access to the GPS will be able to record the GPS-coordinates while the accountant is doing her work. However, when the accountant agent is running the mobile enterprise application in her business container, the GPS-coordinate becomes sensitive enterprise data. Meanwhile, the enterprise policy management system could not disable the GPS in the personal accountant container because of the employee's needs, and the company's policy management system has no control over it˵. Mobile virtualization inherently shares

the underlying system hardware resources without enforcing policy on specific security sensitive hardware interfaces such as the sensors that may easily expose in the side channels attacks. Specifically, there is a need for mutual exclusion policies for these hardware interfaces.
Also, the privilege escalation [16] [7] is a known attack on the Android platform which can be applied using a different mechanism. Many research projects have been conducted to defeat the privilege and privacy escalation on the Android platform such as FlaskDroid [8], CRePE [12] and SE-Android [56]. The SE-Android becomes fully integrated into the Android Open Source Project AOSP [22] since Android Jellybean (v4.3). SE-Android provides two levels of operating system policy inside the Android software stack. 1) Secure Enhancement Linux [44] policy which allows a low-level policy integration with the kernel subsystems and modules. 2) Middleware Mandatory Access Control MMAC policy [31] which allows a middle-level policy integration with the Android framework and third party applications. SE-Android policy integration enhances the Android platform security. However, SE-Android addresses the privilege and privacy escalation in a single Android platform and needs more investigation to be adapted to the Android virtualizations technology. In this paper, we implement se-policy module for the virtualized environment control host (management components), and we propose a new MMAC policy built on the top of the SE-Android to overcome the privacy escalation in Android virtualization technology.
We summarize the contribution of our work as follows::

- Exploring the business usage and security requirements of the mobile virtualization technology, especially with the OS-level virtualization.
- Proposing a new MMAC policy to prevent the privacy exposure on the Android virtualization technology.
- implement SE-Policy module to secure the virtualized environment management component (control-host).
- Design and implement a prototype built on the top of Cells open source project with minimum changes to the Cells source code.

The paper's roadmap organized as follows. Section 2 has the related work. In section 3 we give background information on the virtualization technology, the business use cases and the implementation of OS-level virtualization on Android platform. In section 4 we explore the security requirements for the mobile virtualization technologies. In section 5 we mentioned the adversary model that we apply to the mobile virtualization technology. In section 6 we explain the new MMAC policy properties we proposed Section 7 contain the technical details of SE-Persona prototype design and implementation. Section 8 We evaluate the performance and security of SE-Persona. Finally in section 9 the conclusion and future works.

II. Related Work

The Android platform uses the sandbox mechanism [25] to isolate applications and processes. Also, Android platform has a set of *uses-permission* [24] to allows applications gain access to system resources. Furthermore, any application can declare and enforce permissions to share its modules with other applications [27]. It was shown that the Android platform security architecture has weaknesses at [6], [16]. Many research projects such as TrustDroid [63], CRePE [12] and FlaskDroid [8] extended the Android platform to overcome the Android security weaknesses. In contrast to our approach, the user-level isolation mechanisms that introduced by these contributions requires massive changes to the Android platform components and introduce more complexity to the overall system. L4Android [41] used the hypervisor (micro-kernel) virtualization to provides isolation between different user spaces. The main disadvantage of the hypervisor virtualization is the complexity of adapting the software with new Linux kernel versions or porting it to new hardware. Furthermore, the hypervisor virtualization shows a higher overhead performance when it compared to OS-level virtualization. Authors of [57] used the OS-level virtualization to provides secure and isolated environments. They adapted the LXC [15] to the Android platform to virtualize different user-spaces environments. Also, they relied on the built-in Linux kernel security models such as mandatory access control (MAC) and access control lists (ACLs) to secure the system components. Both approaches (L4Android and LXC) adapted to provide data isolation and prevent privilege escalation between containers/VMs. However, they did not provide context awareness policy or prevent the privacy escalation between containers as we explained in the introduction section.

III. Background in Android Virtualization

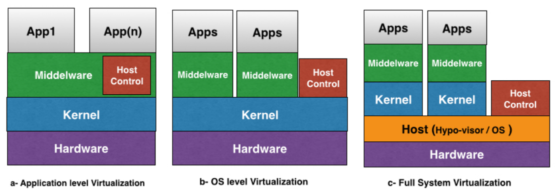
In the 1960s, the first virtualized computer was created by IBM [13]. At that time, the main purpose of virtualization technology was to share the system resources between different running applications/processes. Over the years, virtualization technology has evolved. Today we have server virtualization, desktop virtualization, mobile virtualization, and more recently, embedded system virtualization. As virtualization technology developed through these different stages, it required changes in both hardware and software computer system architecture.

A. Virtualization Technology Classification

Virtualization technologies can be divided into two major perspectives.
a) Hardware perspective: The Hardware perspective classifies virtualization technology into three types: 1) Full-Virtualization refers to a mechanism by which the virtual machine operating system does not require any changes in its components as it is unaware of the host virtualized environment. 2) Para-Virtualization refers to a mechanism by which the virtual machine operating system requires changes to its interface hardware components to be able to communicate with the hardware through the hypervisor interface. And 3) Hardware Support Virtualization refers to a mechanism by which the CPU supports the host environment with different modes of operations to run the processes with a different level of privileges. The host OS uses the root mode to run high privilege processes and use other modes to run the less privileged processes.
b) Operating system perspective: The OS perspective classifies virtualization technology into three categories; 1) Application level virtualization refers to a mechanism by which the

virtualized components exist in the middleware layer, and the control component is part of the middleware layer such as, the Dalvik virtual machine on Android platform [17]. 2) Operating System level virtualization refers to a mechanism by which the containers share kernel layer. However the control component is not part of the container middleware framework, such as Linux containers [15]. 3) Full System Virtualization refers to a mechanism by which the virtual machine OS is running without sharing any part of its software stack as Xen Hypo-Visor [18] and KVM [14]. Figure 1 shows the three types of virtualization technology from the operating system perspective.

Fig. 1: Different Types of Virtualization Technology



B. Business usages of Android Virtualization

Data isolation and secure communication are the main purposes of Android virtualization technology. As security is the main requirement of enterprise systems, we identify the most common usage of Android virtualization with the enterprise systems as follows.

a) Service and Application isolation [28]: In the smart device, enterprise applications and services are placed in a sandbox environment where the back-end enterprise system can wipe or maintain damage of the enterprise application's data in case the sandbox environment is compromised. The sandbox environment can provide secure communication with the enterprise system. However, it does not provide data isolation. The sandbox environment manages the application data after it becomes compromised to restrict the data damages, but does not prevent the data damage. Another limitation in the sandbox environment is that the enterprise system does not have the ability to apply a restricted policy on the smart device. This defect can lead to privacy and privilege escalation with the enterprise application's data.

b) Two isolated containers [50]: two rigid separated environments (business and personal) run simultaneously on the same physical smart device to cover the enterprise system security requirements. In the smart device, the enterprise applications and services run in the isolated business container environment with the ability to be remotely managed by the enterprise system. Moreover, the enterprise system can apply a restricted policy on the smart device during the working hours and locations and has full control over the smart device. The main disadvantage of the two-container environments is that the user loses full control over her/his private data as the enterprise system can manage the personal container environment. Furthermore, from the users' perspective, their private data may be compromised by the enterprise system restricting the users' personal usage of the smart device.

c) Multi-OS experiences [45]: the user can run multiple Android platform simultaneously on the same smart device depending on the used virtualization technology. The OS-level virtualization restricts the system to share a common kernel between the Android containers. However, it can let the user run many different OS with a constraint of same kernel compatibility. Users can create their business Android container and give the enterprise system full control over it. The enterprise system can manage the enterprise applications in the business container and apply a restricted policy over the smart device while the business container is in the foreground and actively used by the user. In the meantime, the user has full control over her/his personal container without affecting the enterprise system security requirements. Moreover, since there are many new mobile operating systems on the rise based on Linux kernels such as Android, Ubuntu Touch OS [45], Firefox OS [4] and Tizen [3], this may be useful for the enterprise system to use its secure platform on the smart device and also useful to many experienced users.

C. OS-level Virtualization implementation

Cells: is an OS-level virtualization that shares a single kernel across all containers with virtualized identifiers, kernel interfaces, and hardware resources. Figure 1-b typically shows the cells' architecture. Cells uses different namespace's isolation mechanisms [39] applied on the kernel devices with hardware resource multiplexing to provide process isolation with native performance. In more technical details, the purpose of namespace isolation is to provide an abstraction of a global system resource that makes it appear to the processes within the namespace that have their isolated instance of the global resource. Cells uses the Mount, Net and UTS namespace isolation mechanisms. These namespace's isolation mechanisms required a wrapper device interface for the kernel drivers, as well as modification to the kernel device subsystems to manage the namespace ID. The cells architecture showed that the root namespace (control-host) is part of the Trusted Computing Base (TCB), and it is not accessible by the containers. The Cells system starts by booting up the root namespace, then initializes the Celld process, and the minimum required components. When the system creates a new container, the Celld process mounts the file system then clones itself with a new namespace ID and starts the init process to boot up the new container. Also, Celld process is responsible for managing containers (create, start, stop, switch and destroy). The Cells user will be able to manage the system by connecting the device to Desktop PC and use the ADB (Android Debug Bridge) command to gain access to the root namespace shell and then use the cell command to manage the system.

LXC: LXC (Linux Containers) is an OS-level virtualization environment for running multiple isolated Linux-base systems (containers) on a single Linux control-host. Condriod [10] and Android Container [2] both are OS-level virtualization adapt the LXC with the Android platform. Condroid uses the namespace isolation and cgroups (control groups) [43] kernel features to limit, account and isolate the system resources usage. The cgroup provides a partitioning mechanism for a set of tasks and set all future children into hierarchical groups in which each group of tasks is isolated. The Condroid architecture shows the control host as a complete Android system. Besides, All other containers will run on the top of control host. Condroid considers that the host Android system is a part of the trusted computing base (TCB), and the device's user can not download or install third party applications on it. Condroid adapt the LXC tools to manage the system functionalities such as create, start, stop and destroy containers. The condroid's user will be able to manage the system through the complete host Android system. Also, condroid provides a service sharing interface and virtualized binder component to enhance the system performance.

Security Weaknesses: Both of Cells and Condroid provide data isolation as the containers are running in different namespaces. However, Cells and Condroid did not provide context awareness policy for the running containers. Also, consider the control host as part of the TCB architecture in both systems without evaluating the security risks and possible threats make the systems under high potential security risk. For example in Cells, the user must connect the device to a Desktop PC and use the Android Debug Bridge (ADB) to communicate and manage the virtualization environment. Also, condroid assume that the user will not misuse the base Android host environment.

IV. Adversary Models

The virtualization technologies provide data isolation between containers/VMs by nature as each container/VM runs as a complete software stack. On the server platform, security solutions such as firewall and anti-virus protect the server platform from possible threats and side channel attacks. However, the smart device usage is different. The user can easily install/uninstall third party applications and connect to different networks that could lead to misuse her/his private data.

Side channel attacks: In our adversary model we consider the side channel attacks from the running background containers is a series threat. Researchers have been shown a side channel attacks using camera at [59], microphone at [51], Frame Buffer at [42] and motion sensors at [9]. All the defense mechanisms to defeat these side-channel attacks based on a single platform. For instance, the AudioFlinger service will prevent audio data from leaking to untrusted applications during a phone call. However, with virtualizing the AudioFlinger service, the foreground container/VM will block the audio data for the untrusted applications in the same container as it is unaware of the running background containers. The same attack concept could be applied on the Framebuffer to access the screen pixels or to LocationManager to access the GPS coordinate data. The attack vector defines as follows: a process P belongs to a running background container C´, will be able to access data from a resource system R that is under restricted access policy in the foreground container C˝. Where C´and C´´belong to Cs is a set of running containers on the smart device.

V. Security Requirements in Mobile Virtualization Technology

The security requirements for the enterprise systems are variant depend on the business cases and scenarios. Enterprise mobile applications usages and security requirements were showed in [48]. In the following, we will list the security requirements of the enterprise system regards to mobile virtualization technology.

Isolation between containers/VMs and virtualized environment control-host: Initially the separation of the software stack of the running containers/VMs either sharing the kernel between container as OS-level virtualization or running a complete OS as the hypovisor virtualization both provide data isolation. Furthermore, file system encryption capability offers more security to the stored data. However, the virtualized environment control-host has a possible security risk to be compromised from one of the running containers/VMs, especially in the OS-level virtualization. In fact, exposing the virtualized environment control-host could lead to privilege escalation attacks between containers/VMs.

Policy enforcement and Context Awareness: The running foreground container/VM should have the ability to apply a context awareness policy on its running applications and processes. Meanwhile, as the foreground container/VM is not aware of other background running containers/VMs, it should have the ability to apply a high level restricted policy on the smart device resources such as camera or microphone based on time and location.

Secure Remote Management: The users needs to manage their virtualized environment. There are two possible techniques to manage the virtualized environment depend on the virtualized platform architecture. 1) Local management where the virtualized environment management module exists in one of the running containers/VMs. The host control container works as a master container, and it is part of the trusted computing based TCB architecture. 2) Remote management where the virtualized environment control-host can be accessible by an authenticated network connection or through a connected desktop PC (USB/Serial connection). The virtualized environment management module is part of the trusted computing based architecture, and it runs early in the system's components boot up sequence. Both techniques should be implemented in mobile virtualization technologies due to usability and security requirements.

VI. MMAC Policy for Android Virtualization

In the Android platform, applications should be digitally signed by an X.509 certificate to be installed on the system. The signing certificate allows the Android platform to provide signature-based permissions enforcement between applications that use the same certificate. Applications with the same signing certificate can share applications' modules, data, and files. The SE-Android MMAC policy is built on top of Android platform's middleware framework security architecture. The MMAC policy controls the application's access (grant/deny uses-permission) to the system resources even after the permission was granted to the application during the installation process. The MMAC policy categorizes the installed applications depending on their signing certificates. In Android open source project AOSP, the system applications are placed under the AOSP signed categories and the third party applications are placed under the default category. The Android platform vendors can insert a new applications category to the MMAC policy by using the Insert-Key tool [49] and they can grant/deny the uses-permissions [23] to third party applications. The MMAC policy source file is mac_permission.xml that is written in XML format with predefined tags [21]. Table I describes the available XML tags in the MMAC policy.

Context Awareness: The applications categorization in the MMAC policy provides flexibility to manage the system *uses-permissions* with different applications instead of defining a

TABLE I: MMAC Policy Tags

MMAC Policy Tags and Description
<signer signature="" >
Signer tag required a signature with a hex encoded X.509 certificate, Ex:<signer signature="PLATFORM" >The platform is tag referred by Keys.conf file
<seinfo value="" >
Seinfo tag represents additional info that each app can use in setting a SE-Android security context on the eventual process, Ex: <seinfo value="platform">
<package name="" >
Package tag defines allow and deny android system permission for a certain package name protected by the signature Ex: <package name="com.source.test">
<allow-permission name="" >
Allow-permission tag define the allowed Android system permission for a certain applications category or application package Ex: <allow-permission name ="android.permission.INTERNET" >
<deny-permission name="" >
Deny-permission tag defines the denied Android system permission for a certain applications category or application package Ex: <deny-permission name ="android.permission.WAKE_LOCK" >
<default >
Default tag is used to define the default policy that will be applied on the third party applications.
<allow-all >
Allow-all tag is used to allows any Android system permission requested by the applications under certain category.

policy for each application. However, it is missing the context awarenesses. The security requirements for different stakeholders depend on the smart device's usage (times, locations and sensitive process execution). As we explained in the introduction section, the security policy of an enterprise system might dictate that certain assets in the smart device such as the microphone should be restricted during the working hours or while the device is connected to the enterprise network. According to the current MMAC policy properties, the policy Access Control AC decision will be taken as black/white list concept. The smart device user will not be able to change the application state from the whitelist (allow uses-permission) to the blacklist (deny uses-permission) until she/he reload the MMAC Policy (mac-permission.xml). Moreover, in the virtualized environment, the running foreground container is unaware of the running background containers' MMAC policies that could easily lead to data privacy escalation between containers.

A. SE-Persona MMAC policy

To enable the AC (access control) context awarenesses decision inside the MMAC policy on Android platform, we propose new properties to the MMAC policy *Exclusive-use*, *Exclusive-deny*, *Trusted-app*, *Security-level*, *Restricted-Permission* and *Restricted-Applications* their definitions as follows:

1) *Exclusive-use*: The application has declared the required uses-permission to gain access to a system resource such as GPS. Meanwhile, at the application's execution time, the application should be the only process that has access to the required system resource.
2) *Exclusive-deny*: At the application's execution time, the application might need to restrict the accessibility of a certain system resource such as microphone due to security concern. However, the application is not required access to the system resource to work properly.
3) *Trusted-app*: The MMAC policy should specify which application or applications' category will have the ability to apply the *Exclusive-use* and *Exclusive-deny* policy. Usually the trusted applications that have an X.509 certificate signed by the Android OS owner such as Google, SAMSUNG, Telcom provider and the Enterprise system. Otherwise, any third party application can easily interrupt the system resources.
4) *Security-level*: As the MMAC policy could have multiple categories or applications defined under the *Trusted-app* property, the *Security-level* prioritize the trusted application. Otherwise, the MMAC policy rules might cause conflicts in the AC decision.
5) *Restricted-Permission*: According to the different business usage of the enterprise systems, the enterprise system should have the ability to restrict the access to a certain system resource during certain times or locations. the *Restricted-Permission* will be applied on all running containers (background and foreground) on the smart device and it has the highest security level.
6) *Restricted-Applications*: In Android platform AOSP the Play Store runs as a system application. To restrict the employees from use the Play store application and install third party applications in the business container. The Play store application package name can be defined under the *Restricted-Applications* tag to restrict the employee from using the play store.

Since usability and security are concerns to the smart device's user and enterprise systems, the *Exclusive-use* and *Exclusive-deny* properties should be defined by the enterprise mobile application in its Manifest file and the *Trusted-app* property should be declared under the enterprise application's category in the MMAC policy (mac_ permission.xml). As the MMAC policy and the Manifest file are written in XML format, we defined the new policy's proprieties as XML tags. Tables II shows the new MMAC policy's properties tags and description.

B. The MMAC policy Rules Conflicts

In Android virtualization environments, the running foreground container is unaware of other running containers' applications and processes. The policy AC decision conflict will happen in two cases. A) When two applications running in the same container are declared in the MMAC policy under two different categories and both of them required *Exclusive-use-permission* for the same system resource. In this case, the policy AC decision should be taken by the smart device's user to choose which application should gain access to the system resource or by prioritizing the applications' categories using the security-levels property. B) When two applications are declared in the MMAC policy under two different categories with same security-level and they are running in two different containers. Both of them required *Exclusive-use-permission* for the same system resource. In this case, the system will grant access to the application that runs in the foreground container. We solve the policy AC decision conflict by taking the AC decision in two steps; 1) Inside the container as each container responsible for its policy AC decision; and 2) Inside the root namespace (control-host) as the root namespace is the

TABLE II: The new MMAC Policy Properties tags definition

The Properties' Tags and Description
<Exclusive-use-permission android:name="" >
We used the same Android uses-permission names to specify which system resource is required for the exclusive use. e.g:<Exclusive-use-permission android:name="android.permission.ACCESS_ FINE_ LOCATION" />
<Exclusive-deny-permission android:name="" >
We used the same Android uses-permission names to specify which system resource is required for the exclusive deny. e.g:<Exclusive-deny-permission android:name="android.permission.RECORD_ AUDIO" >
<Trusted-app value="" >
The Trusted-app properties value could be 1 or 0 (1 = trusted, 0 = untrusted) and by default it is 0. Also it should be defined after the signer or package tag. e.g: <signer signature="" > <Trusted-app value="1" ></signer>
<Security-level value="" >
The Security-level value depends on the policy creator prioritization and it should be defined under the Trusted-app tag. e.g: <Trusted-app value="1" ><Security-level value="1" >. We initially set 3 security-levels
<Restricted-Permission x="" y="" len="" start-time="" end-time=""> </Restricted-Permission>
The Restricted-Permission could be defined under the signer tag to be applied on certain applications' category or could be defined without a signer tag to be applied on the smart device. The restricted permission should be listed within the Restricted-permission tag. e.g: <Restricted-Permission x="79.223" y="23.88" len="1000" start-time="9:00:00" end-time="14:00:00"><uses-permission android:name="android.permission.ACCESS_ FINE_ LOCATION"/ ></Restricted-Permission>
<Restricted-Application >
The Restricted-Application will be define to restrict the uses of some system applications such as the Play store application in some containers. The application's package name should be defined under the Restricted-Application tag. e.g: <Restricted-Application><package name="com.android.vending"/ ></Restricted-Application>

only module that aware of all running containers' applications and processes. The policy AC decision inside the container formulated as follows.

- $AC\ (A, R, A \in Ta, Ta \in Sl) \mapsto 1/0$

AC is the access control decision function will grant the application (A) which is the first input parameter to access the system resource (R) which is the second input parameter if: A ∈ Ta where Ta is a set of the *Trusted-app* in the MMAC policy. Ta ∈ Sl where Sl is a set of the highest *Security-level* categories in the container's MMAC policy.
The policy AC decision inside the root namespace formulated as following:

- $AC\ (A, R, A \in Ta, A \in Fc, Ta \in Sl) \mapsto 1/0$

AC is the access control decision function will grant the application (A) which is the first input parameter to access the system resource (R) which is the second input parameter if: A ∈ Ta where Ta is a set of the *Trusted-app* in the MMAC policy. A ∈ Fc where Fc is a set of the running applications in the foreground container. Ta ∈ Sl where Sl is a set of the highest *Security-level* categories in the foreground container's MMAC policy.

VII. Design and Implementation

We implement the SE-Persona prototype based on the Cells open source project [37] that is built on top of the Android 4.3 JB version. In Android 4.3 JB version, there are about 50-70 system services that give the mobile applications capabilities to communicate with the system resources through callable interfaces [60]. The Android platform provides Binder IPC mechanism [52] to allow communication between applications and system services. There are 150 built-in permissions on Android platform to constrain the application accesses to the system resources. We classify the system resources into two types; 1) The simulated resources by which the resource could simulate its state and data. For example, the location manager service that represents the geographic coordinates will give fake coordinate (x=0,y=0). And 2) The unsimulated resources by which the resource can not simulate its operation or data such as the bluetooth and network. The idea behind this classification is to prevent the application's segmentation fault or to crash while applying the MMAC policy enforcement. The applications that require access to a simulated system resource will receive a simulated data, and those applications that require access to unsimulated system resources will receive the access denied signal if they were actively running before the MMAC policy enforcement decision. The default mac_permissions.xml file which contain the MMAC policy definition exist in the Android open source project AOSP under external/sepolicy/mac_permissions.xml in the source tree and as /system/etc/security/mac_permissions.xml on the device. In Cells, the read-only file system is shared between containers. A file system union mechanism [58] is used to provide a union view to the container consist of its read-write file system and the read-only file system. We exclude the mac_permission.xml from the shared read-only file system and include it in the read-write file system to let each container has its MMAC policy. Initially, any new container will have the default MMAC policy as it exist in the AOSP then the container will be able to reload its MMAC policy.The default Android platform permissions check mechanism is managed by the package manager PM [54] and the requested service. For example, when an application sends a binder request to query data from the location manager service, the location manager service will consult the package manager through the system_ server [1] to verify if the application had granted the required permission that is the ACCESS_ FINE_ LOCATION permission in this example. The location manager will send the data if the required permission has been granted otherwise will send security exception signal to the application. We modified the PackageManager module in the Android framework to meet the new requirements of the MMAC policy, and we implement two new modules that are the Permission Controller Manager PCM and Root Permission Controller Manager RootPCM. In the following, we explain our modification to Android framework components.
PackageParser: [46] The package parser module verify and extract the information of the mobile application executable file (app.apk file) at the beginning of the installation process. As we added new tags to the application Manifest file (*Exclusive-use* and *Exclusive-deny* permission), we modified the package parser to be able to extract this information.
PackageManager: [54] The package manager service is responsible for installing and uninstalling applications on the Android platform. Also, it manages the installed applications meta information such as the package name and its related UID, the installation directories, the required permissions and the application certificate signature. All these meta information is stored in the packages.xml file under /data/system/ directory. We modified the package manager to include the *Exclusive-use* and *Exclusive-deny* permission tags that exist in the application's Manifest file and during the installation process

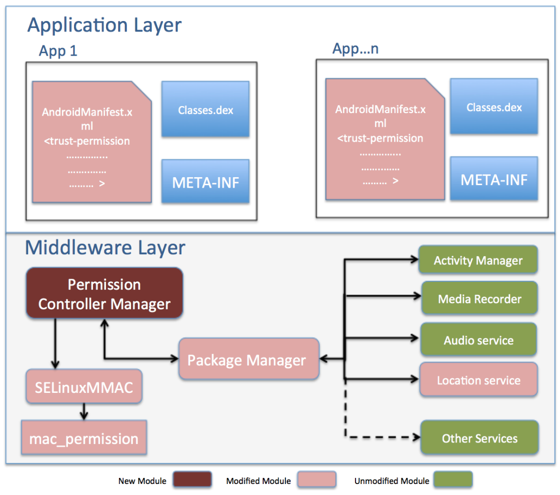
checks for the application's related category in the MMAC policy.

SELinuxMMAC: [30] The SELinuxMMAC module is responsible for reading, parsing and validating the MMAC policy. We extend the SELinuxMMAC to consider the new MMAC policy properties we have proposed and store the new policy tags with its relative application's package information.

PermissionControllerManager PCM: PCM is the heart of SE-Persona implementation, PCM will start early at the container initialization. PCM will load the MMAC policy and initialize a list of the Exclusive-use/deny permissions that required by the MMAC policy. Each permission in the permission list has a status active/inactive. When the Activity Manager [53] starts an application, the PCM will check the application's category, related permissions (Exclusive-use/deny), uses-permission and Security-Level. When untrusted application starts and its uses-permission match one of the Exclusive-use/deny permissions that required by the MMAC policy, the PCM will add the application to the tracked applications list. When a trusted application starts, the PCM will add the application to the active application list then check its corresponding Exclusive-use/deny permissions and update the related permissions status to the active status. The PCM will send an intent to the corresponding system service (such as location-manager) to simulate its data and for the unsimulated system service (such as the microphone) the PCM will send an intent to the package-manager to deny its uses-permission if requested during the trusted application execution time. In our prototype, we only make a modification to the location manager service to simulate its data. Also, the PCM has a global setting for the Geolocation and day time to apply the high level restricted policy such as *Restricted-Permission* and *Restricted-Application*. Figure 2 shows the Architecture of the added components in Android security model.

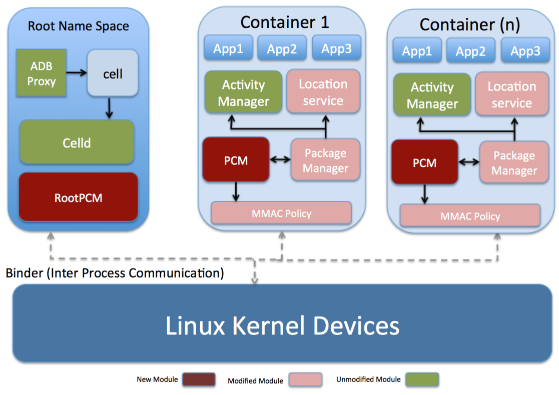
RootPermissionControllerManager RootPCM: The Root-

Fig. 2: Architecture of the Security Model Inside Single SE-Persona



PCM runs in the root namespace on Cells platform. When a new container starts, the RootPCM will register its PCM component at the running container list to notify the PCM about other containers' policy requirements. When an Exclusive-use/deny permission has an active status inside a container, the RootPCM will notify the others containers PCM component through a binder interface [52] to add the Exclusive-use/deny permission to its permission list (if not exist) and change the permission status to active. The RootPCM and PCMs will communicate according to the Exclusive-use/deny permission status change. Figure 3 shows the new security architecture we propose for the Android platform and Cells. **ADB Proxy for**

Fig. 3: Architecture of SE-Persona security model

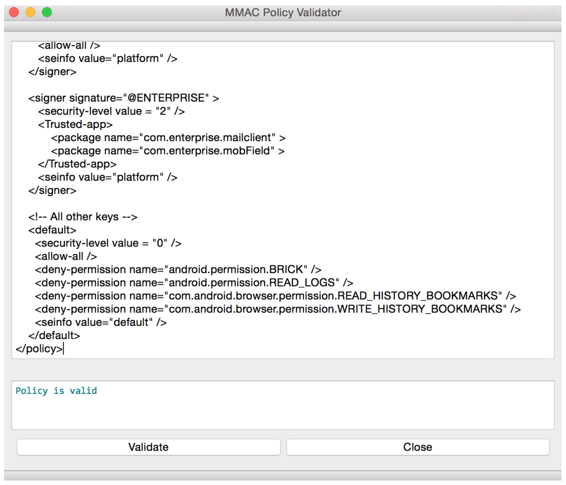


Mobile virtualization technology: In this section, we propose an ADB proxy component to virtualize the Android Debug Bridge (ADB) component on Android platform and restrict the ADB access to the root namespace. The ADB in both sides Desktop PC and the Android smart device acts as a transparent transport mechanism [61]. Its two most important components are 1) the adb server that is running on the host (Desktop PC) and 2) the adbd daemon that is running on the target (Android smart device). These two components effectively implement a proxy protocol on which both adb services are implemented, and they linked together either through USB or TCP/IP connection [**?**]. We modified the adbd daemon (the target device) to virtualize and restrict the adb executed commands from the adb server (the host device). When the ADB connection is established between the smart device and a desktop PC, adbd daemon service will request from celld daemon service the current running foreground container's name and developer options accessibility. The smart device's user will be able to use all the adb commands and functionality [26] with the running foreground container as a normal Android platform. All other running background containers are not accessible by the adb commands as the ADB is not aware of them.

A. *Remote Management*

For simplicity, in our prototype we implement a desktop application based on the ADB proxy component to let the smart device user able to manage the containers (create/start/stop/delete) and update the MMAC policy of each container. When the user is trying to add a new application category to the MMAC policy, she/he should provide the X.509 certificate that will be used for the new application category. In the real scenario, the new application category certificate should be signed by the Enterprise system and the mobile enterprise application as well. The remote management application will validate the new MMAC policy before send it to the target container. Figure 4 show the remote management application.

Fig. 4: MMAC Policy Validator



```
      <allow-all />
      <seinfo value="platform" />
    </signer>

    <signer signature="@ENTERPRISE" >
      <security-level value = "2" />
      <Trusted-app>
         <package name="com.enterprise.mailclient" >
         <package name="com.enterprise.mobField" >
      </Trusted-app>
      <seinfo value="platform" />
    </signer>

    <!-- All other keys -->
    <default>
     <security-level value = "0" />
     <allow-all />
     <deny-permission name="android.permission.BRICK" />
     <deny-permission name="android.permission.READ_LOGS" />
     <deny-permission name="com.android.browser.permission.READ_HISTORY_BOOKMARKS" />
     <deny-permission name="com.android.browser.permission.WRITE_HISTORY_BOOKMARKS" />
     <seinfo value="default" />
    </default>
</policy>
```

```
Policy is valid
```

B. SE-Policy implementation

As we implement our prototype using Cells open source project. We implement the celld.te module within the SE-Policy modules [55] to protect the celld process (see background section III-C) from privilege escalation attacks. The SE-Policy contains the definitions of the security classes and their associated permissions and rules that are used by the Linux Security Modules (LSM) to apply the MAC on Android kernel. The procedures to extend and customize the SE-Policy modules are well explained at [29]. The celld.te module contain the definition of the celld process as security class and its associated type, rules and permissions. We define the celld process type as domain type, and we assigned to it the allow rules to control the cgroups and mount operation. Also, we add the celld process as a security class in class security module [33] and define the label configurations of celld and cell binaries in the file_context.te module [32]. The Celld process has the privilege to mount the system partitions, starts the init process and control cgroups in the kernel systems. Figure5 shows the celld.te module source file.

Fig. 5: Celld SE-Policy Module

```
# celld
type celld, domain;
type celld_exec, exec_type, file_type;

init_daemon_domain(celld)
typeattribute celld mlstrustedsubject;
# Override DAC on files and switch uid/gid.
allow celld self:capability { dac_override setgid setuid fowner };
# Drop capabilities from bounding set.
allow celld self:capability setpcap;
# Move children into the peer process group.
allow celld system:process { getpgid setpgid };
allow celld appdomain:process { getpgid setpgid };
# Write to system data.
allow celld system_data_file:dir rw_dir_perms;
allow celld system_data_file:file create_file_perms;
# Control cgroups.
allow celld cgroup:dir create_dir_perms;
allow celld self:capability sys_admin;
# Check validity of SELinux context before use.
selinux_check_context(celld)
# Check SELinux permissions.
selinux_check_access(celld)

# Setting up storage.
allow celld rootfs:dir mounton;
allow celld sdcard_type:dir { write search setattr create add_name mounton };
dontaudit celld self:capability fsetid;
allow celld tmpfs:dir { write create add_name setattr mounton search };
allow celld tmpfs:filesystem mount;
allow celld labeledfs:filesystem remount;

# Handle --invoke-with command.
allow celld celld_exec:file { execute_no_trans open };
allow celld ashmem_device:chr_file execute;
allow celld init:binder call;
allow celld shell_data_file:file { write getattr };
allow celld system:binder { transfer call };
allow celld servicemanager:binder { call };
```

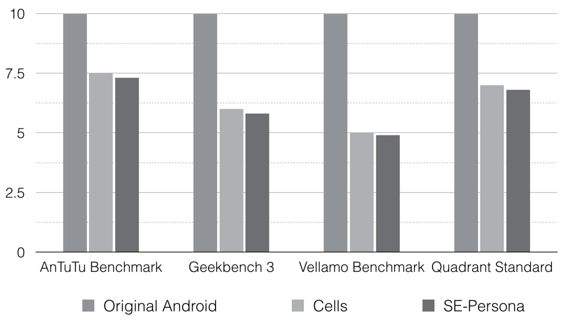
VIII. Evaluation and Performance

In this section, we present measurements regards to the SE-Persona platform performance and then we have a discussion about the system security.

A. Performance

We evaluate the performance of SE-Persona prototype by comparing the original Cells platform performance with SE-Persona prototype performance using four benchmarks AnTuTu Benchmark, Geekbench 3, Vellamo Mobile Benchmark and Quadrant Standard Edition. These benchmark applications have been used to evaluate the original Cells implementation and also used by different researchers to evaluate the Android performance [62]. The benchmarks environment consist of three running containers, and the used device is Nexus 7 version 2012 (1.3GHz quad-core Tegra 3 processor, 1GB of RAM, 12-core GPU, 16GB flash memory). These benchmarks are designed to test the different aspect of the device functionality and resources such as 2D and 3D graphics performance, Disk I/O, Memory I/O and CPU performance. The scores explain how well the device is running after the modification that SE-Persona added to Cells. We run these benchmarks five times in each container, and we normalize the results values to the manufacturer's unmodified Android OS, which has the benchmarks score value 10. Figure 6 shows the result of the benchmarks that ran in both SE-Persona prototype and original Cells.

Fig. 6: Benchmarks score result of three running containers



B. Evaluation

To evaluate the new MMAC policy we propose, we will discuss some of the current Android platforms that provide isolated environment for enterprise mobile application execution.

Android-for-Work: Many Android devices such as Black-Phone [11] provide a secondary isolated environment to run security-sensitive or privileged operations that do not need the functionality of a multipurpose OS. The isolated environment is referred to as a Secure OS. The Secure OS is implemented on the top of Trusted Platform Module (TPM) using another processor in the hardware layer or can be isolated beneath the kernel on a shared processor using secure execution environment such as TrustZone in ARM processors. Android for work [35] provides the Secure OS with a strong encryption functionality to encrypt the enterprise mobile application data and with a configurable security policy. The configurable security policy prevents some functionalities such as screen capture from being executed during the Secure OS execution. However, it is missing other functionalities such as the motion sensor that could be used in a side channel attack as we mentioned in the adversary models section IV. In fact, Android for work require the mobile enterprise application to implement the Device Administrator module using the Android SDK [34] to be able to use the Android for work capabilities that overwhelm the enterprise side. Moreover, the Secure OS not based on a virtualization that limit the end user for only two separated environments.

Samsung Knox: SAMSUNG has develop Knox [50] as a complete enterprise mobile solution. Knox developed based on the integration between hardware layer using TrustZone (TZ) execution and with software layer using SE-Android and Knox containers. It focuses on providing capabilities such as Trusted Boot, encrypted file-system, Digital Rights Management (DRM) and isolated mobile applications store. As for policy enforcement, Knox integrates SEAndroid and provides management APIs to customize security policies. Despite that Knox APIs are integrated into Android 5.0 (Android for work based on Knox), its adoption is limited to SAMSUNG devices only. Moreover, it requires TrustZone hardware support, which limits its deployment to only ARM based Android platforms.

Secure space: Secure Spaces is similar to Cells as an OS-level virtualization and management solution for Android smartphones and tablets [19]. It enables multiple Persona (virtual Android OS) to co-exist on a single Android device. It focuses on the Persona type (personal/work/games) to apply the security policy and management. While Secure Space provides strong data isolation and hidden private Persona [20], it missing the context awareness policy as Cells.

SE-Persona security: SE-Persona provides a flexibility and scalability to manage the container security policy without updating the Android middleware framework. By integrating the new MMAC policy in Cells, the third party applications are not able to access the system resources such as Audio-Flinger service during the *Exclusive-use* or *Exclusive-deny* permission is applied in one of the running containers. Moreover, the *Security-level* in MMAC policy applied more fine-grained access control on the application's execution. For example, SE-Persona can set up the mobile banking applications under a higher security level rather than normal third party applications and prevent access to accelerometer sensor while the banking application is running. Also the *Restricted-Permission* added more usability and scalability to the Android platform permission, the user able to grant/deny a certain permission depend on her/his location and times without required the application to define its *Exclusive-use* or *Exclusive-deny* permission. The *Restricted-Application* added more scalability to the containers. For example, in Android platform AOSP the Gmail, Play Store, and YouTube applications run as trusted system application when the smart device's user create a new container for her/his kids for child activity. The kids could misuse the Play Store by installing undesirable applications or use the Youtube to watch undesirable videos. By using the *Restricted-Application* the smart device user can prevent access to these applications in her/his kids container platform.

Privilege escalation: The root user permission is required for the Android virtualized environment. The root user in each container is running under different namespace ID, which prevent the super user application from getting a privilege to other containers. The control-host (as celld process for SE-Persona) is the attacker target to control the virtualized environment. Based on the root user privilege [38] we were able to execute the cell command from one of the running containers that let us create, start, stop and destroy other containers. We explained the attack success by the fact that the celld process and cell command were not defined nor implemented in the SE-Policy modules on the original Cells platform. We implement the celld.te module (see sectionVII-B) for SE-Persona to prevent the privilege escalation to the control-host from the running containers.

Limitation: SE-Persona implementation goal was making a minimum modification to the original Cells and Android platform. That was satisfied by using the on/off technique to control the system permission depend on the current context. However, this technique limits the context awareness policy to be extendable nor self-defined. For example, SE-Persona can not define the device context depend on the application execution behavior. SE-Persona can not recognize the device context depend on application type (e.g., background service app, chat app, media player app). Also, SE-Persona classifies the applications depend on its signed certificate that mismatch with other application classification such as the applications' categories in Google Play store. Overall SE-Persona context awareness policy can satisfy the enterprise systems security and the user privacy. However, the Android middleware framework components might require massive changes to let the MMAC policy self-adapted not relying on the end user.

IX. Conclusion

In this paper, we explored the business usage and security requirements of the mobile virtualization technology. We presented SE-Persona a secure enhancement container for the Android platform. We focused our work on the context awareness policy of Android platform and securing the virtualized environment control-host. In contrast to existing solutions based on context awareness policies and full system virtualization, our prototype implementation has a minimum modifications to the original Cells and Android platform. Our evaluation to SE-Persona showed that SE-Persona can satisfy the enterprise system usage. However, mobile virtualization technology need changes to fit with the internet of things (IOT) era we started.

References

[1] "Android zygote startup section Sequence of system_server startup steps," http://elinux.org/Android_Zygote_Startup, August 2010.

[2] "Android container," https://code.google.com/p/seek-for-android/wiki/AndroidContainer, Feb 2012.

[3] "Tizan operating system," https://www.tizen.org/, 2012.

[4] "Firefox os," https://www.mozilla.org/en-US/firefox/os/, 2013.

[5] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh, "Cells: a virtual mobile smartphone architecture," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 173–187.

[6] 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*. ACM, 2010, pp. 73–84.

[7] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry, "Towards taming privilege-escalation attacks on android." in *NDSS*, 2012.

[8] S. Bugiel, S. Heuser, and A.-R. Sadeghi, "Towards a framework for android security modules: Extending se android type enforcement to android middleware," Tech. Rep. TUD-CS-2012-0231, Center for Advanced Security Research Darmstadt, Tech. Rep., 2012.

[9] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion." in *HotSec*, 2011.

[10] W. Chen, L. Xu, G. Li, and Y. Xiang, "A lightweight virtualization solution for android devices," *IEEE Transactions on Computers*, 2015.

[11] S. C. company, "Black phone 2," https://www.silentcircle.com/products-and-solutions/devices/.

[12] M. Conti, V. T. N. Nguyen, and B. Crispo, "Crepe: Context-related policy enforcement for android," in *Information Security*. Springer, 2011, pp. 331–345.

[13] R. J. Creasy, "The origin of the vm370 time-sharing system," *IBM Journal of Research and Development*, vol. 25, no. 5, pp. 483–490, 1981.

[14] C. Dall and J. Nieh, "Kvm/arm: the design and implementation of the linux arm hypervisor," in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*. ACM, 2014, pp. 333–348.

[15] S. G. Daniel Lezcano, Serge Hallyn, "Linux containers," https://linuxcontainers.org/, Aug 2008.

[16] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, "Privilege escalation attacks on android," in *Information Security*. Springer, 2011, pp. 346–360.

[17] D. Ehringer, "The dalvik virtual machine architecture," *Techn. report (March 2010)*, 2010.

[18] L. Foundation, "Xen project hypervisor for arm architecture," http://www.xenproject.org/developers/teams/arm-hypervisor.html, 2013.

[19] graphite software, "Secure space android rom," http://www.securespaces.com/WP2015/.

[20] ——, "Secure space hidden persona," https://www.youtube.com/watch?v=G6p2clFHdyc, Oct 2015.

[21] R. Haines, "Install-time mmac policy file," in *The SELinux Notebook-4th Edition*, vol. 4, 2014.

[22] A. Inc, "Android open source project," https://source.android.com/, October 2003.

[23] ——, "Android platform permissions," http://developer.android.com/reference/android/Manifest.permission.html, 2007.

[24] ——, "Android platform uses-permission," http://developer.android.com/guide/topics/manifest/uses-permission-element.html, 2007.

[25] ——, "Android security the application sandbox," https://source.android.com/devices/tech/security/#the-application-sandbox, 2008.

[26] ——, "Adb commands," http://developer.android.com/tools/help/adb.html#commandsummary, 2009.

[27] ——, "Declaring and enforcing permissions," http://developer.android.com/guide/topics/security/permissions.html#declaring, 2010.

[28] ——, "Android platform device administration," http://developer.android.com/guide/topics/admin/device-admin.html, Oct 2011.

[29] ——, "Implementing selinux," https://source.android.com/devices/tech/security/selinux/implement.html, 2012.

[30] ——, "Se-linux mmac source code," https://android.googlesource.com/platform/frameworks/base/+/b267554/services/java/com/android/server/pm/SELinuxMMAC.java, 2012.

[31] ——, "Middle-ware mandatory access control in android platform," http://seandroid.bitbucket.org/MiddlewareMAC.html#middleware-mac, Jun 2013.

[32] ——, "File_context," https://android.googlesource.com/platform/external/sepolicy/$+$/android-4.4.4\textunderscorer2.0.1/file\textunderscorecontexts, Aug 2014.

[33] ——, "Security_class," https://android.googlesource.com/platform/external/sepolicy/$+$/android-4.4.4\textunderscorer2.0.1/security\textunderscoreclasses, Aug 2014.

[34] G. Inc, "Device administration," http://developer.android.com/guide/topics/admin/device-admin.html, 2013.

[35] ——, "Android for work security white paper," https://static.googleusercontent.com/media/www.google.ca/en/CA/work/android/files/android-for-work-security-white-paper.pdf, May 2015.

[36] D. Jaramillo, M. Ackerbauer, and S. Woodburn, "A user study on mobile virtualization to measure personal freedom vs. enterprise security," *SOUTHEASTCON 2014, IEEE*, pp. 1–5, 2014.

[37] N. K. Jeremy C. Andrus, "Cells: Lightweight virtual smartphones," https://cells-source.cs.columbia.edu/#/q/status:open,n,z, Oct 2011.

[38] J. C. Jongma, "How-to su," http://su.chainfire.eu/, Oct 2012.

[39] M. Kerrisk, "Namespaces in operation, part 1: namespaces overview," http://lwn.net/Articles/531114/, Jan 2013.

[40] E. B. Koh, J. Oh, and C. Im, "A study on security threats and dynamic access control technology for byod, smart-work environment," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 2, 2014.

[41] M. Lange and S. Liebergeld, "L4android: Android on top of l4," 2011.

[42] C.-C. Lin, H. Li, X. Zhou, and X. Wang, "Screenmilker: How to milk your android screen for secrets," in *NDSS Symposium 2014*, 2014.

[43] P. Menage, "Cgroups," https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt, 2014.

[44] N. S. A. NSA, "Secure enhancement linux," https://www.nsa.gov/research/selinux/, 2009.

[45] J. Palandri, "Dual boot installation android and ubuntu touch," https://wiki.ubuntu.com/Touch/DualBootInstallation, 2014.

[46] A. Platform, "Package parser," https://android.googlesource.com/platform/frameworks/base/core/java/android/content/pm/PackageParser.java, 2007.

[47] D. Raths, "Are you ready for byod?." *The Journal*, vol. 39, no. 4, pp. 28–32, 2012.

[48] K. Rhee, W. Jeon, and D. Won, "Security requirements of a mobile device management system," *International Journal of Security and Its Applications*, vol. 6, no. 2, pp. 353–358, 2012.

[49] W. Roberts, "Mmac policy: Insert key tool," https://android.googlesource.com/platform/external/sepolicy/+/android-4.4.4_r2.0.1/tools/insertkeys.py, Dec 2012.

[50] SAMSUNG, "Knox an enterprise mobile platform," http://www.samsung.com/global/business/mobile/platform/mobile-platform/knox/, April 2013.

[51] R. Schlegel, K. Zhang, X.-y. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound trojan for smartphones." in *NDSS*, vol. 11, 2011, pp. 17–33.

[52] T. Schreiber, "Android binder," *A shorter, more general work, but good for an overview of Binder. http://www. nds. rub. de/media/attachments/files/2012/03/binder.pdf.*

[53] A. SDK, "Android activity manager," http://developer.android.com/reference/android/app/ActivityManager.html, Aug 2014.

[54] ——, "Android package manager," http://developer.android.com/reference/android/content/pm/PackageManager.html, 2014.

[55] S. Smalley and R. Craig, "Se-policy," https://android.googlesource.com/platform/external/sepolicy/, 2013.

[56] ——, "Security enhanced (se) android: Bringing flexible mac to android." in *NDSS*, 2013.

[57] S. Wessel, F. Stumpf, I. Herdt, and C. Eckert, "Improving mobile device security with operating system-level virtualization," in *Security and Privacy Protection in Information Processing Systems*. Springer, 2013, pp. 148–161.

[58] C. P. Wright, J. Dave, P. Gupta, H. Krishnan, D. P. Quigley, E. Zadok, and M. N. Zubair, "Versatility and unix semantics in namespace unification," *ACM Transactions on Storage (TOS)*, vol. 2, no. 1, pp. 74–105, 2006.

[59] N. Xu, F. Zhang, Y. Luo, W. Jia, D. Xuan, and J. Teng, "Stealthy video capturer: a new video-based spyware in 3g smartphones," in *Proceedings of the second ACM conference on Wireless network security*. ACM, 2009, pp. 69–78.

[60] K. Yaghmour, *Embedded Android: Porting, Extending, and Customizing*. " O'Reilly Media, Inc.", 2013.

[61] ——, *Embedded Android: Porting, Extending, and Customizing*. " O'Reilly Media, Inc.", 2013.

[62] H.-J. Yoon, "A study on the performance of android platform," *International Journal on Computer Science and Engineering*, vol. 4, no. 4, p. 532, 2012.

[63] Z. Zhao and F. C. C. Osono, "trustdroid: Preventing the use of smartphones for information leaking in corporate networks through the used of static analysis taint tracking," in *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*. IEEE, 2012, pp. 135–143.