ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THESIS PRESENTED TO
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
MASTER IN APPLIED SCIENCE
M.Sc.A.

BY
Mohamed EL-SERNGAWY

SECURING ENTERPRISE SYSTEMS DATA ON SMART DEVICES

MONTREAL, JUNE 17, 2015

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS:

Mr. Chamseddine Talhi, thesis director
Software Engineering and IT , École de Technologie Supérieure

Mr. Chakib Tadj, committee president
Dep. Génie électrique, École de Technologie Supérieure

Mr. Abdelouahed Gherbi, committee member
Dep. de Génie logiciel et des technologies de l'information, École de Technologie Supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON "DEFENSE DATE"

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

I am thankful to my wife and family for all the support, patience and love that they have provided to me. My special gratitude to Prof. Chamseddine Talhi for his guidance and encouragement throughout the course of this endeavor.

SECURING ENTERPRISE SYSTEMS DATA ON SMART DEVICES

Mohamed EL-SERNGAWY

ABSTRACT

In near future smart devices such as phones and tablets will be the main computing device in the business world. The mobility behavior of the smart device helps the business workers to easily access their client's information and make the right decision at the required time. However, people use smart devices for personal and business usage, which affect their companies and organization's enterprise systems. For instant, the information that smart device is able to produce could be valuable business data at certain time and could be valuable private data at another times such as camera photos. This overlap between the personal and business usage in smart devices leads the companies to pay attention to their enterprise systems data security and apply restrictions on the smart devices usage. However, the smart device users are not comfortable with losing control of their private data or having restrictions on their smart device. In this work, we will study the usage and security of the smart device with the enterprise system. We will classify the Enterprise mobile applications usage. We will study the possible threats to expose the smart device user's data and applying a new screenshot attack as an example of these threats. Finally we will study the mobile virtualization technology and investigate its security.

Keywords: Smartphone, Android, security architecture, threat vectors, risk analysis

SECURING ENTERPRISE SYSTEMS DATA ON SMART DEVICES

Mohamed EL-SERNGAWY

ABSTRACT

Dans un avenir proche dispositifs intelligents tels que les téléphones et les tablettes sera le dispositif principal de l'informatique dans le monde des affaires. Le comportement de la mobilité de l'appareil intelligent aide les travailleurs d'entreprises d'accéder facilement à l'information de leurs clients et de prendre la bonne décision au moment voulu. Cependant, les gens utilisent des appareils intelligents pour l'utilisation personnelle et d'affaires, qui affectent leurs entreprises et organisations des systèmes de l'entreprise. Pour instant, les informations que dispositif intelligent est capable de produire pourrait être précieuses données commerciales à certain moment et pourrait être utile données privées à un autre temps, comme des photos de l'appareil photo. Ce chevauchement entre l'utilisation personnelle et professionnelle dans des dispositifs intelligents conduit les entreprises à faire attention à leur sécurité des données des systèmes d'entreprise et d'appliquer des restrictions à l'utilisation intelligente des dispositifs. Toutefois, les utilisateurs d'appareils intelligents ne sont pas à l'aise avec de perdre le contrôle de leurs données privées ou ayant des restrictions sur leur appareil intelligent. Dans ce travail, nous allons étudier l'utilisation et la sécurité du dispositif intelligent avec le système de l'entreprise. Nous allons classer les applications mobiles Enterprise utilisation. Nous allons étudier les menaces possibles pour exposer les données de l'utilisateur de l'appareil intelligent et l'application d'une nouvelle attaque comme un exemple de ces menaces de capture d'écran. Enfin, nous allons étudier la technologie de virtualisation mobile et enquêter sur sa sécurité.

Keywords: Smartphone, Android, architecture de sécurité, vecteurs de menaces, analyse des risques

CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

INTRODUCTION

We live in the smart device era, people start use the smart device (phones and tablets) as the main computing device in daily basis. This revolution in technologies affects every computing system and shows the need of new systems architectures such as cloud computing and requires new security models to protect the users' data. Android is one of the leading platforms adopted by smart phones industry and it has 87% of the current used smart phones in the world (87). In fact, by the end of 2014 Android applications store "Google Play" has more than 850,000 applications with more than 40 billion download (52). Therefore, our research project focuses on Android platform. The wide use of smart phones encourages organizations and companies to adopt them in their enterprise solutions and to access their intra-networks. In fact smart phones have many benefits that could help improve enterprise solutions, such as mobility which allow immediate access to customer's data or feed the decision makers with real time information or provide quick feedback response. Therefore mobility can improve operational effectiveness and visibility across value chain, reducing operational cost of an organization, and enhancing decision making. However, when organizations start integrating smart phones to their enterprise solutions, they face many management and security challenges. First, the diversity of smartphone platforms is challenging the portability of the solution on various devices, especially for "bring your own device" BYOD approaches. Second, the mobility characterizing of smart phones make them easy to stolen which make organization data confidentiality under risk. Third the increasing number of mobile applications vulnerabilities discovered makes the mobile enterprise solution under high potential security risks. Indeed, the popularity of Android has attracted the attention of hackers and malware developers who spent double efforts to take advantage of smartphones vulnerabilities. In addition, the large users' community offered excellent opportunities for social engineering attacks. In fact, Google play, which is the Android's official market, is open to anyone to upload applications. While Google is putting efforts to verify the uploaded applications before publication, many 'malicious' ones succeed to integrate the market and get downloaded by thousands of end users before being unmasked. Consequently, the enterprise applications are executed in a hostile environment where any personal application installed by the end user can represent a source of attack! Unfortunately, the

base Android security model cannot protect from or even detect many of the attack scenarios that may threaten the security and privacy of enterprise applications.

- **Objectives:** The objectives of this research are the followings:

    a. Understanding mobile enterprise applications and the security risks they are facing on android mobile devices. This study includes a survey of existing security solutions and their evaluation.

    b. Experimenting security attacks on Android mobile devices. This study includes evaluating the security mechanisms of android devices and building attacks exploiting them in order to threaten enterprise applications.

    c. Proposing security enhancement to better protect privacy on Android mobile devices. The target security improvements vary from best practices to security configurations of and modification of the middleware managing mobile devices.

- **Contributions:** We summarize the contributions of our work as follows.

    a. we proposed a taxonomy for the enterprise mobile applications, based on which, we surveyed the current security solutions protecting them. This study revealed the risks facing enterprise applications on Android devices and the limitations of the proposed solutions. The study showed the emergence of mobile virtualization and multi-persona platforms to provide better isolation between enterprise applications and personal ones on the same device.

    b. we evaluated the security of mobile banking applications against screenshot attacks and investigated the possible protection mechanisms to defeat them. These attacks scenarios take advantage of password visibility feature on Android smart devices (largely used by other mobile platforms including IOS (55). According to our knowledge, this is the first research experiment testing the password visibility feature against real attack. During the study, we have experimented more than 130 mobile banking applications hosted on Google play store.

c. we studied the security weaknesses of multi-persona mobile technology, more precisely Cells solution (53) which is based on OS-level virtualization. During this study, we investigated three main attack scenarios: privacy escalation, privilege escalation and remote management. To counter the uncovered vulnerabilities, we proposed an SE-Policy module to secure the virtualization control management components in Cells. In addition, we defined and integrated a new MMAC policy to defeat the privacy escalation. Based on the proposed security mechanisms, we designed and implemented a prototype that has been integrated to Cells.

- **Organization:** The remainder or the thesis is organized as follows. In Chapter 1 We give a background on the Android platform security and privacy. Then we will explain the security requirements for using the smart devices in enterprise and we will survey the current proposed solutions to manage the security risks; in chapter 2 We present a new screenshot attack to expose the user credential from mobile banking applications on Android platform; in chapter 3 we will explore the security requirements and privacy of android virtualization technology and we propose a new Middle-ware Policy to Android platform to apply more fine grained access control on Android platform permissions; finally we explain our conclusion and recommendations.

CHAPTER 1

SECURING BUSINESS DATA ON ANDROID PLATFORM

1.1 Android Security

The Android platform uses the sandbox mechanism (9) to isolate applications and processes. The sandbox refer to mechanism by which applications run with their own set of user and group identifiers (UID and GID, respectively). The constrained manner in which applications execute make it impossible for one application to read or write data from another. To facilitate information sharing and interprocess communication among applications and processes Android platform has a set of *uses-permission* (6) to allows applications gain access to the system resources. Furthermore, any application can declare and enforce permissions to share its modules with other applications (13). The application's enforcing permissions and uses-permission should be declared in the application's Manifest file (2). The application Manifest file contains essential information about the application and it is written in XML format with predefined tags. During the application installation process, a prompt dialog contains the application's required permissions (uses-permission and enforce permissions) will appear to the smart device user, and the user should accept or reject the application installation. After application's installation, there is no possibility to modify or change the granted permissions to the application.

1.2 Private Data and Android Vulnerabilities

Users store their private data such as photos and emails on the smartphone without giving serious attention to the data's security. The high number of malwares detected for Android platform shows that there are weaknesses in the Android security architecture and development environment. We will explain the terms of smartphone privacy and the main attacks threatening them as follow.

1.2.1 Smart Phone Privacy

a. *Smartphone Identification.* Smartphone identifier could be phone number, international mobile equipment identity number (IMEI), International Mobile Subscriber Identity (IMSI) or SIM Card Serial Number (ICCID). Many applications use one of these unique identifiers to access their networks or check user identity. In real scenarios, these unique identifiers are bundled with username and password to create unique user identification in many applications. Consequently, accessing these unique identifiers facilitates attacking users' privacy since they are reused in other applications.

b. *Physical Location.* There are many applications providing Location Base Services LBS like map navigation, nearest services identification, location sharing in social networks. While the presence of these applications on the device is legitimate and justified, they cannot be trusted when the device is running enterprise applications. In fact, these applications can track the user physical location and thus may threaten enterprise privacy and even the physical safety of the user. For example, "consider a rural mobile accountant agent who uses an enterprise application to record transactions as he goes around customers and distribute money. Typically, any third party application that has the required Android permissions would have access to the location information. However, when the accountant agent is running the enterprise application, location information becomes enterprise sensitive data", and thus should be accessible only to enterprise applications. This is very important to protect the physical security of the accountant agent.

c. *External Storage.* Any data stored in the external memory storage is accessible to any other application having the permissions required to access external storage. In fact, the permissions of the majority of files stored in external storage are defined as RW or not encrypted, thus any application can access them or even modify them.

d. *Super User Access.* Rooting Android means granting the user full privilege to control the Android OS; known as root or super user access (91). The goal of rooting Smartphones is to overcome limitations that carrier and hardware manufacture put on the device. However the problem with rooted phones is that any malware successes to gain the root user

privilege will get full control of the Android OS. With root permissions, malware is able to access any application's data in the internal memory, user contact list or control the hardware function such as WiFi, Bluetooth, GPS, etc. All these malicious activities could happen without notifying the user.

1.2.2 Main Attacks

After specifying the user private data, we will explore the possible attacks that could expose the user private data.

a. *Privilege Escalation.* Android OS use IPC mechanism (Binder) (7) to let applications and system modules communicate with each other's. The authors of (58) investigate the permission escalation of IPC mechanism and showed that a possible security thread happens when "the application with less permission is not restricted to access components of a more permission application". In other words, Android's security architecture does not ensure that a caller application is assigned at least the same permissions as a callee application or component". For example, an application Ax having only a permission to access the user contacts list and there are another application Bx having a permission to access the internet and GPS functions, through the IPC mechanism Ax could use a component of Bx to send the user contacts list to third party server through Internet.

b. *Root Kit Attack.* Rootkit is software used by an attacker to gain root-level access to the Android OS. Rootkits allow hackers to administratively control the Smartphone, which means executing files, hiding processes, accessing logs, monitoring user activities, and even changing the Smartphone settings like GPS, Wifi, etc. Rootkits infect the Smartphone by installing themselves as loaded kernel modules LKM and thus, are loaded each time the operating system is booted up. The author of (137) shows that there is a technique to hook the system_call_ table through /dev/kmem access technique on Android platform which makes the attacker able to inject his malicious code. The rootkit attack in (138) shows an E-Finance service application that stores the public authentication certificate into internal saving structure in the SdCard. The rooting attack makes it possible

to acquire manager's authority and get access to every system file. Therefore, the attacker succeeds to get this authority and become able to expose the public authentication certificate of E-Finance service in order to use it later.

The wide use of Smart phones encourages organizations and companies to adopt them in their enterprise solutions and to access their intra-networks. In fact smart phones have many benefits that could help improve enterprise solutions, such as the mobile behavior which allow immediate access to customer's data or feed the decision makers with real time information or provide quick feedback response. Therefore mobility can improve operational effectiveness and visibility across value chain, reducing operational cost of an organization, and enhancing decision making. However, when organizations start integrating smart phones to their enterprise solutions, they face many management and security challenges. First, the diversity of smartphone platforms is challenging the portability of the solution on various devices, especially for "bring your own device" BYOD approaches. Second, the mobility characterizing of smart phones make them easy to stolen which make organization data confidentiality under risk. Third the increasing number of mobile applications vulnerabilities discovered make the mobile enterprise solution under high potential security risks. In this chapter, we will classify the uses of the enterprise mobile applications, then we will argue the security risks of the enterprise solutions with Android platform, and at the end we will survey and evaluate different solutions are proposed to enhance the security of Android platform.

1.3 Enterprise Applications Taxonomy

We focus on Android platform; however the taxonomy we propose could fit on all smart phone platforms. We adapted the enterprise mobile applications taxonomy proposed in (127). While the taxonomy in (127) addresses the issue of developing enterprise mobile applications; our taxonomy is security-driven. In fact, for each category of mobile applications, we discussed the main business scenarios and identified the associated security risks depending on data confidentiality and business data transactions point of view. The result presented in figure 1, a bottom-up five-layer classification where lower layers represent applications with lower secu-

rity risks and upper layers represent applications with higher security risks and these layers are overlapped, these layers are as follow.

a. **Public Broadcast Applications:**

These applications usually broadcast static information contents or general public services for a wide group of users, like company flayers, information about university campus, emergency exits, etc. This kind of applications usually produce public data with limited or without data transactions which mean low security risks in terms of data confidentiality.

b. **Information Applications:**

These applications are used to present products, offers, events, etc. thus they usually have heavy data transactions between end users and the enterprise solution. In terms of data confidentiality these applications could be used to capture the end users private data like interest search words, GPS coordinates, network provider, etc. From the company' side, the end user has accessibility to feed the enterprise solution with valuable data, for example th e integrity of users' feedback can be altered which could affect the validity and efficiency of its processes. Those both sides of data confidentiality make the system with high security risks to control and manage.

c. **E-Finance Applications:**

Applications with electronic money transaction could be a special kind of the previous category information application. Indeed, the enterprise solution has another actor to interact with rather than the end user, for example a financial mediator should be integrated with the system to complete the money transaction operation. In terms of data confidentiality the system has the same security risks as information application, however it has new actor to interact with, which adds new security risks to manage and control.

d. **Data Operation Applications:**

These applications are designed to achieve complex business scenarios for which any data operation should be performed between internal trusted users like employees and their company systems within the company's network. For example, applications that al-

lowing employees to submit their timesheets and tasks. In case of "BYOD model" where employees are allowed to use their own smartphone, these applications can represent a source security threats for the enterprise systems. Indeed the enterprise solutions should have ability to isolate and secure their own data transaction from the employee personal use, which is not easy to achieve as we will see later in this paper. In terms of data confidentiality the company data could be exposed through employee device, which means high security risks to manage and control.

e. **Collaborative Applications:**

These applications could be considered as a special kind of the previous category "Data Operations" however, they are designed for collaborative purpose between trusted users such as video conference services, instance file sharing applications, etc. In terms of data confidentiality the system has the same high security risks as the previous category but with more complex scenarios to manage.

In terms of business work-flow transition, the first three categories could be fit under B2C "Business to Consumer" (68) model and the last two categories could be fit under B2E "Business to Employee" (68) model. We focus our work on the last two categories B2E.

1.4 Security Requirements

In the following we will explore the security requirements that enterprise systems need to control the enterprise mobile applications security risks.

a. **Policy Management:**

- The enterprise solution should has the ability to enforce and validate a security policy to any smartphone will communicate with it.

- Fully controlling the smartphone while it is communicating with the enterprise solution and releasing this control once disconnected.

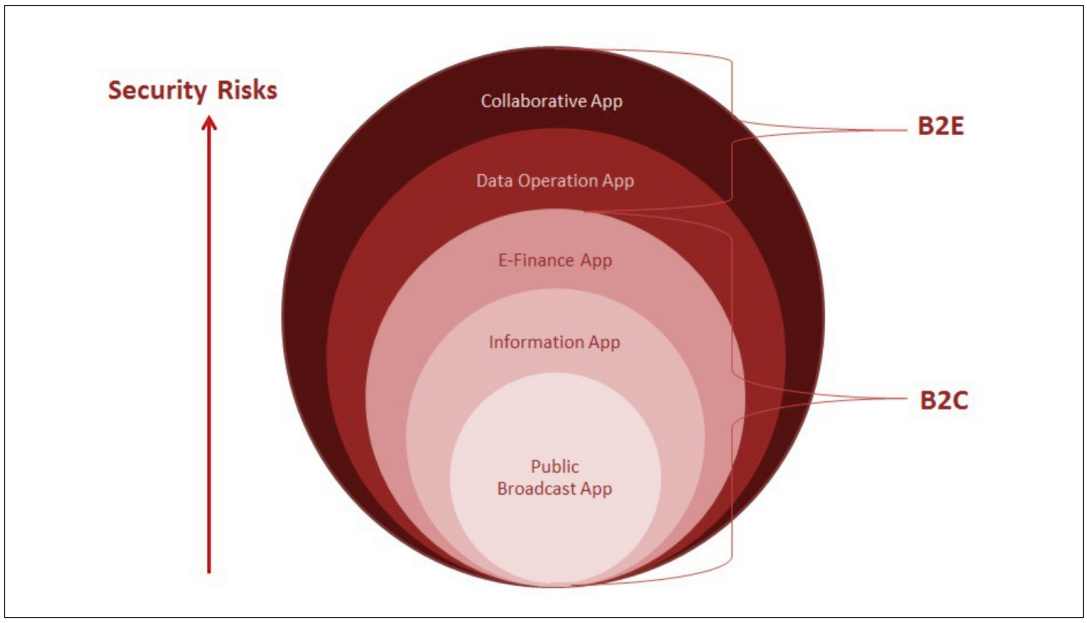


Figure 1.1 Taxonomy of Enterprise mobile Apps and their security risks

b. **Data Isolation:**

- Enterprise solution should provide data isolation between its own stored data and other personal data of the user.

c. **Data Encryption:**

- Provide encryption function for data and commands.

d. **Secure Communication:**

- Establishing secure communication parameters between the smartphone and enterprise solutions are required.

e. **Easy Integration:**

- Ensuring smooth integration with existing systems.

f. **Performance and Overhead:**

- Introducing as less overhead as possible on the smartphone and on the enterprise solution architecture.

1.5 Security Solutions from Academia

In the following, we will explore and evaluate the most common academic ideas that proposed to enhance the security of enterprise mobile application usage.

1.5.1 Trust Droid

Trust Droid (59) is a proposed solution to provide data isolation and applications policy management at different layers of the Android software stack. Trust Droid assumes that the enterprise networks are trusted and the employee is trusted. However, smartphones are generally not trusted because employees are prone to security critical errors such as installing malware or disabling security features. The main idea of Trust Droid is to classify the applications into three predefined categories; or "Colouring them" as the authors mentioned (59). The first category gathers per-installed system applications like the content providers and services, the second category is trusted third party applications provided by the enterprise systems, and the third category is dedicated to untrusted third party applications such as any application installed by the employee for personal use. This classification is performed by checking the application certificate at installation time. Based on this classification, each application will run under its category's policy where the untrusted applications cannot access resources or communicate with applications belonging to the trusted categories. Trust Droid provides mandatory access control MAC (77) for each category domain to control files accessing and sharing. In addition, Trust Droid provides a firewall to control network sockets and Internet protocol; the firewall rules are based on the policy of each category. *Evaluation:* from security perspective Trust Droid could be vulnerable to runtime attacks like buffer overflow because the TCB model of Trust Droid assumes that the low level system layer is secure. In addition, if an adversary identifies vulnerabilities in one of these pre installed applications, he could break the domain isolation and get access to data belonging to trust applications. Trust Droid has other weaknesses in managing enterprise dynamic data, for example it cannot prevent an enterprise application from uploading personal files, also managing categories policy limited to application development phase, and it cannot be updated after installation which limits the evolution of the

solution over time. From performance perspective, based on section 5.2 of (59), Trust Droid introduces acceptable performance overhead. Finally, there is no data encryption functionality to be managed by the solution.

1.5.2 Unified Security Enhancement Framework

Authors of Unified Security Enhancement Framework (97) propose a unified and effective kernel-level framework to secure the Android OS by introducing three mechanisms: first is Root Privilege Protection (RPP) which keeps track of a list of trusted programs with root privileges, second is Resource Misuse Protection (RMP) which keeps track of important system resources that are vital in the Android OS and finally Private Data Protection (PDP) which disallows trusted programs to access sensitive data through enforcing the least privilege principle in the permission based access control. *Evaluation:* from security perspective the framework was evaluated by conducting three experiments based on three malicious behaviors, first one is gaining root access, second is changing system resources such as configuration files which is always target by attacker, last one is trying to access user private data such as contact list. The framework succeeded to prevent all these attacks; however the framework just narrows the threats attacks possibilities because any malicious application installed by super user can inherit all root access. The RMP mechanism restriction related to phone configuration itself which cannot be modified by any application and contradict the case in the enterprise applications which need more flexibility to prevent conflicts between applications. The system does not have remote management capabilities for installed applications. The performance showed in section 5 (97) that monitoring files and system calls by the framework component increase the overhead by almost 25% average rather than the normal Android OS.

1.5.3 Polite Policy Framework

The proposed idea of polite policy framework (94) is to control applications behavior at execution time. Indeed, developers use a modified API to provision security at the build phase of the application which enables polite framework to achieve fine grained policy control as well as be easily adopted in the mobile application development life cycle. Policies are dynamically

fetched from an enterprise policy server at runtime which gives the enterprise administrators high management capabilities. When an enterprise application starts, the system creates a parallel thread that monitors the phone state and stops or kills the application if the enterprise policies are not met. The advantage in this solution is that the enterprise policy is enforced only during the enterprise applications executions time; thus, the personal use of the device will not be affected. *Evaluation:* from security perspective, the solution has two types of policies; first is API level policy which manages the device resources such as GPS, WiFi, etc. and the second is application level policy which manages the data flow. Also Polite framework gives the developer data encryption functionalities to secure enterprise data and perform remote device wipe functionality for device stolen case. However, the policy classification does not prevent system component itself to be exploited under root access attack or privilege escalation attack. Another weak point is the enterprise application itself is responsible for the communication with the policy server and policy enforcement! That could lead to policy conflict between the enterprise applications and it could increase threats on policy server. For performance, section VI-c (94) shows that polite framework takes 6% more time than normal native library command execution.

1.5.4 CRêPE

CRêPE (62) is a solution for Enforcing Fine-Grained Context-Related Policies on Android. It acts as a security mechanism in addition to the standard Android security and allows users and other predefined trusted parties to define context-related policies which can be installed, updated, and applied at runtime. These policies can be applied in a fine-grained manner, e.g. for each application the context-related policy consists of two types of policies:

a. Access control policy: access rules that use the XACML standard (100).

b. Obligation policy: specifies the actions that should be done such as activate or disable a system resource like GPS, Wifi, camera or start and stop applications.

The system component manages policies check at all android stack layers: application layer, framework layer, and kernel layer. Moreover, the user is able to manage policies rules, i.e.,

"create, update and delete" locally form the device (through GUI applications) or remotely via SMS message, Bluetooth and QR-code using public key infrastructure PKI schema. *Evaluation:* CRêPE extends the permission check mechanism in android OS by adding further checks to the current active CRêPE policy, this approach can be considered as prevention of privilege escalation attacks. Also CRêPE has CRePEIPTables component which is working in the kernel level as firewall to filter the network access. However there are two weaknesses in this solution; first defining the policy depends on the user orientation of security needs which is not enough due to the user miss understand of security risks; second CRêPE protects the communication between its components by PKI system using X.509 certificates stored as root authority, a rootkit attack as described in (138) will be able to expose the authentication certificate and the system will be vulnerable to any income message. For performance, Section VI-B in (62) shows that policy activation and deactivation overhead is influenced by the number of conflict rules defined in the policy, but the overall efficiency overhead is acceptable.

1.5.5 SE for Android

Android has a Linux Kernel OS and thus it relies on the Linux discretionary access control (DAC) to implement the permission model of Android security architecture. Indeed, Android uses DAC in two ways:

a. Sandboxing technique to provide data and code execution isolation between applications (9).

b. Authorizing applications to access system resources like Wifi, GPS, etc.

Security Enhanced Linux (SE Linux) (77) was originally developed as a Mandatory Access Control (MAC) mechanism for Linux. The goal of MAC is to allow the OS constraining the ability of a subject/initiator to access or generally perform some sorts of operation on an object/target depending on a wide set of rules. *Evaluation:* Authors of (120) evaluate SE Android by investigating previously published malwares and vulnerabilities. Regards to the root exploits malwares, SE-Android is able to prevent some of applied rootkit attacks such as Ginger Master (15) and Zimperlich (118). However, the SE-Android is not able to prevent other attack such

as KillingInTheNameOf and psneuter exploits (16). Regarding the application layer vulnerabilities, SE-Android provides an effective means of preventing applications from performing privilege escalation attacks and unauthorized data sharing through kernel interfaces. However, SE-Android has some limitations that should be considered. First, the effectiveness of its security depends on the defined policy which means SE-Android cannot mitigate anything not defined in the policy rules. Second, SE-Android was not able to mitigate Kernel level vulnerabilities applied by rootkit attacks. Third, SE-Android cannot protect against threats originating from shared hardware resources. For performance, the SE-Android overhead was negligible compared with normal Android OS.

1.5.6 L4Android

L4Android (96) is a microkernel derived from the L4Linux (95). L4Android created by adding the required code for Android platform to the L4Linux components. L4Android divides the OS kernel functions into small components, each component implements one basic service and is equipped with only permissions needed for its correct operation. The goal of L4Android is to run the Android platform as a virtual machine on the top of the microkernel. Thus the smartphone can run two Android OS; one will be use for personal purpose and the other will be use for business purpose. *Evaluation:* Virtualization provides high domain data isolation between business partition and personal partition as long as the business partition is not infected. For policy management perspective, the enterprise system will need to apply different policies on the business platform as there are many use cases. For performance, L4Android has more execution time than the normal the Android OS and related to (1) it has performance issues with graphic driver components. There are many hypervisor developed to achieve the same domain data isolation technique like Xen (76), KVM (64) and OK4L Android (66). Virtualization provides secure environment and data isolation. However, the performance, power efficiency and implementation coast will be barriers to be applied.

1.5.7 Improving Security with OS-Level Virtualization

Linux Containers (LXC) (65) is a lightweight OS level virtualization that isolates processes and resources without the need to provide instruction interpretation mechanisms and other complexities of full virtualization such as the hypervisor virtualization mechanism. Authors of (130) provide a user-space container to isolate and control the resources of single applications or groups of applications running on top of the Android OS kernel. This typically includes a unique hostname, process identifiers (PIDs), inter process communications (IPC), a file system, and network resources. *Evaluation:* The major advantage of OS-level virtualization technique compared to full system virtualization is sharing the kernel layer between different containers decrease the virtualization overhead meanwhile provides isolation between user-spaces' data and process execution. The system has a remote management component integrated within the kernel level so it is isolated from Android user-space. Also encryption functionality could be integrated at the user-space level to encrypt the file system or by a file basis. For performance, in section-9 at (131) the evaluation test shows that negligible performance overhead happen compared with the original Android OS performance.

1.5.8 Matrix of Proposed Solutions and Security Requirements

In this matrix we relate the proposed solutions' features with the suggested functions to cover security risks we considered. We used three symbols: $(-)$ mean this function is weak or does not exist, $(*)$ mean this function has good implementation, $(+)$ mean this function needs more enhancement.

1.6 Security Solutions from Industry

Many smartphone manufacture, telecommunication providers and security solution companies start collaborate together to provide a complete secure solutions for managing smartphone in Enterprise systems using the same ideas as we presented in section 4. Samsung, one of these smartphone manufactures, provides a complete solution for secure smartphone management in enterprise system named KNOX. KNOX (113) is an Android based solution built on the

Table 1.1 MMAC Policy Tags and Description

	Policy Management	Data Isolation	Data Encryption	Secure Communication	Easy Integration	Performance and Overhead
Trust Droid	+	*	−	*	+	−
Security enhancement framework	−	+	−	+	−	+
Polite Policy Framework	*	+	*	−	*	+
CRêPE	*	*	−	*	+	+
SE-Android	*	+	−	+	+	+
L4Android	+	*	−	−	−	+
OS-Level Virtualization	+	*	−	+	+	*

integration between hardware layer using Trust Zone (TZ) technology (56) with software layer using SE for Android and Android Containers technology. The advantage of using TZ technology is partitioning the memory and CPU resources into a “secure” and “normal” world, which allow TZ based Integrity Measurement Architecture (TIMA) to continuously monitor the integrity of the Linux kernel. TIMA runs in the secure-world and cannot be disabled, while the SE for Android Linux kernel runs in the “normal” world. The Android Container provides isolation between the enterprise programs operation environment and other programs operation environment like the OS level virtualization solution we discussed in previous section. Finally, the system has data encryption function that allows enterprise IT administrators to encrypt data on the entire device and has virtual private network (VPN) support to establish secure communications. There are other companies providing secure enterprise mobile solutions like

VMware (129) and TrendMicro (102), however all these solutions still new in the industry and need times to be evaluated.

CHAPTER 2

CAPTURE-ME : ATTACKING THE USER CREDENTIAL IN MOBILE BANKING APPLICATIONS

Mobile banking is a service that allows the customers of financial institutions to conduct a number of financial transactions such as pay bills and transfer money through a smart device. Moreover financial institutions use the mobile banking service to inform their customers about the new services and offers they provide. The mobile applications vulnerabilities and security threats that discovered and reported every year (39) make the financial institutions pay attention to their mobile banking service security and safety. A few studies have shown that the smart phone screen is vulnerable to privacy exposer by using the spying attack such as ``iSpy'' (108) and ``fast eavesdropping attack'' (101) or by applying screenshot attack such as ``Screenmilker'' (99). These attacks were able to expose the user typing data due to the advantage of the keystroke animation feature on smart phone keyboards. Figure 2.1 shows the keystroke animation feature. The screenshot attack Screenmilker used the keystroke ani-

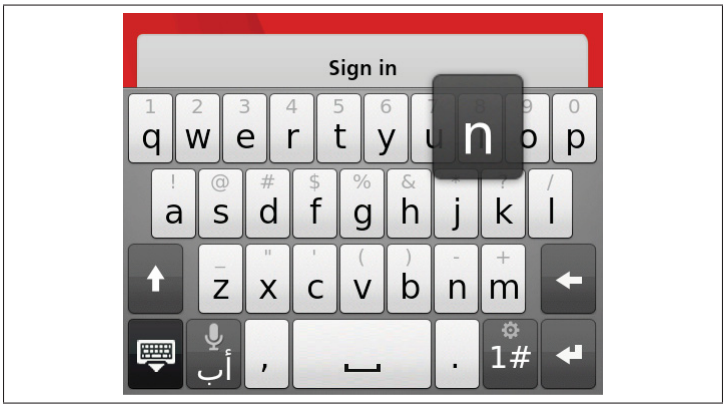


Figure 2.1 Keystroke Animation Feature on Keyboards

mation feature to expose the user credential data (user name and password), contacts list and social applications posts. However, these attacks either screenshot attack or spying attack will fail if the keystroke animation feature is disabled by the user or in smart devices with screen size more than 6 inches, such as tablets where the keystroke animation feature is disabled by

default. In this chapter we introduce a new screenshot attack, Capture-Me , to expose the user credential (user id and password) by using the advantage of the password visibility feature on Android platform. The password visibility feature controls whether passwords typed into the smart device are visible on screen, or hidden by replacing the letters with dots. The smart device users usually have difficulty with typing letters accurately using the on-screen keyboard. The password visibility feature will give a visual feedback on-screen in the password text field. The password visibility feature shows the typed letter for 0.5 to 1 sec (depending on the user typing speed) and then convert the letter to a dot. Figure2.2 shows the password visibility feature on a mobile banking application. Most of the mobile apps required a login procedure to

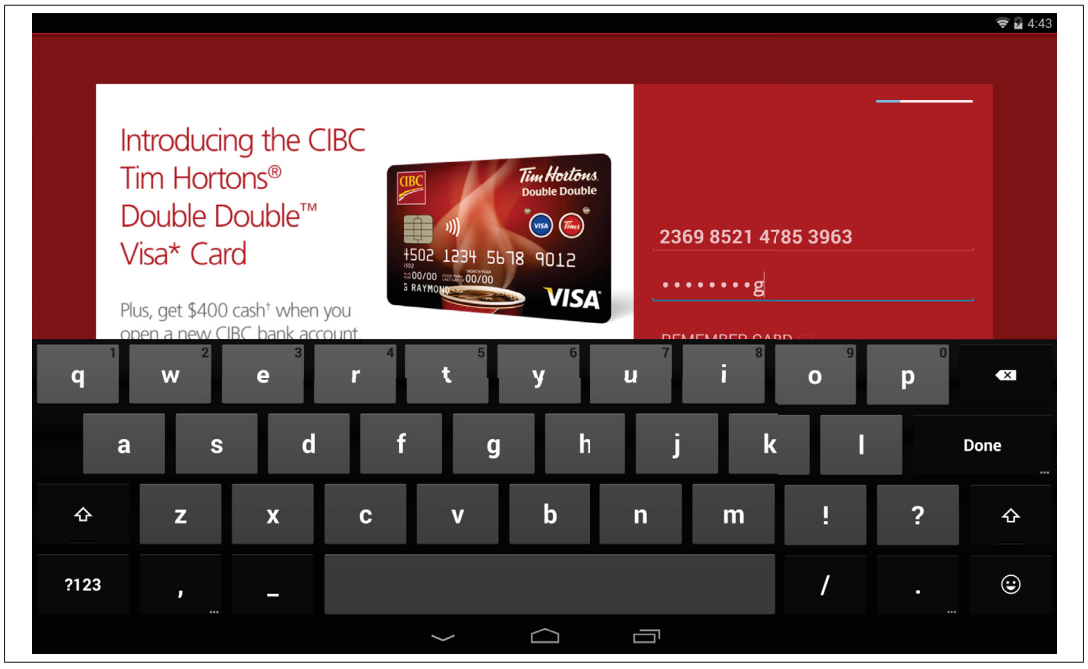


Figure 2.2 Password Visibility Feature

let the user use its features and services. However, mobile apps are swinging between security and usability. Mobile apps such as social networks, email, chat, etc., give more attention to the usability rather than security. These kinds of apps only require the user to enter her credential (user id and password) at the first time the application runs. Then, the application stores the user credential in the data directories and the user will not need to enter her credential the next

time the application runs. Nevertheless mobile banking apps pay more attention to security and require the user to enter her credential (user-id and password) every time the mobile banking application runs. Even with the remember me option the user require to enter her password every time the application runs. Capture-Me focuses its attack on mobile banking apps because of the repetition of entering the user credential (user id and password) which increases the attack success space. Capture-Me will take a series of screenshot images while the user enters her credential in the mobile banking application and apply OCR analysis on those screenshot images using the tesseract-ocr engine. The tesseract-ocr (85) is one of the most accurate open source OCR engines and is used in many mobile apps such as (111) and (126) to recognize text from images. Tesseract-ocr engine is an independent platform, it runs on Android, IOS and could be compiled to run on other platforms such as windows phone. One of the key features in tesseract-ocr is the training procedure (31) to support languages other than English. In fact, the training procedure feature is the advantage that makes Capture-Me use the tesseract-ocr engine instead of use any other OCR engine such as Ocrad (67) or GOCR (117). We used the tesseract training procedure to create a new language dataset, which consisted of the English language letters, password masking character (dots) and keyboard cursor (vertical bar), and named it as password language. Capture-Me combined the tesseract engine with the password language dataset to expose the user credential data during the OCR analysis phase. We will show later in 2.2.4 the procedure of the training process and the needs of our new password language dataset. Our development of Capture-Me attack shows that most of the mobile banking apps in Google play store are vulnerable to Capture-Me attack and they did not implement the basic security protection mechanism to defeat the screenshot attack. We explore the possible protection mechanisms to defeat the Capture-Me attack with more than 130 mobile banking apps hosted on Google play store.

2.1 Screenshot Functionality

Smart device users mainly take screenshots for social activities and for that reason many screenshot applications such as Screenshot UX (121) and Screenshot Ultimate (86) have high download rates and good reviews on Google Play store. Before we explain how these screenshot

applications work on Android platforms, we will give a brief introduction about the Android Graphics framework.

2.1.1 Frame Buffer FB

Linux Frame buffer (128) is an abstract component that provides access to the graphical output from the graphic hardware device. Android relies on a standard frame buffer deriver which exists in the root directory of the Android platform under dev/fb0 or dev/graphics/fb0. The fb0 directory is only accessible by the root user and graphics group's users. The output of FB consists of the screen pixel data and its format, which requires doing an I/O ioct (17) control operation to extract the fb0 data. In more technical details, the frame buffer has front and back buffers. The front buffer has pixel data of the current surface presented on the screen, and the back buffer is used for composition by the surface flinger service(50).

2.1.2 Application Surface view

Each application window on the Android platform has its own surface view (51), which holds the pixel data that are being composited to the screen. The displayed screen holds many applications' surfaces, like the foreground application activity, the status bar and the navigation bar. All these application surfaces are managed and displayed on the screen by the surface flinger service.

2.1.3 Surface Flinger service

The surface flinger (50) is a wide composition engine that runs as a daemon system service on the Android platform to perform the main following functions: 1) Composite multiple surfaces in a single frame buffer 2) Pass the composite frame buffer to one or more displays 3) Manage and synthesize the composite buffer allocation and data. When an application comes to the foreground; the surface flinger communicates with the window manager (40) service to receive the window status (visibility, z-order) and communicate with the activity manager (35) service to receive the foreground activity status. These communications are done via binder interface ISurfaceComposer(104) which help the surface flinger to collect the required data before

compose the frame buffer. Figure 2.3 shows the sequence diagram of surface flinger service operations. Moreover, the surface flinger is not available to third party applications. Only required system services or system signed applications with ACCESS_SURFACE_FLINGER permission (33) are able to communicate with the surface flinger service.

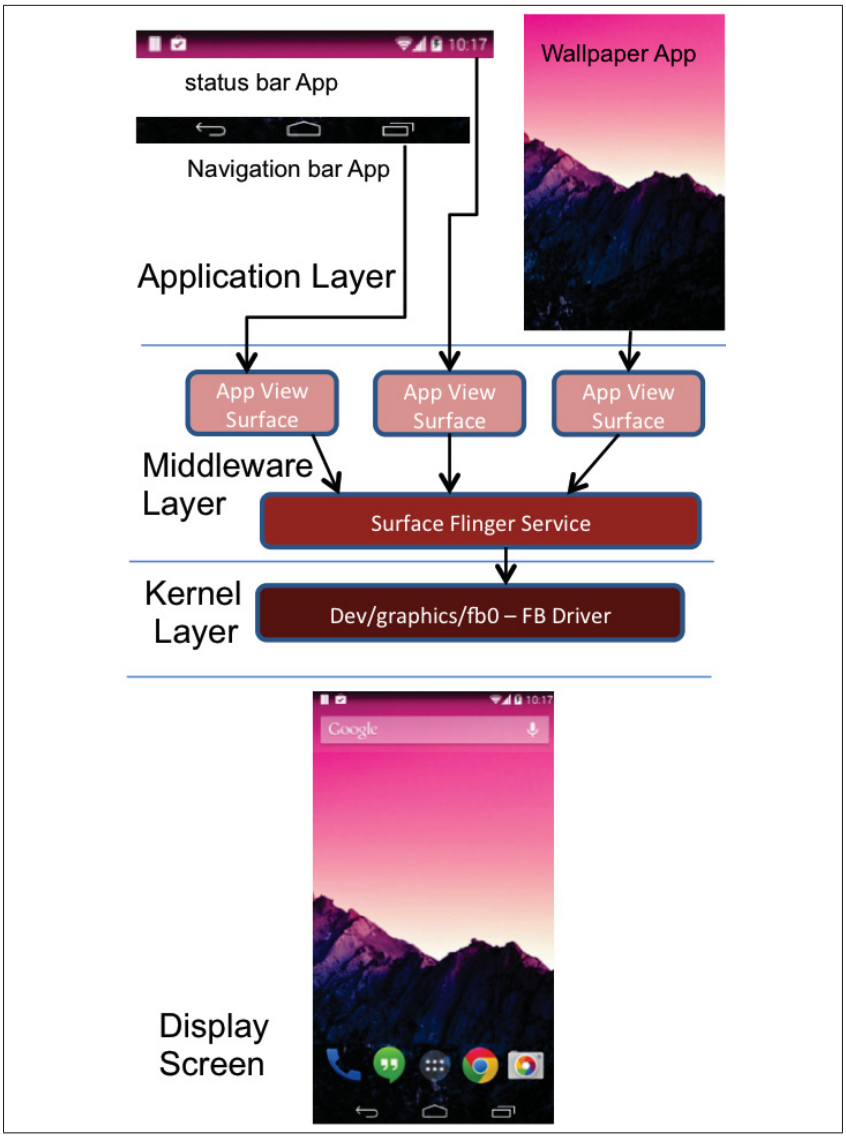


Figure 2.3 Surface Flinger Service sequence diagram

2.1.4 Screenshot implementation on Android

There are two techniques to take a screenshot image on the Android platform; a) by using the Screencap command (54) which is a native system function b) by executing an I/O ioctl on the frame buffer (dev/graphics/fb0) to read the current active display screen frame buffer data.

a. **Screencap:** Screencap is a command line function in the Android framework, which allows authorized system applications and the Android Debug Bridge ADB (3) to take a screenshot image. In more detail, the Screencap command communicates with the surface flinger service through the ScreenShotClient object to read the frame buffer of the active screen. Only authorized system applications, ADB, and any process that has a super user permission are able to execute the Screencap command.

b. **FB deviec (dev/graphics/fb0):** Before Android 4.x versions, the Screencap function did not exist. For that reason many screenshot applications such as screenshot library (89) develop their own implementation to access the fb0 and read the frame buffer data of the displayed screen on Android platform 2.x versions.

2.1.5 ScreenShot Applications

We choose the four most downloaded screenshot applications on the Google play store and performed a reverse engineering on each one of them using apktool (88) and JAD GUI (69) to discover how they get access to the Screencap function or FB device. Table 2.1 shows the screenshot applications that have the highest download rate in Google play store. We

Table 2.1 The highest Downloaded Screenshot Applications

Application Name	**Download Rate**	**Feedback**
ScreenShot UX	10,000,000-50,000,000	4.3 star
ScreenShot Ultimate	5,000,000-10,000,000	3.9 star
ScreenShot	10,000,000-50,000,000	3.6 star
ScreenShot Easy	1,000,000 - 5,000,000	4.3 star

installed these applications on different Android platform versions: Android 2.3.6 unrooted/-

rooted, Android 4.1 unrooted/rooted and Android 4.4.2 unrooted/rooted. We found that all of these applications use different techniques to take a screenshot image depending on the running environment platform.

- **On Android 2.3 unrooted phones:** these applications rely on an ADB vulnerability in Android 2.3 platform, which lets the user run a native executable service on the smart device with ADB signature. The procedure for this technique consists of 3 steps:

  a. Install the screenshot application from play store.

  b. Connect the smart device to the user PC through USB connection then download and run a support patch by the screenshot application.

  c. The screen shot application will communicate with the native service through TCP socket connection to retrieve the frame buffer data.

  Moreover, we performed a reverse engineer on the support patch of the screenshot applications and we found that this patch consists of an executable file and a shell script. The shell script contains the ADB commands to copy the executable file to the smart device then executes it to run as a native service. The native service will get access to the frame buffer device (the dev/graphic/fb0 file) as the owner of the native service process is the ADB (3).

- **On Android 4.1, 4.3, 4.4 unrooted phones:** some of these applications mentioned in their description on Google play store they might work using the same technique as Android 2.3 unrooted platform, but all our trails on Nexus 7, Nexus 10 and Samsung Galaxy Note 10 did not work, as many Xda developer forums mentioned (84) (57).

- **On Android 2.3 rooted phones:** Screenshot applications use the ADB vulnerability method. They can also rely on the super user's (91) permission to gain access to the frame buffer device fb0 and execute their own implementation of reading the frame buffer data.

- **On Android 4.1, 4.3, 4.4 rooted phones:** Screenshot applications rely on the super user's permission to execute the Screencap command or to gain direct access to the frame buffer device fb0. Some of these applications ask the user to perform ADB vulnerability tech-

nique, however the ADB should gain the super user's permission to be able to copy and execute the native service. Table 2.2 shows a matrix of all Android platforms we tested and the corresponding screenshot techniques that work with them.

Table 2.2 Matrix of Screenshot techniques and relative Android Platform

	Android 2.3	Android 2.3 Rooted	Android 4.x	Android 4.x Rooted
ADB TCP Method	*	*	-	*
ScreenCap function	-	-	-	*
Direct access to fb0	-	*	-	*

2.2 Design and Implementation

Capture-Me runs as a background service on the Android platform to perform the following actions:

a. Monitor the current foreground application on the smart device

b. Take the right screenshot when the user starts entering her credentials on the mobile banking applications.

c. Perform an OCR analysis on the taken screenshot images to extract the user credentials.

d. Send the extracted user credentials to the attacker server.

2.2.1 Attack scenario

In our attack model, we consider an adversary who can disguise Capture-Me in his attractive app or convince the user to install Capture-Me as a separate mandatory application component to run his app (Many applications on Android Play Store required the user to install dependency component as another application hosted on the Play store). Capture-Me required Internet access permission to send the extracted user credential to the attacker server and the user should

have installed at least one of the targeted mobile banking applications. Figure 2.4 typically shows Capture-Me threat scenario steps.

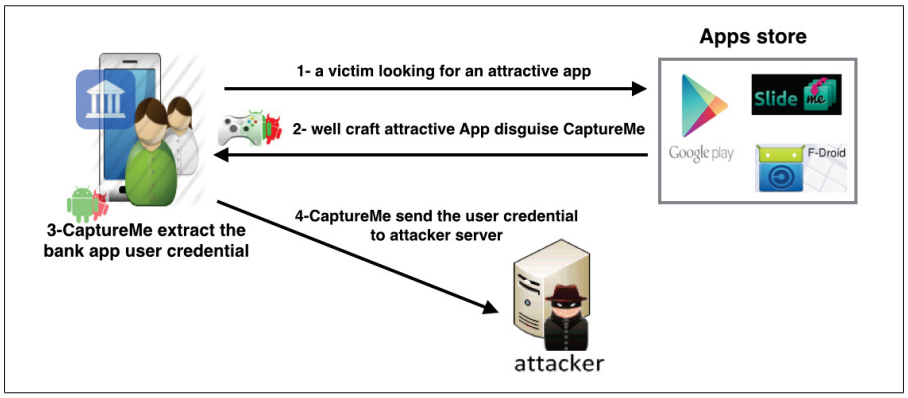


Figure 2.4 Capture-Me Attack Scenario

2.2.2 Preliminary Design Considerations

We analyzed six mobile banking applications and the PayPal mobile application (as Paypal is a financial institution) to investigate these applications operation steps and behavior. Capture-Me will target those chosen mobile banking applications to test the attack scenario. The mobile application package name is a unique identifier, Capture-Me relies on it to detect whether the target mobile banking application is invoked and placed on the foreground display to start capturing the screenshot images. Capture-Me has the mobile banking applications package names in its configurations. During the analysis of the chosen mobile banking apps, we found some of these apps work in the off-line mode (i.e., without requiring an Internet connection); they start with showing the login screen, and after the user enters her credentials, they check for the Internet connectivity. Other apps only work if there is an Internet connection. This information helps Capture-Me to decide the right time to take the screenshot image. Furthermore, in the login screen of each mobile banking application we checked the user id types (i.e: card number, user name or email) and validate the password text field length. Some of those apps have a length validation on the password text field and others do not. The password text field length information helps Capture-Me to determine the exact password length as there are some mobile

banking applications required the user for 6-8 password characters. Relevant information from the selected mobile banking apps are summarized in Table 2.3.

Table 2.3 List of Selected Bank Applications

Application Name	**Package Name**	**Password field Length**	**User ID**
CIBC Bank App	com.cibc .android.mobi	12 chars	Card number
BMO Bank App	com.bmo .mobile	6 chars	Card number
Natioanl Bank App	ca.bnc .android	26 chars	Card number
RBC Mobile Bank App	com.rbc .mobile.android	32 chars	Card number / User name
ScotiaBank App	com. -scotiabank .mobile	More than 50 chars	Card number
TD Canada Bank App	com.td	32 char	Card number / User name
PayPal App	com.paypal .android .p2pmobile	More than 50 chars	Email address

2.2.3 Monitoring and Screen Capture Phases

Android platform has useful resources we can use to identify the current foreground application process such as the Top command (98), which gives continual reports about the running processes in the Android system. By examining the usage of the CPU and memory, we can identify the current foreground application. However, the easiest technique to identify the foreground application process is to check the IMPORTANCE FOREGROUND (11) attribute of application process that can be found in the applications processes list in the activity manager service (35). Capture-Me monitors the current foreground application process by using the activity manager technique. When the foreground application package name matches one of the target mobile banking apps,Capture-Me initiates the screen capture phase. We incorporate all existing screenshot capturing techniques (see Section 2.1) in Capture-Me , and based on the detected working environment, Capture-Me selects the best technique to use. We also analyze the performance of each technique with our experimental devices on two different Android platforms by measuring the required time to take a screenshot image and store it on the

Capture-Me directories; see Table 2.4. The screenshot image size depends on three factors; 1)

Table 2.4 Time required for screenshot Technique on different Android platforms

	Android 2.3.6 HTC Desire HD	Android 4.4.2 Nexus 10
ADB, TCP Socket technique	1 sec	1.1 sec
ScreenCap technique	-	1.4 sec
Direct access to fb0 technique	0.6 sec	0.8 sec

the device screen size, 2) the target application window content (images and colors) and 3) the image resolution. We run the screen capture phase in Capture-Me multiple times with different image parameters (e.g; resolution and dpi) to adjust the best suitable image size and quality that Capture-Me can take. The average screenshot image size is between 80–140 KB with a resolution of 2560 x 1600, and 15–70 KB with a resolution of 480 x 800 in Nexus 10 and HTC Desire HD, respectively. Capture-Me stores the screenshot images that are taken from every session in different directories until the OCR analysis phase is completed.

2.2.4 OCR Analysis Phase

Capture-Me starts the OCR analysis phase after the target mobile banking application is terminated. The tesseract-ocr engine (85) works in a step-by-step manner to extract the text from the image file as figure2.5 shows. The first step is to convert the given image file to a binary image. Next step is to determine the text layout e.g; horizontal or vertical and left or right. Then the connected component analysis step is responsible for determining and extracting the characters outlines. In the finding lines and words step, the characters outlines will convert to blobs. These blobs are organized horizontally as the text lines and the blobs lines and regions are divided into fixed areas or are relative to the text size. Then the word recognition occurs in two passes; during the first pass, an attempt is made to recognize each word from the given blobs. Each word that is satisfactorily recognized will pass as training data to an adaptive classifier. The adaptive classifier enhances the text recognition in a more accurate manner. In the sec-

ond pass, as the adaptive classifier has now learned new things about the words and characters shapes from the previous pass, the second pass is to resolve various issues and to re-recognize the skipped text from the first pass. More details regarding every phase are available at (123) and (122).

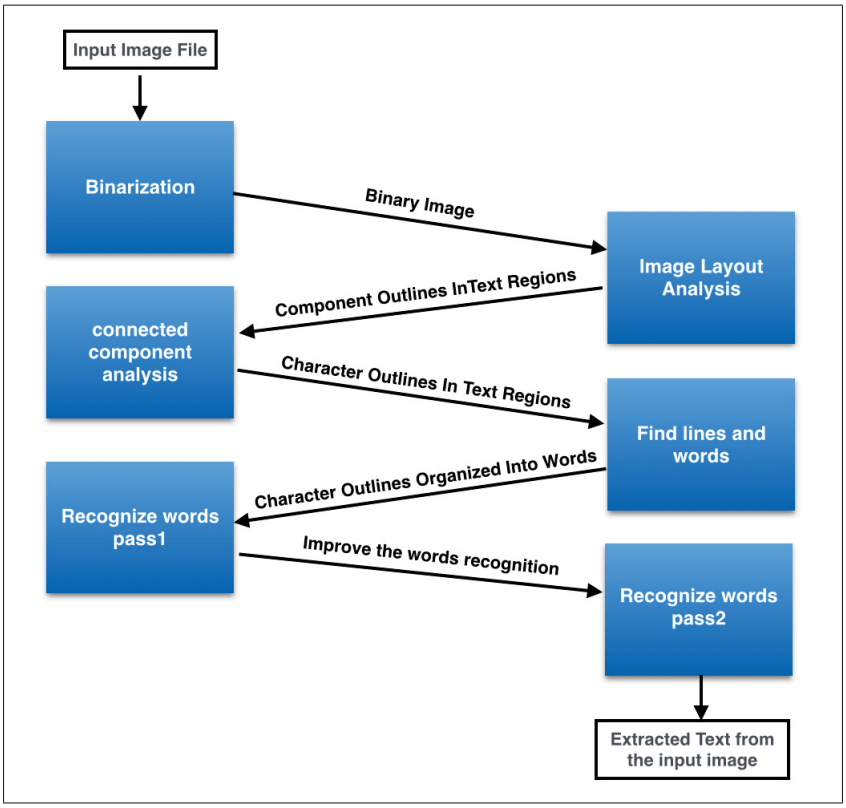


Figure 2.5 Block Digram of Tesseract OCR Architecture

2.2.4.1 Password language

Initially Capture-Me used the tesseract-ocr engine with the English language dataset to extract the user credential (User ID and password) from the taken screenshot images. Although it succeeded to extract the User ID, it failed to extract the password characters with high error rates. When investigating the challenges that the tesseract-ocr engine faces with password characters there are two points to consider: (a) the password masking character and keyboard cursor are special characters and they are not considered in the English language dataset and (b) the

keyboard cursor appears so close to the typed character that it prevents the tesseract-ocr engine from recognizing the difference between the typed character and the keyboard cursor. To better extract the password characters, we designed a new language data set to use with the tesseract-ocr engine and called it the password language. Letters in the password language consist of English letters, some of the most used special characters (e.g :$), the password masking character and the keyboard cursor character, both in different shapes and sizes. Table 2.5 shows the different shapes of the keyboard cursor, password masking characters and their corresponding Unicode.

Table 2.5 Password masking and keyboard cursor shapes

Password Masking	**Keyboard Cursor**
· U+0387	ǀ U+2758
· U+00B7	| U+007C
· U+00b7	| U+2016
• U+2022	| U+2225

- **Valid Words:** In the password language, there are several restrictions on the valid word: any word cannot have more than one English letter; a word may consist of a keyboard cursor character and at least a password masking character; the word containing the first typed character in the password text field, consists of an English letter and the keyboard cursor character. Finally, we included some reserved words that exist in any mobile banking app login screen such as Login, password, remember me and card number. Figure 2.6 shows different samples of password language words.

- **Tesseract-ocr training procedure:** An important feature of the tesseract-ocr engine is the training procedure to support languages other than the English language. Before we start the training procedures, we setup the password language dataset and the words dictionary into text files to be an input to the training procedure. The training procedure guide is well explained at (31), we will briefly describe the training procedure steps as follows:

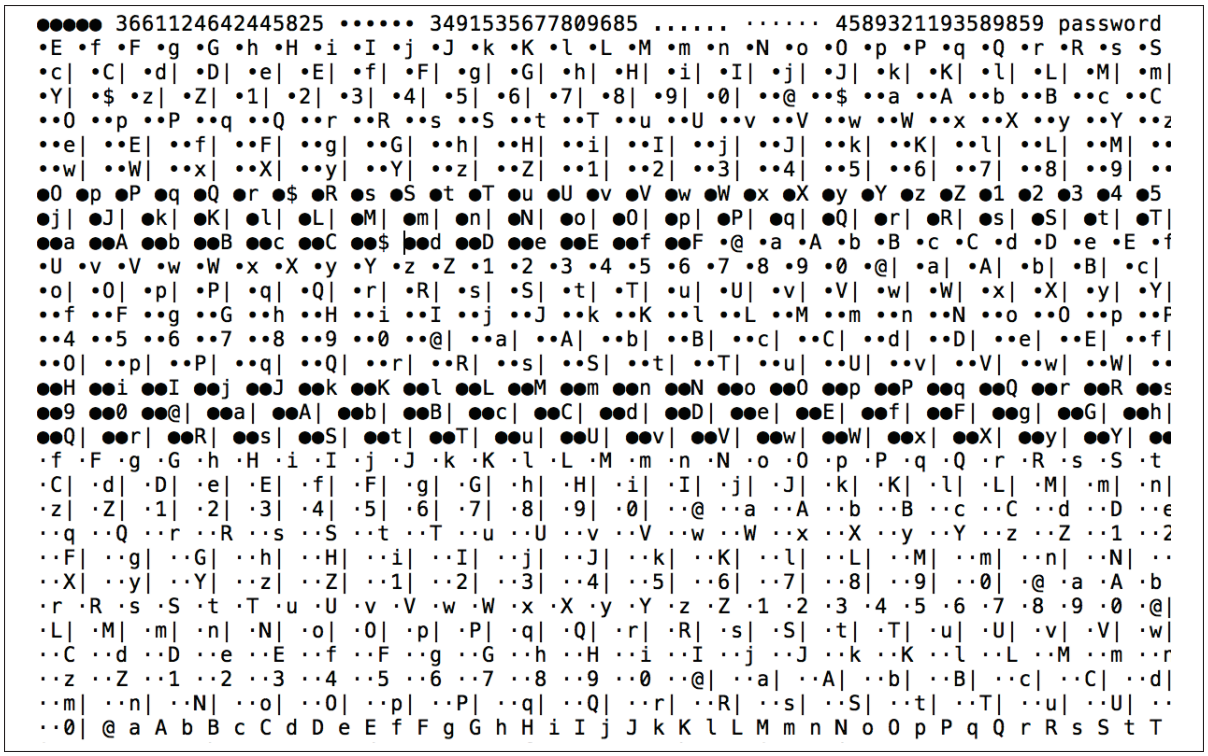


Figure 2.6 Sample of Password Language Words

a. Generate Training Images: The tesseract-ocr engine creates a text image from the given language text with a specific font. For each new font, tesseract recommends that a new image file be generated. We trained tesseract with three different fonts (Arial, Courier and Georgia) to improve the output result.

b. Create Box Files: The information about the Bounding Boxes for all the characters present in the training image is generated in the box file.

c. Run Tesseract for Training: For each training image and box file pairs, the tesseract-ocr engine runs in training mode to fix any error in the generated box file data.

d. Compute Character Set: For new languages, a character set file is required to specify the information like digits, uppercase, lowercase, etc.

e. Font Properties setting: The font family attributes should be specified in the training data.

f. Clustering: clusters the generated language characters shapes into prototypes.

g. Data Dictionary (Optional): The language dictionary helps the tesseract-ocr engine to decide the likelihood of different possible character combinations. Tesseract can use up to 8 dictionary files.

h. Define Ambiguity Rules (Unicharambigs): unicharambigs file allows removing the intrinsic ambiguity between two similar looking characters or their combinations by using a substitution rule. We designed several ambiguities rules from some initial experimentation on the password language. For example, the tesseract-ocr engine might recognize "n|" as "ri", which contradicts our password language word characteristics (i.e., any word must contain only one English letter). Table 2.6 shows the ambiguities rules we designed (✓=mandatory, ×=non-mandatory).

Table 2.6 Set of Password Language Ambiguities Rules

Ambiguities rules	Substitution
A|➔ 4	✓
r|➔ n	×
a|➔ d	×
L|➔ U	×
l ➔ 1	×
ri ➔ n	✓
q|➔ d	×

i. Generate the Language File: Final step is to combine all generated files into a single file with the extension (file.data). The password language data file is named as pwd.data. Figure 2.7 shows the typical training producers steps.

2.2.4.2 Tesseract-ocr output

Most mobile banking application login windows have a simple design and contain few words to extract as (e.g; sign in, remember my card, card number,..etc), Capture-Me relies on this simple design to identify the password word and card number, both length and position. Figure 2.8

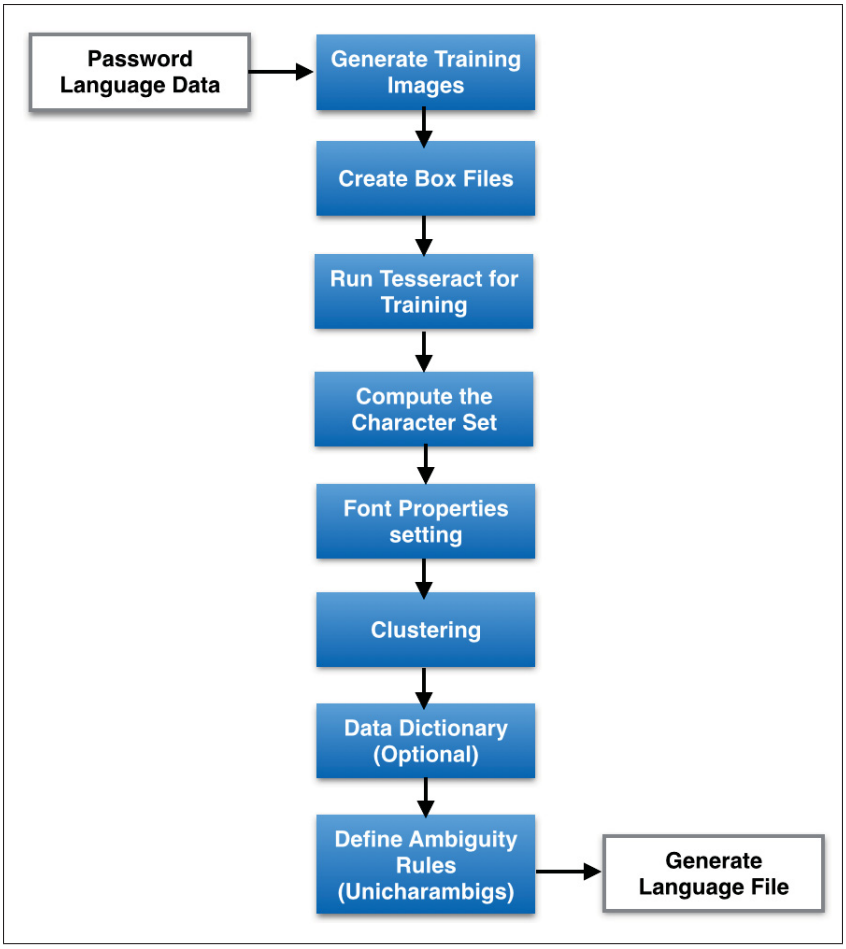


Figure 2.7 Training Data procedure Block Digram

shows a screenshot image from the Scotia bank mobile app and the output from the OCR analysis phase. For every output text extracted from a screenshot image, Capture-Me counts the password mask characters to identify the password length and matches the extracted letter with its position in the password word. The card number or user name label always exists on the top of card number text field (see figure 2.9), and in other apps existing in the card number text field at the initial state. Capture-Me follows the card number/user name label to identify the card number/user name.

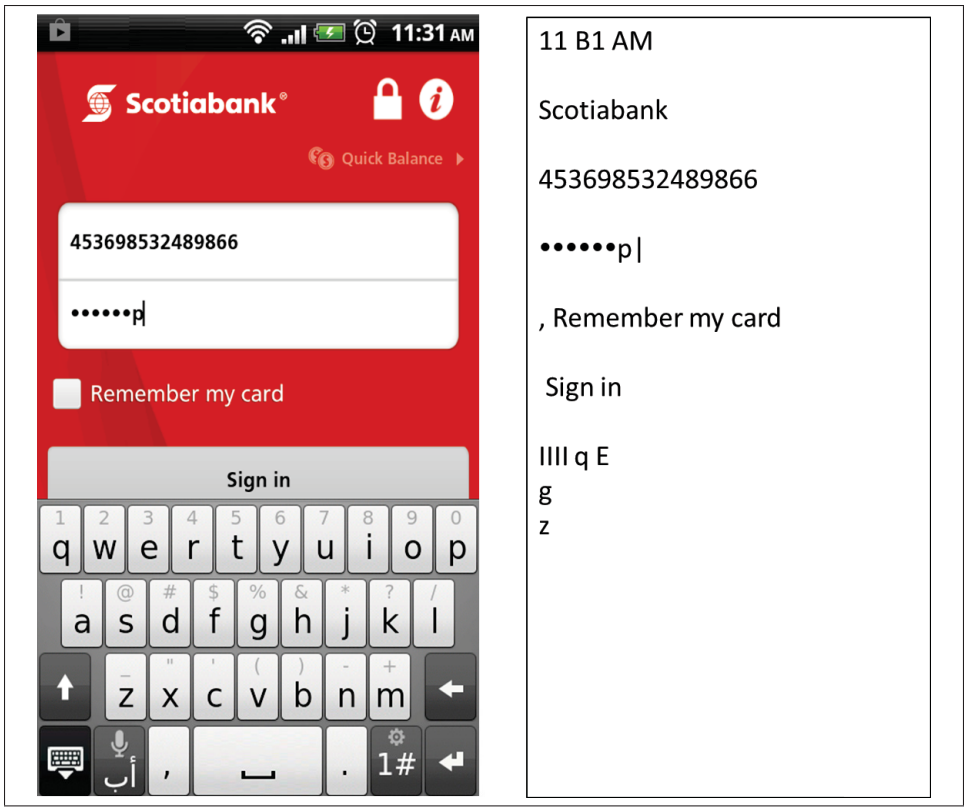


Figure 2.8 Output of OCR Analysis Phase

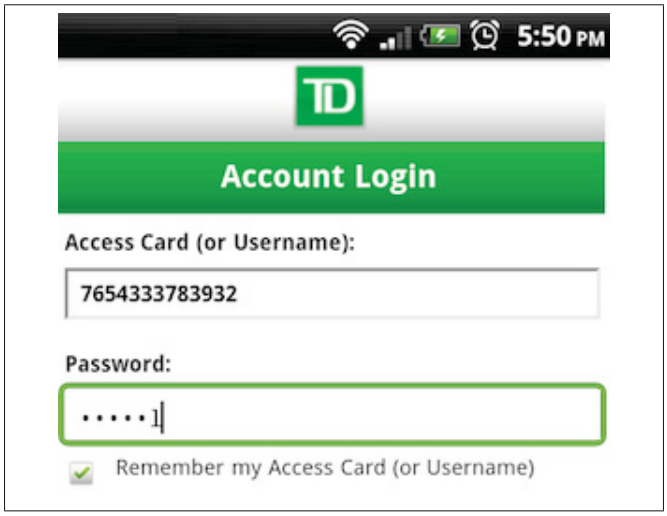


Figure 2.9 TB mobile banking app login window

2.2.5 Implementation optimizations

Finally to keep Capture-Me hidden in the victim device and improve the attack result, we perform the following optimizations: (a) We found that the screenshot images of two different sessions from the same user using the same mobile banking application are almost identical. We modified the implementation of Capture-Me to delay the screenshot function execution by 0.5 msec in the second session. This delay helped Capture-Me to take a different screenshot image in the second session. (b) We register Capture-Me with the system AlarmManager service(36) to re-start Capture-Me even if it is terminated by the Android system. (c) Capture-Me stores the sceenshot images without any file extension to be hidden from the media gallery apps. (d) Capture-Me performs the OCR analysis when the device screen is locked, to completely hide any performance degradation (if noticeable) by users. (e) Capture-Me maintains the device storage by deleting the screenshot images session directory after the OCR analysis is done.

2.3 Evaluation and Performance

The typing speed on smart devices is variant for different users (cf. (115), (75)). When the user typed an unfamiliar password in the chosen mobile banking applications, Capture-Me was able to extract a 10-character password from the first time the user entered her UserID-password pair. However, after applying the attack scenario many times with a familiar password typed by the user, Capture-Me requires at least two sessions to extract each pair of User ID and password.

2.3.1 Evaluation Test

We decided to challenge Capture-Me with a strong password to check the effectiveness of our attack scenario. We typed 30 different 10-character passwords more than fifty times to evaluate Capture-Me with real user typing speed. Each password contained five lower-case letters and an upper-case letter, three digits, and a special character, e.g; Alq012$uvn. We typed a UserID-password pair, in all mobile banking applications for three sessions in the experimental devices

(Nexus 10 and HTC Desire HD). Table 2.7 shows the average number of password characters that Capture-Me succeeded to extract from 30 different passwords in each mobile banking app within each session. Appendix I contains the results of the experimental test. The experimental

Table 2.7 Avg Number of password characters extracted on Nexus 10 and HTC Desire HD in three sessions

Nexus 10			
Bank Name	Session 1	Session 2	Session 3
CIBC Bank	7.2/10	9.5/10	10/10
BMO Bank	3.9/6	6/6	-
Natioanl Bank	4.9/8	8/8	-
RBC Bank	7.3/10	9.5/10	10/10
Scotia Bank	7.2/10	9.5/10	10/10
TD Canada Bank	7.3/10	9.5/10	10/10
PayPal	7.3/10	9.6/10	10/10
HTC Desire HD			
Bank Name	Session 1	Session 2	Session 3
CIBC Bank	6.8/10	9.3/10	10/10
BMO Bank	3.6/6	5.9/6	6/6
Natioanl Bank	4.6/8	7.9/8	8/8
RBC Bank	7.0/10	9.4/10	10/10
Scotia Bank	7.0/10	9.5/10	10/10
TD Canada Bank	6.9/10	9.4/10	10/10
PayPal	7.1/10	9.5/10	10/10

result shows that Capture-Me was able to extract the User-ID and at least six of the password characters from the first session. In the second session Capture-Me was successful extract most of the passwords. However, there are some passwords Capture-Me takes three sessions to successfully extract them. In fact, Capture-Me takes three sessions to extract passwords that have repeated letters such as P$ppppp234 or Hhhhh@123h as it is faster to type the same letter multiple times. However, for strong passwords it is recommended that the passwords do not contain letter repetition, we tried these passwords mainly to challenge Capture-Me . Capture-Me was able to extract more password characters in the Nexus 10 device rather than the HTC desire HD device. This is explained by the fact that the user typing speed on tablet devices is slower than the one on smart phones. According to our experimental test results, Capture-

Me shows a high success rate of password characters extraction. For example, if the user has a password of 12 characters, Capture-Me needs three sessions to fully extract the password. Figure 2.10 illustrates the ratio of password characters extraction per session.

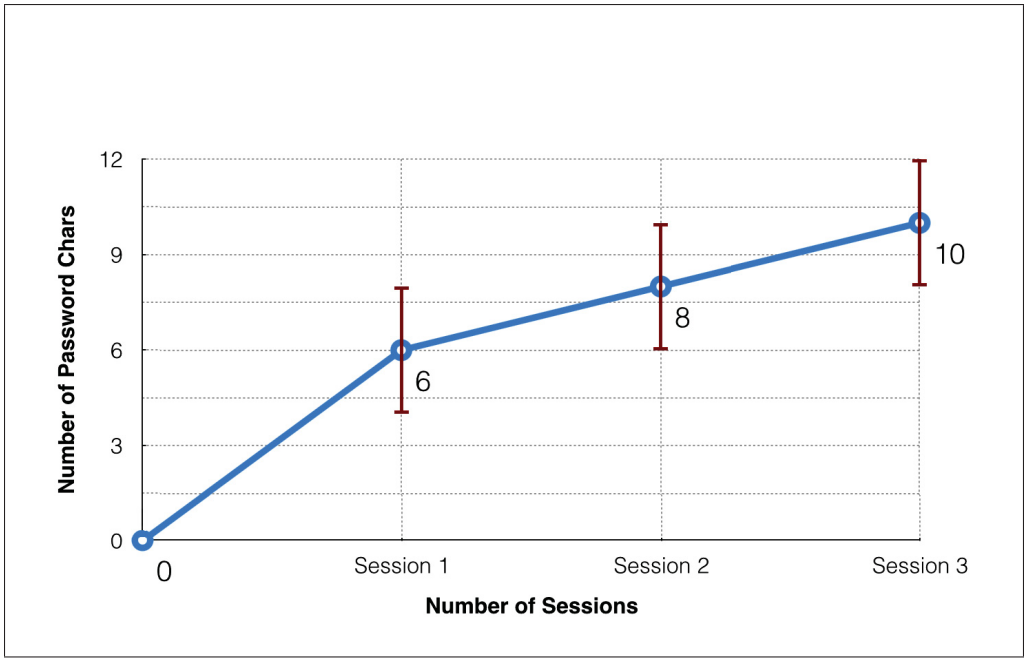


Figure 2.10 Password characters extraction Ratio

- **OCR analysis on bank websites:** We extend the Capture-Me evaluation to apply the attack scenario on the targeted banks websites. We used the Chrome browser on the Nexus 10 device to apply the attack scenario; Figure 2.11 shows the output of the OCR analysis phase on the TD website screenshot image. Capture-Me was able from the first session to extract the URL of the bank website, the User ID and 4 characters from the password. The rich content of the website was somewhat challenging for Capture-Me in terms of screenshot image size (compared to the relatively simple design of app widgets). In addition, the rich content of the website also challenged the performance of the tesseract-ocr engine in Android. Capture-Me faced the challenge of deciding the right time to take the screenshot image of the target website. For mobile apps Capture-Me relied on the application package name and sequence of operations to take the right screenshot image see section 2.2.3.

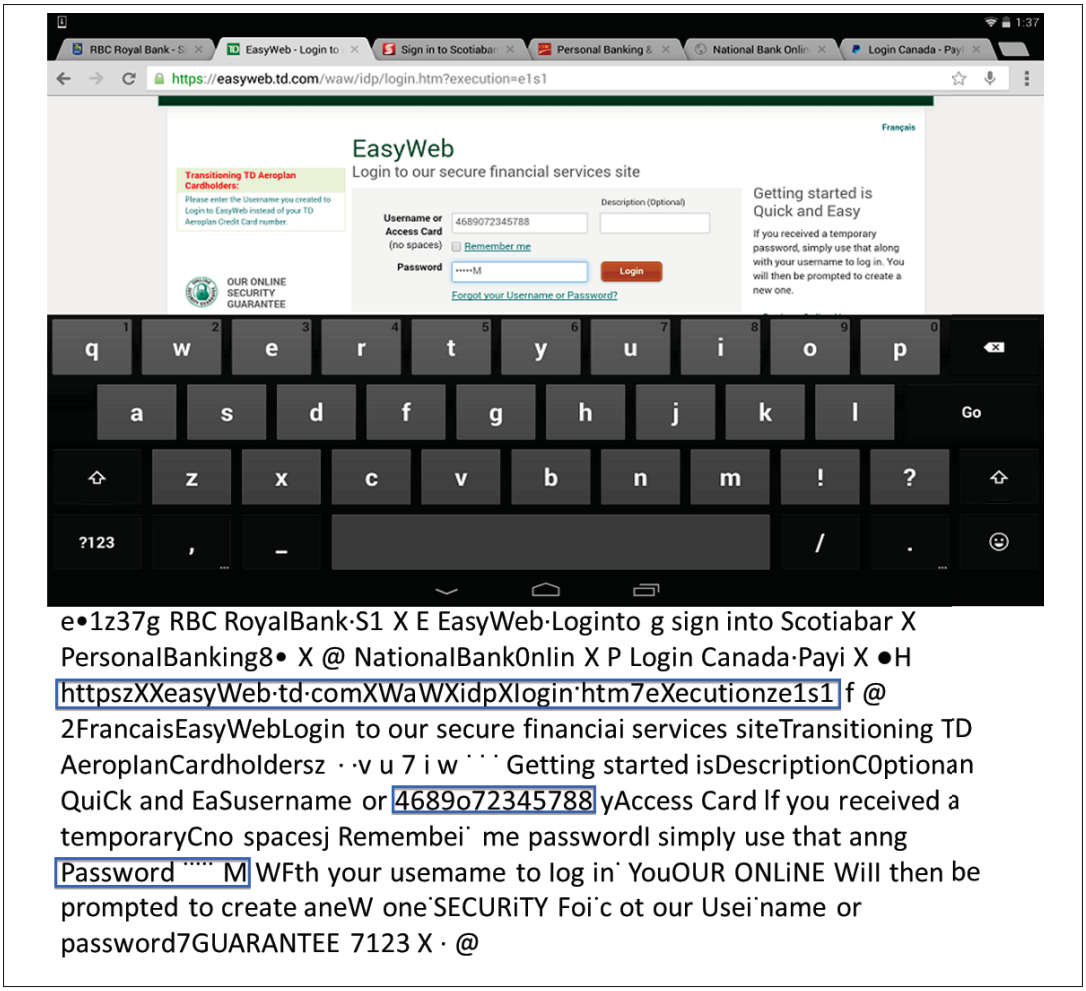


Figure 2.11 OCR analysis on TD bank's website

However, for the bank websites Capture-Me instantly took a screenshot image from the upper part of the screen which contains the website URL, then applies the OCR-Analysis at runtime. We decided to let Capture-Me take a screenshot image from the upper part of the screen to enhance the OCR-Analysis performance (as small images take less time for the tesseract-ocr engine to analyze). The output of the OCR-Analysis will be compared with target website URL to start taking the right screenshot image.

2.3.2 Performance

As Capture-Me goes through different phases to complete the attack scenario, we evaluate the Capture-Me performance in each phase individually. In the Capture-Me monitoring phase,

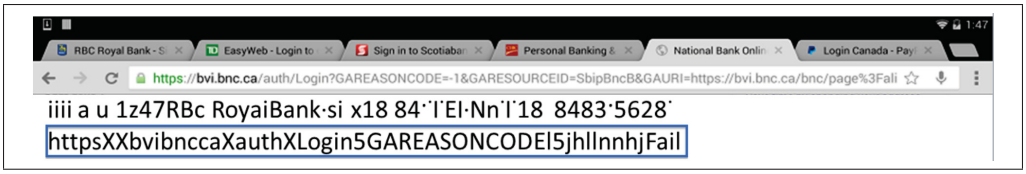


Figure 2.12 A screenshot image of the screen upper part and its OCR-Analysis output

the device is in screen-on state and the user runs any other application rather than the target mobile banking apps. In the Capture-Me capturing phase, the user uses one of the targeted mobile banking apps causing Capture-Me to actively take screenshot images. In the OCR-Analysis phase, the device is in screen-lock state and Capture-Me extracts the user credential from the taken screenshot images. We run the CPU monitor tool (70) in the nexus 10 device for 60 secs in each state with/without Capture-Me actively running. Figure 2.13 shows the average usage of the CPU in the Nexus 10 device with/without Capture-Me running during the different device states. Capture-Me shows small changes in CPU usage during the device's

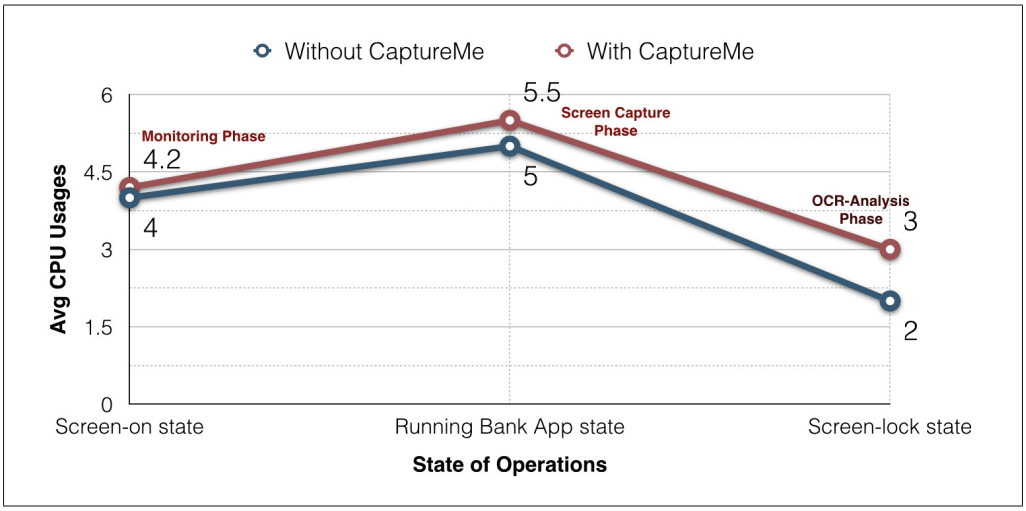


Figure 2.13 The CPU average usage

different states and in fact we did not notice any degradation of the device performance while Capture-Me was actively running.

2.4 Protection and Mitigation

Since Capture-Me uses different techniques to capture the screenshot images, we will explore the possible protection and mitigation mechanisms relative to each technique.

- **Protection against the Screencap command:** As we explained in the background section 2.1 Screencap command relies on the surface flinger service to take a screenshot image. The Android software development kit SDK provides the third party applications with a security flag ``flag_ secure´´(45) to protect the application window (the activity object) from being captured by the screencap command. When the application window object is initialized with the flag_ secure, the surface flinger service assigns a security flag to the frame buffer of the associated foreground application. Then when the Screencap function requests the frame buffer of the foreground application to perform the screenshot function, the surface flinger service replies with an empty frame buffer. We developed a tool to run more than 130 mobile banking applications in the Nexus 10 device. This tool executes the Screencap command as each application starts, in order to check if these apps are using the security flag. After running this tool, we discovered that only 10 mobile banking apps use the security flag. Table 2.8 shows these mobile banking applications. During our test of the

Table 2.8 Mobile Bank Application that use Flag_Secure

Mobile Banking Applications Names
U.S Bank Prepaid Campus Card
ABN AMRO Mobile Banking
Natioanl Bank App
First Niagara Mobile Bank
Meine Bank
Bank Millennium
Capitec Remote Bank
Easy banking BNP Paribas Fortis
ING Smart Bank
VR-Banking

Sceencap command with 130 mobile banking apps, we found two mobile banking apps,

Woori Global Bank (132) and HANA Bank (82) prevent the user from using the application if the smart device is rooted. This prevention limits the smart device user's ability to use the mobile banking apps, but does not protect the mobile banking application from screenshot attacks.

- **Protection against the fb0 method:** The frame buffer device (fb0) is only accessible to the processes that have the graphic group's privileges or have the super user's permission, see section2.1.1. On the Android platform the super user consists of two components: 1) the SU binary (119) which runs as a daemon service and broadcasts an intent message when any process requests the super user's permission; and 2) the Super User application (18) which works as a control manager to receive the intent messages from the SU daemon service. The Super User application will ask the user through prompt dialog to grant or deny the super user permission request. A possible security weakness in the super user's permission usage is that when a process grants the super user permission, there is no control over its further execution. We modified the super user components to limit the execution of the third party applications that grant the super user permission. When the user runs the mobile banking application, the proposed SU daemon service will deny any requests for the super user permission from any third party applications until the banking application process is terminated. Furthermore, any third party application that has granted the super user permission will be terminated when the banking application starts. Since the process that has granted the super user permission is able to modify the Android system files, the user should be notified, if any process tries to modify the system files. Moreover, we modified the super user application to run as a background service and communicate with the SU daemon service through authenticated TCP socket connection to check if the SU binary has been replaced or exploited. As Capture-Me relies on the super user's permission to gain access to the frame buffer device (dev/graphics/f0), our new proposed super user will prevent Capture-Me from granting the super user permission while the mobile banking application is running. Additionally, our proposed super user adds more fine grained access control to the super user's permission on the Android platform and could be extended to work with other mobile apps.

- **Multi Factor Authontication:** The Multi Factor Authentication MFA (124) is a security mechanism that requires more than one form of authentication to verify the legitimate user (74). In bank systems, the MFA is usually a combination of two authentication factors: first, the (User ID and password) and second, something owned by the user, e.g: a physical device or the phone number. Some financial institutions such as HSBC, support their clients with a physical secure key generator. When the mobile banking application asks the user to enter her second authentication factor instead of the user id and password, Capture-Me failed to extract the user id and password. We mentioned the MFA authentication technique because it limits Capture-Me trials to extract the user's credentials for 1 or 2 sessions, as the user will only enter her user id and password at the first time the mobile banking application runs. Table 2.9 shows the five financial institutions we found that use the MFA technique during our test of 130 mobile banking applications.

Table 2.9 Mobile Banking Application that use MFA mechanism

Mobile Banking Applications Names
HSBC Mobile Banking
BNY MELLON Private Banking
ABN AMRO Mobile Bank
Bank of Ireland Mobile Banking
First Security Bank Mobile

CHAPTER 3

SE-PERSONA : SECURE ENHANCEMENT CONTAINERS ON MOBILE PLATFORM

Many companies and organizations are trying to adapt Bring Your Own Device BYOD model (109) with their enterprise networks and systems to help their employees get the maximum benefits of the smart device's usage. The employees like to use their own smart devices to connect to their company's enterprise systems and do their work with greater flexibility and freedom. Other companies have security concerns so they let their employees use the company smart devices and prevent the employees from using their own smart devices. In both cases, companies need a policy management system and a data isolation policy to overcome employees' misuse of the smart devices with their enterprise systems (93). Mobile virtualization technology is one of the solutions that achieves data isolation and policy management in the two aforementioned models. In fact, mobile virtualiztion offers a flexibility with the smart devices usage and addresses the concerns over the personal data privacy, while delivering the security requirements of the enterprise systems. 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 smart device. The separation of the mobile platform VMs (personal and business) let the employee have full control of her/his private data and applications without affecting the enterprise system requirements of controlling their data security and policies management. Different virtualization technologies such as KVM (64), Xen (76) and LXC (65) are being adapted with mobile platforms considering multiple factors such as the hardware specification, the availability of software applications and the back end management systems. The limitation of the smart device's hardware compared with the desktop or server computer challenges virtualization technologies to be adapted to mobile platforms. Fortunately, the lightweight OS-level virtualization technologies such as LXC (65) or Cells (53) are well adapted to the Android platform due to the minimum hardware requirements and the kernel sharing mechanism between containers (virtual machines) which increases the system performance. In the OS-level virtualization, the virtual machine is named as container due to the kernel sharing mechanism which runs different operating systems of the same type simultaneously. Also in mobile virtualiza-

tion technology, the container could be named as Persona due to the different personal usage of different containers. The OS-level virtualization provides a virtual environment (for the Linux kernel-like operating systems) that has its own process and network communication, instead of creating a full-stack operating system as virtual machine. Furthermore, the kernel sharing mechanism incurs less CPU and memory usage and network overhead which is important due to the performance and hardware limitations in the smart device. However, a security weakness due to the shared kernel mechanism in the OS-level virtualization could lead to privilege escalation between containers. Typically any application/process that runs within a container and succeeds in obtaining the root level privilege will have access to all other containers with the same root level privilege. Cells Project (90) is a lightweight OS-level virtualization project which shows an outstanding implementation and performance with the Android platform. The virtualization architecture used by Cells allows multiples Android containers to run simultaneously on the same device where there is one foreground container and other containers running in the background. We investigated the security of Cells system and experimented a privilege escalation attack on the virtualization control components and we succeeded to escalate the root privilege from one container to another. More details about Cells and its security weaknesses will be explained in the security weaknesses section 3.3 as an example of OS-level virtualization on mobile platform. The virtualization technologies were established and used on desktop and server platforms which did not address the different behavior of the smart device. Due to the mobility of 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 business and personal mobile platform VM. The business mobile platform VM has an 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 mobile platform VM has access to the Geo-location information will be able to record the GPS coordinates while the accountant is doing her work. However, when the accountant agent is running the enterprise application in her business mobile platform VM, the Geo-location information becomes enterprise sensitive data. Meanwhile, the enterprise policy management system could not dis-

able the access to Geo-location information in the accountant personal mobile platform VM because of the employee's needs or the company's policy management system has no control over it¨. Another use case ´consider a company preventing its employees from using the smart device microphone or camera during the working hours and locations. The company policy management system applies these rules on the business mobile platform VM. However, in the personal mobile platform VM which is running in the same device, any third party application that has access to the microphone or camera can start recording audio or video track without notifying the business mobile platform VM. Meanwhile, the company policy management system cannot disable the microphone or camera in the personal mobile platform VM because it has no control over it¨. From the previous use cases, our definition of the privacy escalation in mobile virtualization is: In the smart device, a process which belongs to a VM is able to communicate or access data of a certain system resource while the active foreground VM restricts its own processes from accessing or communicating with the same system resource. Many research projects have been conducted to defeat the privilege and privacy escalation on the Android platform such as Apex(103), FlaskDroid(60) and SE-Android(120). The SE-Android becomes fully integrated in the Android Open Source Project AOSP(4) since Android version 4.3 Jellybean (23). SE-Android provides two levels of operating system policy integration in the Android software stack; 1) Secure Enhancement Linux(105) policy which allows a low level policy integration with the kernel devices and components; and 2) Middleware Mandatory Access Control MMAC policy (29) which allows a middle level policy integration with the Android framework and third party applications. The operating system policy integration in SE-Android enhances the Android platform security. However, SE-Android addresses the privilege and privacy escalation in a single mobile platform and needs more investigation to be adapted with the Android virtualizations technology.

3.1 Android Virtualization

In the 1960s the first virtualized computer was created by IBM (63). At that time, the main purpose of virtualization technology was to share the system resources between different applications/processes that run simultaneously. 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.

3.1.1 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 hardware interface 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 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 middle-ware layer and the control component is part of the middle ware layer such as,the dalivk virtual machine on Android platform(71). 2) Operating System level Virtualization refers to a mechanism by which the kernel layer is shared between containers, however the control component is not part of the container middle-ware framework, such as Linux Containers (65). 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 for example Xen Hypo-Visor (76) and KVM (64). Figure 3.1 shows the three types of virtualization technology from the operating system perspective.

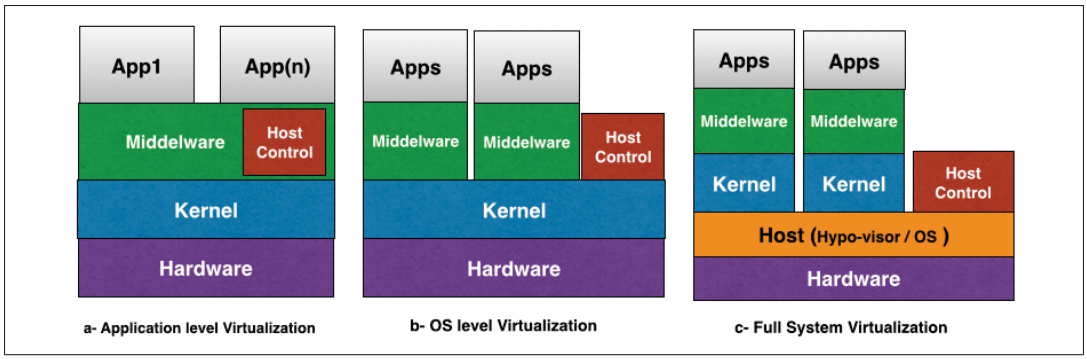


Figure 3.1 Different Types of Virtualization Technology

3.1.2 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 virualization with enterprise systems as following:

a. *Service and Application isolation* (14): In the smart device, enterprise applications and services are placed in a sandbox environment where the back end enterprise system is able to 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 weakness in the sandbox environment is that the enterprise system does not have the ability to apply a restricted policy on the smart device. This in turn, can lead to privacy and privilege escalation with the enterprise application's data.

b. *Two isolated containers* (113): two rigid separated environments (business and personal) run simultaneously in the same physical smart device in order 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 is able to 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. In fact from the users' perspective, her/his private data may be compromised by the enterprise system which will restrict the users' personal usage of the smart device.

c. *Multi-OS experiences* (106): the user has the ability to 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 a number of different OS with a constraint of same kernel compatibility. The user can create her/his business Android container and give the enterprise system full control over it. The enterprise system has the ability to 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 kernel such as Android, Ubunut touch OS (32), Firefox OS (26) and Tizen (22), this may be useful for the enterprise system to use its secure platform on the smart device and also useful to many experienced users.

3.1.3 Cells as OS-level Virtualization

Cells is an OS-level virtualization that shares a single kernel across all containers with virtualized identifiers, kernel interfaces, and hardware resources. Figure 3.1-b typically shows the cells' architecture. Cells uses different namespace's isolation mechanisms applied on the kernel devices with hardware resource multiplexing to provide process isolation with native performance. In more technical details, Cells uses the Mount namespace, Net namespace and UTS namespace (92). These namespace's isolation mechanisms required a device wrapper interface for the kernel drivers, as well as modification to the kernel device subsystems in order to manage the namespace ID. Cells assumes that the root namespace is part of the trusted com-

puting base 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. Celld process is also responsible for managing (start, stop, switch and destroy) the containers. Celld communicates with the containers by using the Inter Process Communication (IPC) mechanism. The base file system in Cells is shared as a read-only file system between containers. The file system unioning mechanism (133) is used to provide the container with a union view of its own read-write file system and the read-only base file system. This technique provides good scalability in creating and managing the containers. However, this mechanism increases the likelyhood of root privilege escalation from a container to another or to the root namesepace as the base file system is shared. We will show in section 3.3 the privilege escalation attack we applied on Cells. Cells relies on two techniques to manage memory usage and performance; 1) the Android low memory killer (8) which is used to increase the number of containers that are possible to run without sacrificing system functionality; and 2) the Linux Kernel Same page Merging KSM (72) which is used to find the anonymous memory pages that have the same content in order to arrange them as a one copy, shared among containers. The KSM technique improves the system memory usage. However, the author of (125) used a memory disclosure attack to expose the shared memory data from different VMs. The memory disclosure attack takes the advantage of the difference in write access times on the deduplicated memory pages that are re-created by the Copy-On-Write mechanism in KSM. We will show in section 3.4 the SE-Policy module Celld.te we created to enhance the security of the Celld component in Cells.

3.2 Security Requirements

The enterprise systems security requirements was showed in (73),(110). In the following, we will list the security requirements of the enterprise system regards to mobile virtualization technology.

a. **Isolation between Containers/VMs and Management module:** 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 capabilities offers more security to the local stored data. However, the virtualized environment management module has a possible security risk to be compromised from one of the running containers/VMs especially in the OS-level virtualization. Exposing the virtualized environment management module could lead to privilege escalation between containers/VMs. We will show in section 3.3 by attacking the virtualized environment management module in Cells, we were able to control the system.

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

c. **Secure Remote Management:** The user needs to manage her/his virtualized environment in the smart device (create, start, stop and destroy containers/VMs). 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 exist in one of the running containers/VMs. The container that hosts the virtualized environment management module works as a master container and it is part of the trusted computing based TCB architecture and cannot be destroyed, 2) Remote management where the virtualized environment management module can be accessible by an authenticated network connection or through connected desktop PC (USB/Serial connection). The virtualized environment management module is part of the trusted computing based TCB architecture and it starts early in the system's components boot up sequence. Many virtualization technologies support libvirt (83) virtualization API for remote man-

agement functionality. Both techniques should be implemented in mobile virtualization technologies according to usability and security requirements. Possible security risks at compromising the master container for local management and at connecting the smart device to a desktop PC for remote management.

3.3 Attacks Models

The virtualization technologies on the server platform rely on the isolation between containers/virtual machines to achieve data isolation, and use other security management systems to protect the server platform such as firewall and anti-virus. Users or applications gain access to the server platform after going through different authentication mechanisms and system policies to protect the server platform from malicious executions and misuse. However, the smart device usage is different. The smart device user has the ability to easily install/uninstall applications which lead to misuse her/his private data. The software architecture of Cells (OS-level virtualization) shows security weaknesses in different levels of the system software-stack. As for example; if a process that belongs to a running container succeeds to obtain the root namespace's privilege, it will be able to control other containers. Also Cells does not provide access control policies or secure remote management to the virtualized environment. We will investigate the possible security weaknesses in Cells by applying three attacks models; privilege escalation, privacy escalation and remote management attack.

3.3.1 Privacy Escalation attack

As the running foreground container is unaware of the background running containers in the same smart device, the privacy escalation attack model is defined as follow: A process P which belongs to a compromised container C´running in background, will be able to communicate or access data of a system resource R which is under restricted access policy in the foreground container C´´where C´and C´´belong to Cs; the set of containers in the smart device. The attack environment consists of two containers C1 and C2. We assume that accessing the GPS coordinate data and audio recorder functionality by the third party applications are restricted in C1. We installed a GPS tracker application and audio recorder application from Google Play

store in C2. The attack model starts by placing the container C2 in the foreground and then the user starts the GPS tracker app and the audio recorder app. The user starts the container C1 and switches it to the foreground then uses the smart device as she/he will move around and talk to people. After the user ends the work with the container C1, she/he switchs back the container C2 to the foreground. The GPS tracker app was able to access the GPS coordinate and the audio recorder app was able to record an audio track while she/he was actively using the container C1. We propose new properties to the MMAC policy to apply more fine grained access control on Android platform system permission to overcome the privacy escalation. We will explain in more details the MMAC policy in section 3.4.1.

3.3.2 Privilege Escalation attack

According to Cells architecture (see section 3.1.3), the privilege escalation attack model is used to obtain the root namespace privilege defined as follow: A running process P belonging to a container C is able to obtain the root namespace privilege which is not originally granted to it. We apply this attack model on Cells open source project (90) built on top of Android 4.3 Jelly Beans platform (23). Since Android 4.3 platform released the secure enhancement Linux kernel module (77) was implemented on Android platform. However, when we checked the SE-Linux module status in Cells, we found that the SE-Linux is disabled. As Cells provides a modified Kernel configuration (12) which disables the security option of the SE-Linux module in the kernel, we modified the Kernel configuration to enable the SE-Linux security option. Also, we integrated the system with the SE-Policy of Android 4.4 KitKat platform (37) which applies the enforcing mode in the SE-policy and has more stability. As we explained in section 3.1.3 celld process is the control component of the virtualized containers, for this reason we tried to obtain access to the celld process to be able to control other containers. In the root namespace, the communication with the celld process is performed by using the command-line cell. By using the command-line cell the legitimate user is able to create, destroy, start, stop, and configure containers. We initialized the attack environment by starting three different containers C1, C2 and C3. We develop an application to check the ability of executing the command-line cell from one of the running containers. The application is installed on C1 and by executing

the application, we where able to create a new container C4 and stop, start, destroy other containers. Furthermore, as celld has the ability to execute a shell command in the running containers, we were able to obtain a shell access to other containers from C1. We explained the success of the privilege escalation attack we applied, by the fact that the Cells' components were not defined nor implemented in the SE-Policy modules. We will show in section 3.4 our proposed SE-Policy module for celld process which is mandatory to defeat the privilege escalation from the running containers to the root namespace.

3.3.3 Remote management Attack

In Cells, the legitimate user should connect the smart device to a desktop PC through USB connection and use the ADB (Android Debug Bridge) (3) command to gain access to the root namespace shell. After the user gains access to the root namespace shell, she/he will be able to manage the virtualized environment (Containers/Persona) by using the cell command. The attack we applied takes advantage of connecting the smart device to a desktop PC to manage the virtualized environment and injects a malware to one of the running containers. In more technical details, Cells stores the containers directories under the data partition in the smart device. Each container has its read-only file system which is the base file system (see section 3.1.3) and also has its own read-write file system which exists under the /data/cells/ directory in the data partition. As the ADB connects to Cells through the root namespace, the ADB gains the root namespace privilege over the data partition which gives it the ability to copy a file from the connected desktop PC to the smart device's data partition then executes it. We designed a simple malware to run as a native service and read the content of any text file in the container external storage then send the file content to the attacker server. The attack scenario is defined as follow: 1) The smart device user connect the smart device to his desktop PC to manage the running persona/container or to transfer some files. 2) The user's Desktop PC was infected by captivating application which will copy and execute the malware (native service) to the smart device once the user connect the smart device to the desktop PC. 3) The native service will scan the container's external storage and read the text files' content than send the file's content to the attacker server. We suggest an ADB proxy component to the Cells architecture

to defeat the remote management attack. The smart device's user should not be able to copy and execute files by using the ADB commands under the root namespace privilege. However, the smart device's user should be able to use the ADB commands with the running containers as a normal Android platform. In some cases, the user may need to use the ADB connection for application development activity. More details will be explain in section 3.4.3.

3.4 Mitigating OS-level Virtualization by SE-Android

Security Enhanced Linux (SE-Linux)(105) was originally developed as a Mandatory Access Control (MAC) (78) mechanism for Linux OS. Android platform built on the top of Linux kernel and thus the SE-Linux was capable to be implemented with Android platform. The author of SE-Android explained the required changes of the Android security architecture and components to integrate the MAC mechanism with the Android platform (120). In this section, we will explain our contribution based on the SE-Policy and Android platform to mitigate the attack scenarios we applied on Cells. We proposed a new SE-Policy module celld.te which secure the celld process against the privilege escalation attack. Also we proposed new properties to the MMAC policy to overcome the privacy escalation between containers. Finally, we proposed a new ADB proxy component to mitigate the possible security threat against the root namespace.

3.4.1 MMAC Policy on Android Virtualization

In Android platform, applications should be digitally signed by a 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 middle-ware framework security architecture. The MMAC policy controls the application's access (grant/deny the uses-permission) to the system resources even after the system resource was granted to the application during the installation process. The MMAC policy categorizes the installed applications depending on the application signing certificates. In Android open source project AOSP (4),

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 (112) and they can control the third party applications' uses-permission. The MMAC policy source file is mac_permission.xml which is written in XML format with predefined tags (80) (28). Table 3.1 describes the available XML tags in the MMAC policy.

Table 3.1 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 [58]
<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.

3.4.1.1 Context Awareness

The applications categorization in the MMAC policy provides flexibility to manage the system *uses-permissions* with different applications instead of defining a 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 and locations. 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 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 white list (allow uses-permission) to the black list (deny uses-permission) until she/he reload the MMAC Policy (mac-permission.xml) into the Android platform. Moreover, in the virtualized environment, the running foreground container is unaware of the running background containers' MMAC policies which could easily lead to data privacy escalation between containers. To enable the context awarenesses access control decision, we propose new properties *Exclusive-use*, *Exclusive-deny*, *Trusted-app* and *Security-level* to the MMAC policy;

a. *Exclusive-use*: The application has declared the required uses-permission to gain access to a certain system resource. Meanwhile, at the application's execution time, the application should be the only process that has access to the required system resource.

b. *Exclusive-deny*: At the application's execution time, the application might needs to restrict the accessibility of a certain system resource due to security concern. However, the application is not required access to the same system resource to work properly.

c. *Trusted-app*: The MMAC policy should specify which application or applications' category will has the ability to apply the *Exclusive-use* and *Exclusive-deny* policy. Otherwise any third party application can interrupt the system resources.

d. *Security-level*: As the MMAC policy could have multiple categories or applications with *Trusted-app* property, the *Security-level* prioritize the different trusted-app categories and applications. Otherwise the MMAC policy rules might cause conflicts in the AC decision.

e. *Restricted-Permission*: According to the different business usage of the enterprise systems, the enterprise system should has the ability to restrict the access to a certain system resource such as microphone or camera during certain times or locations. The enterprise system will be able to specify the times by an interval time period and will be able to specify the geographic locations by a GPS coordinates and buffer length. *Restricted-Permission* will be applied on all running containers (background and foreground) on the smart device and it has the highest security level.

f. *Restricted-Applications*: In Android platform AOSP the Play Store and GMAIL applications runs as system applications. To restrict the company's employees from use the Play store application and install third party applications in the business container. The Play store application package name will be defined under the *Restricted-Applications* tag to restrict the employee from use the play store inside the business container.

The business workflow and security concerns of the mobile enterprise application change due to different reasons. These changes could require also changes to the MMAC policy which will overwhelm the smart device's user as she/he will need to flush the smart device with a new Android ROM (24). Since both usability and security are concerns to the smart device's user and enterprise systems, the *Exclusive-use* and *Exclusive-deny* policy's properties will be defined by the mobile application in its package file. The mobile enterprise application should declare in its Manifest file the *Exclusive-use* permissions and the *Exclusive-deny* permissions and the *Trusted-app* property should be declared under the enterprise application's category in the MMAC policy. By this technique we met the mobile enterprise application security requirements with the usability concern of the smart device's user. As the MMAC policy and the Manifest file are written in XML formate, we defined the new policy's proprieties as XML Tags. Tables 3.2 shows the new policy's properties tags and description.

Table 3.2 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 depend on the policy creator prioritization and it should defined after the Trusted-app tag. e.g: <Trusted-app value="1" ><Security-level value="1" >
<*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>

3.4.1.2 The MMAC policy Rules Conflicts

In the Android virtualization environments such as Cells, Containers are sharing the same MMAC policy. However, the running foreground container is unaware of other running containers' applications and processes. The policy AC decision conflict will happen in two use-cases; a) When two applications running in the same container are declared in the MMAC pol-

icy 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 security levels. b) When two applications are declared in the MMAC policy under two different categories 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 rely on the foreground container to take the policy AC decision. 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 as the root namespace is the only module aware of all running containers' application and processes. The policy AC decision inside the container formulated as following:

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

The 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 $\in$ Ta where Ta is a set of the *Trusted-app* in the MMAC policy. Ta $\in$ Sl where Sl is a set of the highest *Security-level* category 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

The 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 $\in$ Ta where Ta is a set of the *Trusted-app* in the MMAC policy. A $\in$ Fc where Fc is a set of the running applications in the foreground container. Ta $\in$ Sl where Sl is a set of the highest *Security-level* category in the foreground container's MMAC policy.

3.4.2 SE-Policy in Android

The SE-Policy (30) contains the definitions of the security object classes and their associated permissions and rules that are used by the Linux Security Modules (LSM)(134) to apply the MAC in Android security architecture. The objective of SE-Policy is to classify the system

resources such as files, sockets, etc and define their related permissions that represent accesses to those resources such as reading or sending operations (81). SE-Policy consist of source files (file.te) which are used to generate the kernel binary policy file. In the following we will give a brief description of the major SE-Policy source files on Android platform.

a. *security_classes* (49): The security classes source file contains the definition of the system resource objects such as file, socket, binder, etc.

b. *access_vectors* (34): The access vectors source file contains the definition of the system resources permissions. For example the access vector of the file object as a security class is (ioctl, read, write, create, etc).

c. *attributes* (41): The attributes source file contains the declaration of policy object attributes and types. We can consider attributes as a property of policy types, usually used as a group of types e.g; domain, exec_type, port_type, etc.

d. *domain.te* (42): The domain source file contains the declarations of rules assigned to all domains e.g. allow domain fs:file { read getattr }.

e. *file.te* (43): The file.te source file contains the definition of all android system types e.g. file_type, bluetooth_type, adb_socket, etc.

f. *file_context* (44): The file context source file contains directories and files labels configuration which used by the Android platform to set the read, write and execute permissions of the files and directories.

g. *property_contexts* (47): The property context source file contains the configurations of the Android platform services properties which used to specify the security context of Android daemon service during the permission checking operation.

h. *seapp_contexts* (48): The seapp context source file contains the labels configurations that used to label different applications processes and package directories.

i. *mls* (46): The mls source file contains the declaration of the multi level security constrains that used to isolate application processes and files such as mlsconstrain process { transition dyntransition } ((h1 eq h2 and l1 eq l2) or t1 == mlstrustedsubject).

The SE-Policy is designed to be extended and to cover the security requirements of different Linux like operating systems. The procedures of extending the Android platform SE-Policy are explained at (20) and (19). To achieve a secure OS-level virtualization on Android Platform, we must first define a set of subjects and objects which are essential for the system to work properly. We followed the SE-Policy customization procedure to create the celld policy module.

3.4.2.1 Celld policy module

We added the celld.te source file to the SE-Policy modules which contain the definition of the Celld security object class and its associated type, rules and permissions. We define the celld type as domain type and we assigned to it the allow rules to control the cgroups and mount operation. Also we define the celld process as a security class in security_class source file and define the label configurations of celld and cell binaries in the file_context.te source file. The Celld process should have the privilege to mount the system partitions, starts the init process and control cgroups in the kernel systems. Figure3.2 shows a part of the celld.te policy source file.

3.4.3 ADB Proxy for Mobile virtualization technology

In this section we propose an ADB proxy component to virtualize the ADB component on Android platform and restrict the ADB access to the root namespace in Cells. The ADB in both sides the Desktop PC and the Android smart device acts as a transparent transport mechanism (136). Its two most important components are 1) the adb server which is running on the host (Desktop PC) and 2) the adbd daemon which is running on the target (Android smart device). These two components effectively implement a proxy protocol on which all adb services are implemented and they linked together either through USB or TCP/IP connection (3). We mod-

```
# 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 };
```

Figure 3.2 Celld SE-Policy Module

ified the adbd daemon (the target device) to virtualize and restrict the adb command executed 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 (10) 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. The smart device's user is able to gain access to the root namespace by the "adb shell -root" command and all other adb commands are restricted on the root namespace.

3.5 Architecture and Design

In Android 4.3 JB version, there are about 50-70 system services which give the mobile applications capabilities to communicate with the system resources through callable interfaces (25). The Android platform provides Binder IPC mechanism (7) 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 (5). We classify the system resources into two types; 1) The simulated resources by which the resource could be simulate its state and data. For example, the location manager service that represents the geographic coordinates could be simulated to gives simulated coordinate (x=0,y=0). and 2) The unsimulated resources such as the bluetooth and network. Based on this classification we control the running third party applications and processes access to the system resources during the MMAC policy enforcement. The idea behind this classification is to prevent the application's segmentation fault or crashing 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 denied access signal or will be terminated if they were actively running before the MMAC policy enforcement decision. We implement the SE-Persona prototype based on the Cells open source project (90) which is built on top of the Android 4.3 JB version. 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. As Cells uses the same Android framework version between containers and use the unioning file system mechanism (see background section) we exclude the mac_permission.xml from the read-only file system (base FS) and include it in the read-write file system in order to let each container has its own MMAC policy. Initially every new container will have the default MMAC policy as it exist in the AOSP (28) then the container will be able to reload its own MMAC policy. We modified different components in the Android framework to meet the new requirements of the MMAC policy. We proposed and implement new component to the Android framework which is the Permission Controller Manager PCM and we developed its virtualized component to Cells architecture which is the Root Permission Controller Man-

ager RootPCM. PCM and RootPCM will be responsible for enforcing the new MMAC policy requirements. In the following we will explain the modified Android framework component, PCM and RPCM. Figure 3.3 shows the new security architecture we propose for the Android platform and Cells.

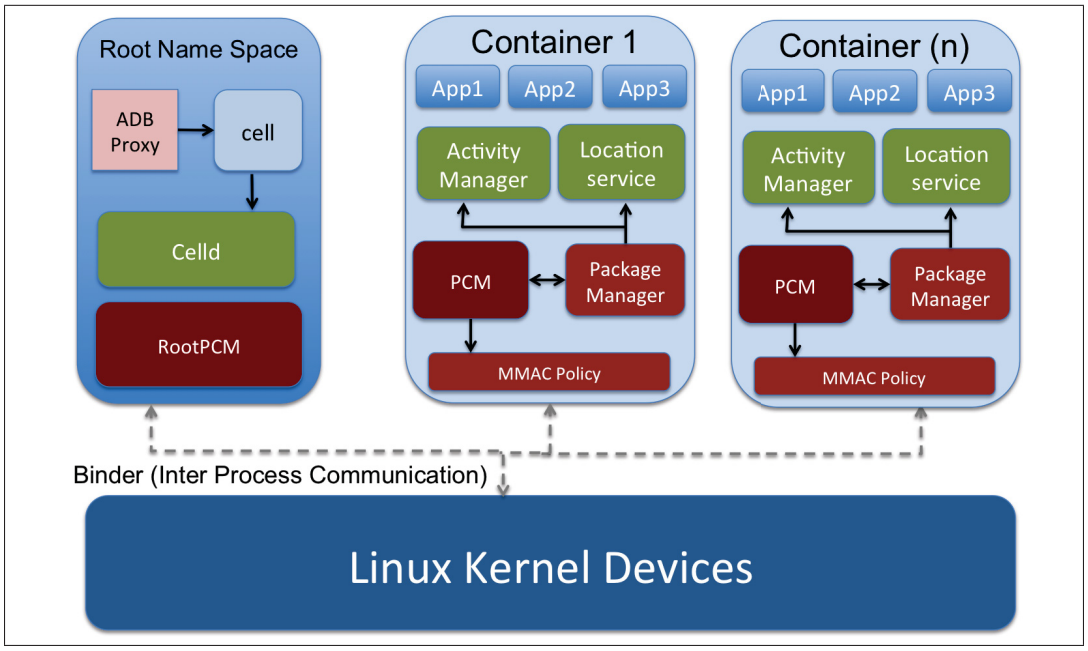


Figure 3.3 Architecture of SE-Persona security model

3.5.1 PackageParser

(107) 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 these information..

3.5.2 PackageManager

(38) The package manager service is responsible for installing and uninstalling applications on the Android platform. Also it manages the installed applications meta informations 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.

3.5.3 SELinuxMMAC

(21) 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.

3.5.4 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 (35) 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 microphone) the PCM will send intent to the package-manager to deny its uses-permission if requested during the trusted application execution time. In our prototype, we only make modification to the location-manager service to simulate its data. Also the PCM has a global setting for the Geo-location and day time in order to apply the high level restricted policy such as *Restricted-Permission* and *Restricted-*

Application. Figure 3.4 shows the Architecture of the added components in android security model.

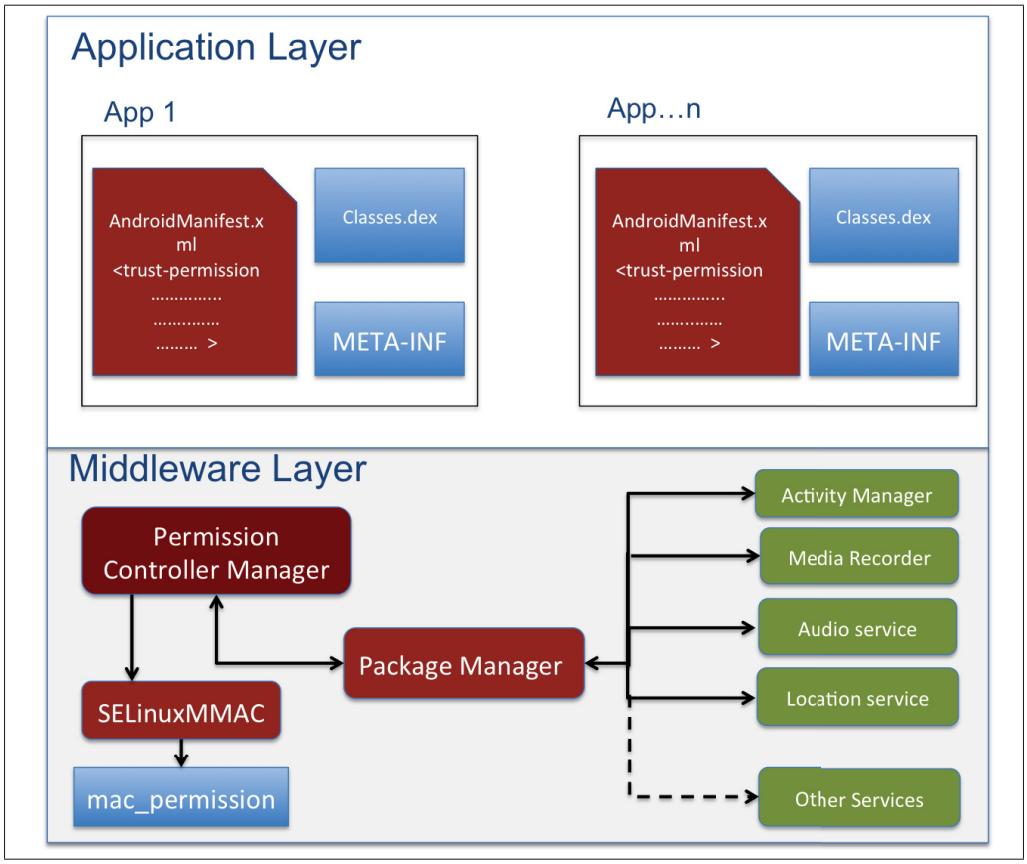


Figure 3.4 Architecture of the Security Model Inside Single SE-Persona

3.5.5 RootPermissionControllerManager RootPCM

The RootPCM 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 in order 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 (116) 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 3.3 shows the new security architecture we propose for the Android platform and Cells.

3.6 Evaluation

In this section, we present measurements regards to the SE-Persona platform performance and we examine the SE-Persona security against privilege escalation and private data exposer.

3.6.1 Performance

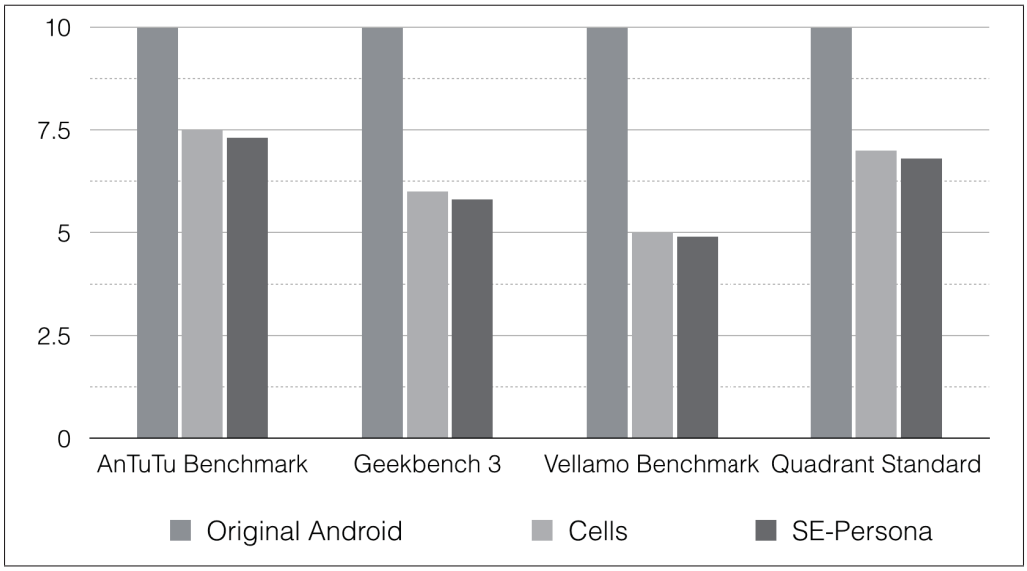
We evaluate the performance of SE-Persona prototype by comparing the original Cells platform performance with SE-Persona prototype performance using four benchmarks AnTuTu Benchmark, Geekbench 3, Vellamo Mobile Benchmark and Quadrant Standard Edition. The benchmarks environment consist of three running 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 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 3.5 shows the result of the benchmarks that ran in both SE-Persona prototype and original Cells.

3.6.2 Evaluation

SE-Persona implementation is based on Cells which use namespace isolation mechanisms and cgroup (control group) to provide isolation and secure environment. In order to evaluate the SE-Persona platform security we applied different attacks in order to gain higher privilege and expose privacy.

privilege escalation: As we mentioned in the Adversary model sec3.3, the control-host (celld process) is the attacker target in order to control the virtualization environment. Based on the

Figure 3.5 Benchmarks score result of three running containers



root user privilege (91) We were able to execute the cell command from one of the running container which let us create, start, stop and destroy containers from one of the running container. We explained the attack success by the fact that Cells' components (celld process and cell command) were not defined nor implemented in the SE-Policy modules. SE-Persona has defined and implement the SE-Policy module for the celld process (see section3.4.2) in order to prevent the privilege escalation to the control-host. We applied the root user privilege attack on SE-Persona to expose the celld process. However, the attack failed and the celld process only accessible through the root namespace.

privacy escalation: By integrating the context awarenesses MMAC policy in SE-Persona, the third party apps were not able to access the system resources such as location manager or audio service while 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. As for example, it was shown in (135)(61) using the accelerometer sensor could lead to expose the user typing data (could be user-name/password). SE-Persona can define 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. The *Restricted-Permission* added more usability and scalability to the

Android platform *uses-permission*. The user able to grant/deny a certain permissions 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. As for example, in Android platform AOSP the Gmail, Play Store and YouTube applications runs as system applications. While the smart device's user create a new container for her/his kids for child entertainment activity. The kids could miss use the Play Store application 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.

Limitation: SE-Persona implementation goal is make 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 device context. However, this technique limits the context awareness policy to be extendable nor self-defined. As 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 classify the applications depend on its certificate only which is mismatch with any other classification such as the applications' categories in Google Play store. Overall SE-Persona context awareness policy can satisfy the enterprise systems usage. However, it needs massive changes on the OS-level virtualization architecture and Android platform to be more flexible and extendable.

CONCLUSION

In this work we studied the smart device usage with the enterprise solutions. We proposed a security driven taxonomy for enterprise mobile applications, then we defined the security requirements that required by the enterprise solution to manage the security risks of the smart devices. We surveyed the current proposed solutions ideas from academia and presented KNOX as an industry solution from Samsung. Also we investigated the security risks of the password visibility feature on the Android platform. We presented Capture-Me as a new screenshot attack to expose the user credentials from the mobile banking applications. We evaluated Capture-Me in a practical attack scenario with six mobile banking applications and PayPal's mobile application. We explored the possible protection mechanisms to defeat the screenshot attacks on the Android platform with more than 130 mobile banking applications. We discovered that most of the mobile banking applications we examined do not use any protection mechanism to defeat the screenshot attacks. The experimental results of the Capture-Me attacks show the weakness of the (user id and password) as the only authentication mechanism used in many mobile banking applications. Our recommendation is that the mobile banking application should implement other authentication techniques such as the multi factor authentication in order to protect their users' data and their system's integrity. Finally, we investigate the security weakness of the virualization technology with smart devices (phones and tables), we were able to apply different attack models on Cells as an example of the OS-level virtualization architecture. We proposed new properties to the MMAC policy with a modification to the Android security architecture to apply more fine grained access control on the Android system resources. Our recommendation is that smart devices platforms need to adapt new policies with paying attention to the user privacy and the mobility of the smart devices. Our future works will focus on privacy exposure attacks on smart and wearable devices. The smart device's mobility makes the smart device screen more vulnerable to different kinds of privacy exposure attacks, especially since the Android wearable SDK (79) became available to third party application developers. There are many wearable devices such as the Samsung Gear watch (114) and Google glass (27) that will be available for normal users, these devices have a good camera that can be used to expose the user's private data from the smart device screen as the ``iSpy´´ (108) study showed.

In the wearable device era, The UX (user experiences) of the smart devices will need new secure authentication techniques to protect the user's credentials and private data.

APPENDIX I

Appendix I, shows the result tables of the Capture-Me attack experimental test. Each table present trials of 30 passwords with the seven mobile banking applications that we considered in the attack scenario.

Table-A I-1 Experimental results of the the Nexus 10 at session 1

Nexus 10 session 1							
Passwords	CIBC Bank	BMO Bank	Natioanl Bank	RBC Bank	Scotia Bank	TD canada Bank	PayPal
Alq012$suv	7	4	5	7	7	7	7
zSd@347ege	8	4	6	8	8	7	8
123Pa$sword	7	4	5	7	7	7	7
Ghau789@fb	7	5	5	8	8	8	7
het239$quI	8	4	6	7	7	7	8
123@Myname	7	4	6	8	8	8	8
Canda$987p	8	4	5	7	7	8	7
loL$123lol	7	3	5	7	7	7	7
Home@123jk	8	5	6	8	8	8	8
Capme@123k	8	4	5	8	7	8	8
Test$890me	6	4	5	7	7	7	8
suv345@Qsd	8	4	5	7	7	8	7
where789@d	7	3	4	8	7	7	7
my478@Test	8	5	6	8	8	8	8
Ets@123cap	7	4	5	8	7	7	7
Mount@345s	7	4	5	7	8	7	8
234$testMe	8	5	5	8	8	8	8
wHen@u234s	8	4	4	7	7	8	7
some$123He	8	4	4	8	7	7	7
How$ukn235	7	5	5	8	7	8	7
Pets$567we	6	4	5	7	7	7	7
Tttt@789un	7	3	4	7	7	6	7
Gggggg@234	7	3	4	7	7	6	7
Mmmmmm@111	6	3	4	6	6	6	6
Jjjjjj$999	6	3	4	6	6	6	6
tesest$123	7	4	5	7	7	8	7
$Pppppp234	7	3	4	6	7	7	7
Hhhhh@123h	7	3	4	6	8	7	7
Whyu@me789	8	4	5	8	8	8	8
how@Cap456	8	5	6	8	7	8	8
Average	7.2	3.9	4.9	7.3	7.2	7.3	7.3

Table-A I-2 Experimental results of the the Nexus 10 at session 2

Nexus 10 session 2							
Passwords	CIBC Bank	BMO Bank	Natioanl Bank	RBC Bank	Scotia Bank	TD canada Bank	PayPal
Alq012$suv	10	6	8	10	10	10	10
zSd@347ege	10	6	8	10	10	10	10
123Pa$sword	9	6	8	9	9	9	10
Ghau789@fb	10	6	8	10	10	10	10
het239$quI	10	6	8	10	10	10	10
123@Myname	10	6	8	10	10	10	10
Canda$987p	10	6	8	10	10	10	10
loL$123lol	9	6	8	9	8	9	9
Home@123jk	9	6	8	10	10	10	10
Capme@123k	10	6	8	10	10	10	10
Test$890me	10	6	8	10	10	9	10
suv345@Qsd	10	6	8	10	10	10	10
where789@d	10	6	8	10	10	10	10
my478@Test	10	6	8	10	10	10	10
Ets@123cap	10	6	8	10	10	10	10
Mount@345s	10	6	8	10	10	10	10
234$testMe	10	6	8	10	10	10	10
wHen@u234s	10	6	8	10	10	10	10
some$123He	9	6	8	9	9	9	9
How$ukn235	10	6	8	10	10	10	10
Pets$567we	10	6	8	10	10	10	10
Tttt@789un	9	6	8	10	9	9	10
Gggggg@234	9	6	8	8	9	9	8
Mmmmmm@111	8	6	8	8	8	9	9
Jjjjjj$999	9	6	8	8	9	8	8
tesest$123	9	6	8	9	9	9	9
$Pppppp234	8	6	8	8	9	8	9
Hhhhh@123h	8	6	8	9	8	9	9
Whyu@me789	10	6	8	10	10	10	10
how@Cap456	10	6	8	10	10	10	10
Average	9.5	6	8	9.5	9.5	9.5	9.6

Table-A I-3 Experimental results of the the Nexus 10 at session 3

Nexus 10 session 3							
Passwords	CIBC Bank	BMO Bank	Natioanl Bank	RBC Bank	Scotia Bank	TD canada Bank	PayPal
Alq012$suv	10	6	8	10	10	10	10
zSd@347ege	10	6	8	10	10	10	10
123Pa$sword	10	6	8	10	10	10	10
Ghau789@fb	10	6	8	10	10	10	10
het239$quI	10	6	8	10	10	10	10
123@Myname	10	6	8	10	10	10	10
Canda$987p	10	6	8	10	10	10	10
loL$123lol	10	6	8	10	10	10	10
Home@123jk	10	6	8	10	10	10	10
Capme@123k	10	6	8	10	10	10	10
Test$890me	10	6	8	10	10	10	10
suv345@Qsd	10	6	8	10	10	10	10
where789@d	10	6	8	10	10	10	10
my478@Test	10	6	8	10	10	10	10
Ets@123cap	10	6	8	10	10	10	10
Mount@345s	10	6	8	10	10	10	10
234$testMe	10	6	8	10	10	10	10
wHen@u234s	10	6	8	10	10	10	10
some$123He	10	6	8	10	10	10	10
How$ukn235	10	6	8	10	10	10	10
Pets$567we	10	6	8	10	10	10	10
Tttt@789un	10	6	8	10	10	10	10
Gggggg@234	10	6	8	10	10	10	10
Mmmmmm@111	10	6	8	10	10	10	10
Jjjjjj$999	10	6	8	10	10	10	10
tesest$123	10	6	8	10	10	10	10
$Pppppp234	10	6	8	10	10	10	10
Hhhhh@123h	10	6	8	10	10	10	10
Whyu@me789	10	6	8	10	10	10	10
how@Cap456	10	6	8	10	10	10	10
Average	10	6	8	10	10	10	10

Table-A I-4 Experimental results of the the HTC Desire HD at session 1

HTC Desire HD session 1							
Passwords	CIBC Bank	BMO Bank	Natioanl Bank	RBC Bank	Scotia Bank	TD canada Bank	PayPal
Alq012$suv	6	4	4	6	7	7	7
zSd@347ege	7	3	5	7	7	7	7
123Pa$sword	7	4	4	7	7	6	7
Ghau789@fb	6	4	5	7	7	6	7
het239$quI	7	4	5	6	6	7	7
123@Myname	7	3	5	7	8	8	8
Canda$987p	8	4	5	7	7	8	7
loL$123lol	6	3	5	6	7	6	7
Home@123jk	7	4	6	8	7	7	7
Capme@123k	8	4	5	7	7	7	8
Test$890me	6	3	5	7	6	6	7
suv345@Qsd	8	4	5	7	7	7	7
where789@d	6	3	4	8	7	7	7
my478@Test	7	4	6	7	8	7	8
Ets@123cap	7	4	5	8	7	6	7
Mount@345s	7	4	4	7	8	7	8
234$testMe	8	5	5	8	8	8	8
wHen@u234s	7	4	4	7	7	8	7
some$123He	8	4	4	8	7	7	7
How$ukn235	7	4	5	8	7	8	7
Pets$567we	6	4	4	7	7	7	7
Tttt@789un	6	3	4	7	7	6	7
Gggggg@234	6	3	4	7	7	6	7
Mmmmmm@111	6	3	4	6	6	6	6
Jjjjjj$999	6	3	4	6	6	6	6
tesest$123	7	4	5	7	7	8	7
$Pppppp234	6	3	4	6	6	6	6
Hhhhh@123h	6	3	4	6	8	7	7
Whyu@me789	8	4	5	8	8	8	8
how@Cap456	8	4	5	8	7	8	8
Average	6.8	3.6	4.6	7.0	7.0	6.9	7.1

Table-A I-5 Experimental results of the the HTC Desire HD at session 2

HTC Desire HD session 2							
Passwords	CIBC Bank	BMO Bank	Natioanl Bank	RBC Bank	Scotia Bank	TD canada Bank	PayPal
Alq012$suv	10	6	8	10	10	10	10
zSd@347ege	9	6	8	9	10	9	10
123Pa$sword	9	6	8	9	9	9	10
Ghau789@fb	10	6	8	10	10	10	10
het239$quI	10	6	8	10	10	10	10
123@Myname	10	6	8	10	10	10	10
Canda$987p	10	6	8	10	10	10	10
loL$123lol	9	6	8	9	8	9	9
Home@123jk	9	6	8	10	10	10	10
Capme@123k	10	6	8	10	10	10	10
Test$890me	10	6	8	10	10	9	10
suv345@Qsd	9	6	8	9	10	9	10
where789@d	9	6	8	10	10	10	10
my478@Test	10	6	8	10	10	10	10
Ets@123cap	10	6	8	10	10	10	10
Mount@345s	10	6	8	10	10	10	10
234$testMe	9	6	8	9	10	9	10
wHen@u234s	10	6	8	10	10	10	10
some$123He	9	6	8	9	9	9	9
How$ukn235	10	6	8	10	10	10	10
Pets$567we	9	6	8	10	9	9	9
Tttt@789un	9	6	8	9	9	9	9
Gggggg@234	9	5	7	8	9	9	8
Mmmmmm@111	8	5	7	8	8	9	9
Jjjjjj$999	9	6	8	8	9	8	8
tesest$123	9	6	8	8	9	9	9
$Pppppp234	8	6	7	8	9	8	9
Hhhhh@123h	8	5	8	9	8	9	8
Whyu@me789	10	6	8	10	10	10	10
how@Cap456	10	6	8	10	10	10	10
Average	9.3	5.9	7.9	9.4	9.5	9.4	9.5

Table-A I-6 Experimental results of the the HTC Desire HD at session 3

HTC Desire HD session 3							
Passwords	CIBC Bank	BMO Bank	Natioanl Bank	RBC Bank	Scotia Bank	TD canada Bank	PayPal
Alq012$suv	10	6	8	10	10	10	10
zSd@347ege	10	6	8	10	10	10	10
123Pa$sword	10	6	8	10	10	10	10
Ghau789@fb	10	6	8	10	10	10	10
het239$quI	10	6	8	10	10	10	10
123@Myname	10	6	8	10	10	10	10
Canda$987p	10	6	8	10	10	10	10
loL$123lol	10	6	8	10	10	10	10
Home@123jk	10	6	8	10	10	10	10
Capme@123k	10	6	8	10	10	10	10
Test$890me	10	6	8	10	10	10	10
suv345@Qsd	10	6	8	10	10	10	10
where789@d	10	6	8	10	10	10	10
my478@Test	10	6	8	10	10	10	10
Ets@123cap	10	6	8	10	10	10	10
Mount@345s	10	6	8	10	10	10	10
234$testMe	10	6	8	10	10	10	10
wHen@u234s	10	6	8	10	10	10	10
some$123He	10	6	8	10	10	10	10
How$ukn235	10	6	8	10	10	10	10
Pets$567we	10	6	8	10	10	10	10
Tttt@789un	10	6	8	10	10	10	10
Gggggg@234	10	6	8	10	10	10	10
Mmmmmm@111	10	6	8	10	10	10	10
Jjjjjj$999	10	6	8	10	10	10	10
tesest$123	10	6	8	10	10	10	10
$Pppppp234	10	6	8	10	10	10	10
Hhhhh@123h	10	6	8	10	10	10	10
Whyu@me789	10	6	8	10	10	10	10
how@Cap456	10	6	8	10	10	10	10
Average	10	6	8	10	10	10	10

BIBLIOGRAPHY

[1] http://old.nabble.com/L4Android-performance-issue-td34125968.html.

[2] "Android app manifest," http://developer.android.com/guide/topics/manifest/manifest-intro.html.

[3] "Android debug bridge," http://developer.android.com/tools/help/adb.html.

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

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

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

[7] "Android binder," http://elinux.org/Android_Binder, 2008.

[8] "Android low memory killer," https://source.android.com/devices/tech/low-ram.html, 2008.

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

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

[11] "Importance foreground info," http://developer.android.com/reference/android/app/ActivityManager.RunningAppProcessInfo.html#IMPORTANCE_FOREGROUND, Nov 2009.

[12] "Android kernel configuration," https://source.android.com/devices/tech/kernel.html, 2010.

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

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

[15] "Cve-2011-1149," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1149, 2011.

[16] "Cve-2011-1149," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1717, 2011.

[17] "ioctl - control device," http://man7.org/linux/man-pages/man2/ioctl.2.html, Nov 2011.

[18] "Su application," https://github.com/ChainsDD/Superuser, Sept 2011.

[19] “Customizing selinux,” https://source.android.com/devices/tech/security/selinux/customize.html, 2012.

[20] “Implementing selinux,” https://source.android.com/devices/tech/security/selinux/implement.html, 2012.

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

[22] “Tizan operating system,” https://www.tizen.org/, January 2012.

[23] “Android jelly beans,” https://www.android.com/versions/jelly-bean-4-3/, July 2013.

[24] “Android se-policy read-me,” https://android.googlesource.com/platform/external/sepolicy/+/android-4.3.1_r1/README, March 2013.

[25] *Embedded Android: Porting, Extending, and Customizing*. " O’Reilly Media, Inc.", 2013.

[26] “Firefox os,” https://www.mozilla.org/en-US/firefox/os/, April 2013.

[27] “Google glass,” http://www.google.com/glass/start/, Feb 2013.

[28] “Mac_ permission.xml,” https://android.googlesource.com/platform/external/sepolicy/$+$/android-4.4.4_r2.0.1/mac_~permissions.xml, Aug 2013.

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

[30] “Se-policy source code,” https://android.googlesource.com/platform/external/sepolicy/, 2013.

[31] “Training tesseract3,” https://code.google.com/p/tesseract-ocr/wiki/TrainingTesseract3, May 2013.

[32] “Ubuntu touch,” http://www.ubuntu.com/phone, March 2013.

[33] “Access secure flinger permission,” http://developer.android.com/reference/android/Manifest.permission.html#ACCESS_SURFACE_FLINGER, Aug 2014.

[34] “Access_vectors,” https://android.googlesource.com/platform/external/sepolicy/$+$/android-4.4.4_r2.0.1/access_vectors, Aug 2014.

[35] “Android activity manager,” http://developer.android.com/reference/android/app/ActivityManager.html, Aug 2014.

[36] “Android alarm manager,” http://developer.android.com/reference/android/view/WindowManager.html, Aug 2014.

[37] “Android kit kat,” http://www.android.com/versions/kit-kat-4-4/, July 2014.

[38] “Android package manager,” http://developer.android.com/reference/android/content/pm/PackageManager.html, Aug 2014.

[39] “Android vulnerability trends over time,” http://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224, 2014.

[40] “Android window manager,” http://developer.android.com/reference/android/app/AlarmManager.html, Sept 2014.

[41] “Attributes,” https://android.googlesource.com/platform/external/sepolicy/$+$/android-4.4.4_r2.0.1/attributes, Aug 2014.

[42] “Domain.te,” https://android.googlesource.com/platform/external/sepolicy/$+$/android-4.4.4_r2.0.1/domain.te, Aug 2014.

[43] “File.te,” https://android.googlesource.com/platform/external/sepolicy/$+$/android-4.4.4_r2.0.1/file.te, Aug 2014.

[44] “File_context,” https://android.googlesource.com/platform/external/sepolicy/$+$/android-4.4.4_r2.0.1/file_contexts, Aug 2014.

[45] “Flag_secure,” http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE, Aug 2014.

[46] “Multi level security mls,” https://android.googlesource.com/platform/external/sepolicy/$+$/android-4.4.4_r2.0.1/mls, Aug 2014.

[47] “Properties_context,” https://android.googlesource.com/platform/external/sepolicy/$+$/android-4.4.4r2.0.1/property_contexts, Aug 2014.

[48] “Seapp_context,” https://android.googlesource.com/platform/external/sepolicy/$+$/android-4.4.4_r2.0.1/seapp_contexts, Aug 2014.

[49] “Security_class,” https://android.googlesource.com/platform/external/sepolicy/$+$/android-4.4.4_r2.0.1/security_classes, Aug 2014.

[50] “Surface flinger and hardware composer,” https://source.android.com/devices/graphics/architecture.html#SurfaceFlinger, Aug 2014.

[51] “Surface view,” http://developer.android.com/reference/android/view/SurfaceView.html, Aug 2014.

[52] “Google play,” http://en.wikipedia.org/wiki/Google_Play, March 2015.

[53] 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.

[54] M. Ankapura, "Screencap command," https://android.googlesource.com/platform/frameworks/base/+/jb-release/cmds/screencap/screencap.cpp, Aug 2012.

[55] I. Apple, "Ios 8," https://www.apple.com/ios/, Aug 2014.

[56] ARM, "Trust zone technology," http://www.arm.com/products/processors/technologies/trustzone/.

[57] bart1996, "Screenshot does not work anymore?" (Sept. 2012) XDA-Developer.

[58] J. Boutet and L. Homsher, "Malicious android applications: Risks and exploitation," *SANS Institute*, vol. 22, 2010.

[59] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry, "Practical and lightweight domain isolation on android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 51–62.

[60] 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.

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

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

[63] 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.

[64] 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.

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

[66] L. Davi, A. Dmitrienko, C. Kowalski, and M. Winandy, "Trusted virtual domains on okl4: Secure information sharing on smartphones," in *Proceedings of the sixth ACM workshop on Scalable trusted computing*. ACM, 2011, pp. 49–58.

[67] A. Diaz, "Ocrad ocr engine," https://www.gnu.org/software/ocrad/, Feb 2004.

[68] P. Dornbusch, M. Möller, and A. Buttermann, *IT-security in global corporate networks*. BoD–Books on Demand, 2003.

[69] E. Dupuy, "Java decompiler," http://jd.benow.ca/, 2008.

[70] A. ECN, "Cpu monitor," https://play.google.com/store/apps/details?id=com.bigbro.ProcessProfiler&hl=en, Sept 2014.

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

[72] I. Eidus, "How to use the kernel samepage merging," https://www.kernel.org/doc/Documentation/vm/ksm.txt, Nov 2009.

[73] M. A. El-Serngawy and C. Talhi, "Securing business data on android smartphones," in *Mobile Web Information Systems*. Springer, 2014, pp. 218–232.

[74] FFIEC, "Financial institutes authentication," http://ithandbook.ffiec.gov/it-booklets/information-security/security-controls-implementation/access-control-/authentication.aspx, July 2006.

[75] T. Fiebig, J. Danisevskis, and M. Piekarska, "A metric for the evaluation and comparison of keylogger performance," Aug. 2014. [Online]. Available: https://www.usenix.org/conference/cset14/workshop-program/presentation/fiebig

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

[77] D. C. Frank Mayer, Karl MacMillan, "Linux security module," in *SELinux by Example: Using Security Enhanced Linux*, July 2006, pp. 40–54.

[78] D. C. Frank Mayer's, Karl MacMillan, "The origins of mandatory access control," in *SELinux by Example: Using Security Enhanced Linux*, July 2006, pp. 5–13.

[79] Google-Inc, "Android wear sdk," http://developer.android.com/wear/index.html.

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

[81] R. Haines's, "Security enhancements for android," in *The SELinux Notebook-4th Edition*, vol. 4, 2014, pp. 296–345.

[82] B. Hana, "Bank hana," https://play.google.com/store/apps/details?id=com.hanabank.ebk.channel.android.hananbank&hl=en, Jul 2014.

[83] R. Hat, "libvirt the virtualization api," http://libvirt.org/, Dec 2005.

[84] HiJames, "Screenshot app for unrooted users," (Apr. 2013) XDA-Developer.

[85] HP-Labs, "Tesseract ocr," https://code.google.com/p/tesseract-ocr/, 1985.

[86] Ice-Cold-Apps, "Screenshot ultimate," https://play.google.com/store/apps/details?id=com.icecoldapps.screenshotultimate, Jan 2014.

[87] I. D. C. (IDC), "Android and ios combine for 91.1% of the worldwide smartphone os market in 4q 12," http://www.businesswire.com/news/home/20130214005415/en/Android-iOS-Combinid, Feb 2013.

[88] IntelliJ-IDEA, "Android apk tool," https://code.google.com/p/android-apktool/, March 2010.

[89] J. Jakub.Li, "Android screenshot llibrary," https://code.google.com/p/android-screenshot-library/, Aug 2010.

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

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

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

[93] 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.

[94] U. Kumar, P. Kodeswaran, V. Nandakumar, and S. Kapoor, "Comp. inf. sci. & eng., univ. of florida, gainesville, fl, usa," in *MILITARY COMMUNICATIONS CONFERENCE, 2012-MILCOM 2012*. IEEE, 2012, pp. 1–6.

[95] A. Lackorzynski *et al.*, "L4linux porting optimizations," *Master's thesis, Technische Universitat Dresden*, 2004.

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

[97] C. Lee, J. Kim, S.-j. Cho, J. Choi, and Y. Park, "Unified security enhancement framework for the android operating system," *The Journal of Supercomputing*, vol. 67, no. 3, pp. 738–756, 2014.

[98] W. LeFebvre, "Unix top utilities," http://www.unixtop.org/man.shtml, 2003.

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

[100] M. Lorch, S. Proctor, R. Lepro, D. Kafura, and S. Shah, "First experiences using xacml for access control in distributed systems," in *Proceedings of the 2003 ACM workshop on XML security*. ACM, 2003, pp. 25–37.

[101] F. Maggi, A. Volpatto, S. Gasparini, G. Boracchi, and S. Zanero, "A fast eavesdropping attack against touchscreens," in *Information Assurance and Security (IAS), 2011 7th International Conference on*. IEEE, 2011, pp. 320–325.

[102] T. Micro, "Endpoint security solutions," http://www.trendmicro.com/us/enterprise/product-security/.

[103] M. Nauman, S. Khan, and X. Zhang, "Apex: extending android permission model and enforcement with user-defined runtime constraints," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ACM, 2010, pp. 328–332.

[104] M. Nelissen, "Isurface composer interface," https://android.googlesource.com/platform/frameworks/native/+/jb-dev/include/gui/ISurfaceComposer.h, Mar 2014.

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

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

[107] A. Platform, "Package parser," https://android.googlesource.com/platform/frameworks/base/+/483f3b06ea84440a082e21b68ec2c2e54046f5a6/core/java/android/content/pm/PackageParser.java, 2007.

[108] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J.-M. Frahm, "ispy: automatic reconstruction of typed input from compromising reflections," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 527–536.

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

[110] 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.

[111] A. Rishi, "Ocr - text recognition app," https://play.google.com/store/apps/details?id=com.offline.ocr.english.image.to.text, Nov 2014.

[112] 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.

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

[114] Samsung-Inc, "Samsung galaxy gear," http://www.samsung.com/ca/consumer/mobile/galaxy-gear/galaxy-gear/SM-V7000ZKAXAC, Sept 2013.

[115] F. Schaub, R. Deyhle, and M. Weber, "Password entry usability and shoulder surfing susceptibility on different smartphone platforms," in *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*. ACM, 2012, p. 13.

[116] 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*.

[117] J. Schulenburg, "Gocr open-source character recognition," http://jocr.sourceforge.net/, 2000.

[118] Sebastian, "Zygote jailbreak (zimperlich) sources," http://c-skills.blogspot.ca/2011/02/zimperlich-sources.html, Feb 2011.

[119] A. Shanks, "Su binary," https://github.com/ChainsDD/su-binary, Sept 2011.

[120] S. Smalley and R. Craig, "Security enhanced (se) android: Bringing flexible mac to android." in *NDSS*, 2013.

[121] SmartUX, "Screenshot ux," https://play.google.com/store/apps/details?id=com.liveov.shotuxtrial, Sept 2012.

[122] R. Smith, "An overview of the tesseract ocr engine." in *ICDAR*, vol. 7, 2007, pp. 629–633.

[123] R. Smith, D. Antonova, and D.-S. Lee, "Adapting the tesseract open source ocr engine for multilingual ocr," in *Proceedings of the International Workshop on Multilingual OCR*. ACM, 2009, pp. 1–8.

[124] J. Steinberg, "System and method of using two or more multi-factor authentication mechanisms to authenticate online parties," Nov 2006, uS Patent App. 11/606,788.

[125] K. Suzaki, K. Iijima, T. Yagi, and C. Artho, "Memory deduplication as a threat to the guest os," in *Proceedings of the Fourth European Workshop on System Security*. ACM, 2011, p. 6.

[126] N. Titov, "Ocr and text search – tfinder," https://play.google.com/store/apps/details?id=com.tnstudio.tfinder, Sept 2014.

[127] B. Unhelkar and S. Murugesan, "The enterprise mobile applications development framework," *IT professional*, vol. 12, no. 3, pp. 0033–39, 2010.

[128] G. Uytterhoeven, "The frame buffer device," https://www.kernel.org/doc/Documentation/fb/framebuffer.txt, May 2001.

[129] V. Ware, "Horizon mobile secure workplace," http://www.vmware.com/mobile-secure-desktop/.

[130] T.-E. Wei, A. B. Jeng, H.-M. Lee, C.-H. Chen, and C.-W. Tien, "Android privacy," in *Machine Learning and Cybernetics (ICMLC), 2012 International Conference on*, vol. 5. IEEE, 2012, pp. 1830–1837.

[131] 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.

[132] B. Woori, "Woori global banking," https://play.google.com/store/apps/details?id=com.wooribank.smlg_android, Jul 2014.

[133] 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.

[134] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux security modules: General security support for the linux kernel," in *Foundations of Intrusion Tolerant Systems*. IEEE Computer Society, 2003, pp. 213–213.

[135] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2012, pp. 113–124.

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

[137] D.-H. You and B.-N. Noh, "Android platform based linux kernel rootkit," in *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*. IEEE, 2011, pp. 79–87.

[138] S. Zhong, L. Li, Y. G. Liu, and Y. R. Yang, "Privacy-preserving location-based services for mobile users in wireless networks," *Department of Computer Science, Yale University, Technical Report ALEU/DCS/TR-1297*, 2004.