Graduate School ETD Form 9
(Revised 12/07)

# PURDUE UNIVERSITY
## GRADUATE SCHOOL
## Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Ishita Verma

Entitled

A Security Analysis of Smartphones

For the degree of Master of Science in Electrical and Computer Engineering

Is approved by the final examining committee:

Brian King
Chair

Maher Rizkalla

Jaehwan (John) Lee

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Brian King

Approved by: Brian King 06/21/2011
Head of the Graduate Program Date

Graduate School Form 20
(Revised 9/10)

# PURDUE UNIVERSITY
## GRADUATE SCHOOL

## Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

A Security Analysis of Smartphones

For the degree of Master of Science in Electrical and Computer Engineering

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Ishita Verma

Printed Name and Signature of Candidate

07/27/2011

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

# A SECURITY ANALYSIS OF SMARTPHONES

A Thesis

Submitted to the Faculty

of

Purdue University

by

Ishita Verma

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

August 2011

Purdue University

Indianapolis, Indiana

To my Mom & Dad

# ACKNOWLEDGMENTS

My sincere gratitude goes to Dr. Brian King. I am thankful that he encouraged and guided me all through my graduate studies, but also because he advised and supported me in every difficulty.

I am also very thankful to Dr. Maher Rizkalla, Dr. John Lee and Dr. Lauren Christopher, both, for their presence in my thesis committee, as well as for the knowledge and confidence that they imparted to me during all my undergraduate and graduate years.

Lastly, I am thankful for my family, who I owe all my love and happiness to, and for my friends, lab peers and mentors who I would be lost without on campus.

TABLE OF CONTENTS

Page

Page

# LIST OF TABLES

# LIST OF FIGURES

# ABBREVIATIONS

| | |
|---|---|
| API | Application Programming Interface |
| BLP | BellLaPadula (model) |
| CA | Certificate Authority |
| DAC | Discretionary Access Control |
| DFU | Device Firmware Update |
| DNS | Domain Name System |
| DOS | Denial of Service |
| IP | Internet Protocol |
| IPC | Inter Process Communication |
| GPS | Global Positioning System |
| LLB | Low Level Bootloader |
| MAC | Mandatory Access Control |
| MNC | Mobile Network Code |
| MTM | Mobile Trusted Modul |
| NFC | Near Field Communication |
| OS | Operating System |
| POSIX | Portable Operating System Interface for Unix |
| PKI | Public Key Infrastructure |
| RBAC | Role Based Access Control |
| RDP | Remote Desktop Protocol |
| RAM | Random Access Memory |
| SHA | Secure Hash Algorithm |
| SMS | Short Message Service |
| SSH | Secure Shell Protocol |

| | |
|---|---|
| SHFT | Secure Shell File Transfer Protocol |
| PC | Personal Computer |
| PCI | Peripheral Component Interconnect |
| SIM | Subscriber Identity Module (card) |
| TCB | Trusted Computing Base |
| TCP | Transmission Control Protocl |
| TLS | Transport Layer Security |
| UDP | User Datagram Protocol |
| UI | User Interface |
| UID | Unique Identifier |
| URL | Uniform Resource Locator |
| USB | Universal Serial Bus |
| VOIP | Voice over Internet Protocol |

# ABSTRACT

Verma, Ishita. M.S.E.C.E., Purdue University, August 2011. A Security Analysis of Smartphones. Major Professor: Brian King.

This work analyzes and discusses the current security environment of today's (and future) smartphones, and proposes a security model which will reduce smartphone vulnerabilities, preserving privacy, integrity and availability of smartphone native applications to authorized parties. For this purpose, we begin with an overlook of current smartphone security standards, and explore the threats, vulnerabilities and attacks on them, that have been uncovered so far with existing popular smartphones. We also look ahead at the future uses of the smartphones, and the security threats that these newer applications would introduce. We use this knowledge to construct a mathematical model, which gives way to policies that should be followed to secure the smartphone under the model. We finally discuss existing and proposed security mechanisms that can be incorporated in the smartphone architecture to meet the set policies, and thus the set security standards.

# 1. INTRODUCTION

In recent years, smartphones, combining telephony and mobile computing, have emerged as a popular trend in consumer electronics. A smartphone is a category of mobile device that provides advanced capabilities beyond a typical mobile phone; running complete operating system software that provides a standardized interface and platform for application developers [1]. Today's smartphones combine high utility, mobility and entertainment, owing to device features such as 4G network connectivity, touch screen, accelerometer, GPS, and upto 1.5 GHz processing speed. Also, smartphones support a sophisticated variety of user applications. However, this means many of these applications may be security-critical, such as mobile banking, or may be untrusted such as third-party games. Also accompanying the number of smartphone applications, there is a corresponding growth in amount of private data stored unprotected on smartphones.

Generally when compared to feature phones, smartphones have larger displays, more powerful processors, advanced computing ability and better data and cellular connectivity [1] [2]. In the spectrum between feature phones and personal computing devices such as PCs or netbooks, the smartphone has started tending more towards the latter than the earlier, in terms of computing power and complexity. On one hand, a smartphone is a mobile phone that offers more advanced computing capability and connectivity than a feature phone, and is a truly pervasive mobile computing device [3]. On the other hand, gradually all applications that are developed for a desktop are now being redone for the smartphone. However, from a security perspective, there are a few essential features that make the smartphone security need unique; these are:

- the smartphone is designed to be single-user, and there is no user login.
- the smartphone provides services through independent server processes that are always on. They always run and are not attached to a user session. As long as power is supplied, the platform is always on, even if no user is logged on.
- the smartphone, especially with its higher usability enhancements, is meant to be used by a large public with little technical understanding, when compared to a desktop.
- the smartphone is an always-connected device, and security breaches in the smartphone will attack a network much larger than the device itself.
- the smartphone has unique access to both cell communication and data network.
- the smartphone is a constrained embedded device in terms or resources such as memory, processing and battery life.
- the smartphone is designed to host several applications that leverage its portability, connectivity and usability.

Statistics indicating the popularity of smartphones have been tending upwards in recent years. In [4], the authors provided the following information: In December 2010, Nielsen reported that 31 percent of US mobile phone owners have a smartphone. Morgan Stanley Research estimated that sales of smartphones will exceed those of PCs in 2012. Gartner stated smartphones accounted for 297 million (19 percent) of the 1.6 billion mobile phones sold in 2010 and forecasted over 500 million smartphones to sell in 2012.

Below, see Figure 1.1, illustrates the percent market hold of the most popular smartphone brands from comScore, Inc. By February 2011 Android held the number one position with 33.0 percent market share. RIM ranked second with 28.9 percent market share, followed by Apple with 25.2 percent, Microsoft with 7.7 percent and Palm with 2.8 percent rounded out the top five. [4]

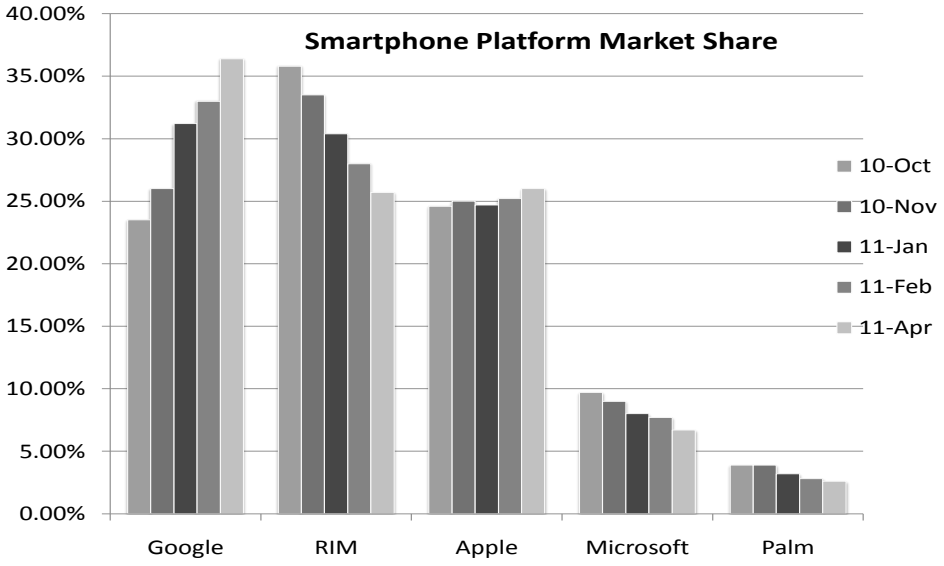


Fig. 1.1.: Market share of smartphone brands

Smartphones have shown continued growth in the areas of advanced processing power, such as currently demonstrated TI four core ARM smartphone processor [5]. Also smartphones are to become more situationally and contextually aware, and offer decision making information to users based on past preference possibly with help of background cloud intelligence. Applications such as augmented reality, social networks and gaming networks show that smartphone are going to be used in aggregation with each other; while also increasing the attack surface and raising raise privacy alarms. Also some more emerging smartphone applications are electronic wallet, electronic ID, car/home keys, as well as portable hard drive. Hence such ambitious applications for the smartphone need a well crafted security policy if users expect fluid connectivity and information sharing, as well as storage of critical data on their smartphones. [6]

In terms of smartphone usage [6], an average of statistics from October 2010 to April 2011, shows that 68.2 percent of U.S. mobile subscribers used text messaging on their mobile device. Browsers were used by 37.2 percent of subscribers, while downloaded applications were used by 35.4 percent of the mobile audience. Accessing

of social networking sites or blogs represented 25.6 percent of mobile subscribers. Playing games represented 24.2 percent of the mobile audience, while listening to music represented 16.5 percent. Looking at Figure 1.2, we see a dominating column for telephony and texting. However as the usage patterns of smartphones broaden, one can soon expect usage of other features to start rising higher. Also applications like e-wallet and location based services to start dominating, which will bring along several privacy concerns.

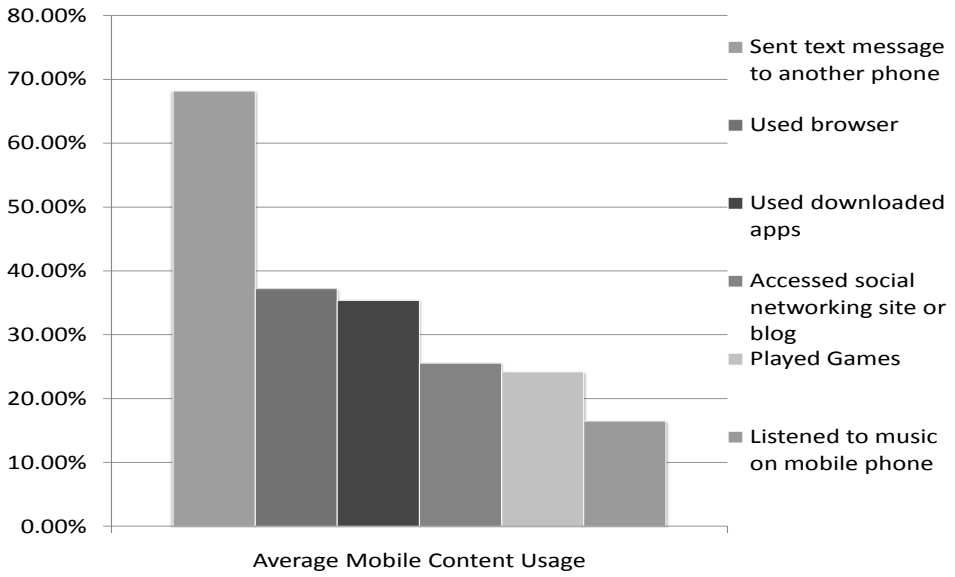


Fig. 1.2.: U.S. mobile subscription

While there are many ideas for future applications, currently smartphones have already gained the attention of many security analysts owing to their popularity, ubiquity as well as ongoing incidences of security and privacy breaches on them. As per a Symantec report, there were 163 known vulnerabilities in mobile operating systems in 2010, up 42 percent compared to 115 in 2009, and criminals are viewing mobile phone hacking as a potentially lucrative activity [7]. Similarly, Panda Security's latest quarterly malware report found smartphone malware dominated the security landscape during the first quarter of 2011. There report highlight several major security

incidents, including the malicious apps that were found on the Android Market and the successful attack against HBGary Federal by the Anonymous hacktivist group [8]. Cyber-attackers created 26 percent more new threats in this quarter than they did during the first quarter of 2010, and 16 percent more than the fourth quarter of 2010. Also, the laboratory received an average of 73,190 new samples of malware everyday, of which 70 percent were Trojans. [8]

The pervasive nature of smartphones and a large, unsophisticated user base also make smartphones prone to attacks. At the same time it is also required to be mindful of smartphones as computers that support telephony and hence their possible security impact on and from the communication network. Given below are recent news of privacy and security breaches on smartphones:

**Example 1** *As per reports in April 2011 [9] a recent vulnerability in the Voice-over-IP Skype application for Android, let private user data such as user profile and contact information and instant message logs unencrypted and in open to be exploited by hackers.to a report. Moreover, Skype stored the username in a static location, and so with every device the hacker could parse the same file, get the username and find the path to Skype's stored data .*

**Example 2** *In April 2011 Apple [10]updated its Safari browser to fix an issue relating to its acceptance of security certificates. The vulnerability could have allowed an attack utilizing fake SSL certificates for a "man in the middle" attack that could have obtained confidential information from a local network.*

**Example 3** *According to a last quarters e-Week report [8], cyber criminals have once again been infecting smartphones with malware that generates premium-rate text messages. Users are unaware of these messages being sent until they receive their monthly bills. As an example, a Russian gang created a Valentine Day application to send romantic images, which unwary to the user sent the message to a premium rate number.*

**Example 4** *In the same report [8], a smartphone Trojan virus was designed to bypass the double authentication system implemented by many banks and financial institutions, which prompted users to enter a phone number to which security certificate should be sent to. When users downloaded the certificate, it had the capability to intercept all SMS messages sent to the phone, such as password codes and security hints used to secure bank accounts.*

**Example 5** *To improve Apple's location based services, every 12 hours, an iOS device's stored geodata gets anonymized with a random string of numbers, and it gets transmitted to Apple in a batch. However two data scientists broke the news [11] that in doing so an unencrypted file was stored iOS devices contains a detailed log of the device's geographical data dating back 10 months.*

**Example 6** *Federal prosecutors in New Jersey [12] are investigating whether numerous smartphone applications illegally obtained or transmitted information about their users without proper disclosures. Federal laws prohibit unauthorized access to information. This probe was significant because it involved potentially criminal charges that could be applicable to numerous companies.*

In this work we highlight security and privacy concerns in the growing market of smartphones and construct security solutions so that current and future applications be developed securily on this platform. We discuss current smartphone platforms, reported and contrived security breaches, as well as solutions to fortify smartphone security. If development for security solutions for smartphones is not kept in pace with the growth in its applications, a smartphone can risk much about the user from current and past locations to personal identifying data and from malicious parties taking control of the device and its services to attacks on the communication channels.

The outline of this study is as follows: Chapter 2 provides background material on operating system security. Chapter 3 explores current smartphone platforms. Chapter 4 is a look into some sensitive near future uses of the smartphone. Chapter 4 is an our list of ten categories of smartphone threats and vulnerabilities. Chapter 6,

is on related work in the field to counterattack some of these threats. Chapter 7 are four attacks that we recreated and analysed on the smartphone. Having constructed the threat model, Chapter 7 describes security policies which should be used secure the smartphone, while Chpater 8 describes a a formal security model and mechanism for the smarthone. Chapter 9 analyses our proposed solutions against the threats we had previously presented. Finally, Chapter 10 concludes this work.

# 2. SECURITY AND PRIVACY IN INFORMATION SYSTEMS

Smartphones need to be secured from security and privacy violations. A smartphone is an example of an information system, it can be analyzed from the point of view of information systems security and privacy. In this chapter, we look at key security terms, principals and models, that can help us start the security analysis of smartphones.

## 2.1 System Security

Computer security rests on the cornerstones of confidentiality, integrity and availability. A system's security is assessed by how well it meets the following three basic requirements:

**Confidentiality** is the concealment of data or resources, maybe even applying to whether such data exists or not [13, p. 4]. It is often supported by the use of access control mechanisms and cryptography. For smartphones, the different stake holders such as the manufacturer, service provider, application developer, and above all the user, expects a portion of its on-device data to remain in confidence, protected from violations by other parties.

**Integrity** is the prevention of improper or unauthorized change to data and resources in the data [13, p. 5]. It involves authentication prior to data write or modification. Integrity mechanisms can be preventive as well as detective. In smartphones, integrity is the guarantee that the information will never be manipulated by untrusted parties. This includes information passed between unauthorized subjects, objects, and both. It ensures that user will be able to

modify, and in some cases, access information only if they explicitly hold the right to do so

**Availability** is the provision requested data, resources or services are provided when required. If a malicious party blocks availability then it is called a denial-of-service (DOS) attack [13, p. 6]. In a smartphone, the functioning of upper-layer applications and even security implementations depend on availability of APIs, the kernel and communication channels. If any of theses are maliciously tied into performing redundant tasks, and is not able to recover from it, then the functioning of the device has been crippled.

### 2.1.1 Secure Design Principles

Saltzer and Schroeder [13, p. 341] draw eight design principles for security systems. These principles though generic and simple, are surprisingly not necessarily followed by several present day smartphone manufacturers. The design principles are:

1. A subject should be given only those privileges that it needs in order to complete a task (Principle of Least privilege). However, coarse grained permissions often provide a larger subset of permissions, then what smartphone applications really need to function.

2. Unless a subject is given explicitly access to an object, it should be denied access to that object (Principle of Fail-Safe defaults). However, many applications data are by default made publicly viewable, removing the need for other applications/parties to request for them.

3. Security mechanisms should be as simple as possible(Principle of Economy of Mechanism). Even though most smartphones try to maintain Mandatory Access Control, the device incorporates several refinements to the basic security model, some of which have subtle side effects and make its overall security difficult to understand.

4. All access to objects be checked to ensure that they are allowed (Principle of Complete Mediation). For instance, based on locality, smartphones do not secure/authenticate physically linked off device communication such as USB.

5. The security of a mechanism should not depend on the secrecy of its design or implementation (Principle of Open Design). A very prominent smartphone manufacturer relies on keeping the operating system design a secret, while in reality the device has been frequently exploited once hackers have been able to reverse engineer library files.

6. A system should not grant permission based on a single condition (Principle of Separation of Privilege). In reality, applications are only authenticated at install time, based on the package certification.

7. Mechanism used to access resources should not be shared (Principle of Least Common Mechanism). However, in all smartphones, application use the same APIs, installer, and directly use the same hardware service hooks.

8. Security mechanisms should not make the resource more difficult to access than if the security mechanism was not present (Principle of Psychological Acceptability). However, the constrained resources of a smartphone, make it difficult to apply common security measures such as anti-malware and firewall, without significantly impacting the device performance.

### 2.1.2 Security Policy and Model

In the following chapter, we shall be seeing the formation of a unified security policy and formal model for smartphones given different threat vectors. Hence, we now introduce the definitions of the two terms as tools that set the context of a secure system. What is secure under one policy may not be secure under another. But if we are able to establish a secure smartphone model under which all foreseeable security

issues are handled, then we can move on to base security policies and mechanisms off the model, in the pursuit of a secured smartphone system.

- A *security policy* with respect to confidentiality, identifies those states in which information flow happens to to those not authorized to receive it. With respect to integrity it identifies authorized ways and entities to alter information. And with respect to availability a security policy defines what services may be provided [13, p. 99].

- A *security model* abstracts specific characteristics of policies, and hence represent that set of policies [13, p. 99].

### 2.1.3 Access Control

Access control is a system that control access to resources. Access control is required in smartphones to monitor activities so that confidentiality, integrity and availability is maintained. There are three type of access control that we can use alone or together in security policies for smartphones.

- Discretionary Access Control (DAC) bases access rights on the identity of the subject and the identity of the object involved. Owner of the object can control who has access to it [13, p. 103].

- Mandatory Access Control (MAC) is rule based access control enforced by the operating system and unalterable by the subjects or owner of objects [13, p. 103].

- Role Based Acces Control (RBAC) is a form of non-discretionary, mandatory access control, but it is not based on multilevel security requirements.Access control decisions are often determined by the roles individual users take on as part of an organization. This includes the specification of duties, responsibilities, and qualifications. [14]

We now look at two classic confidentiality and integrity security models, which have been the basis of many other security models being adapted from them. In Chapter 7, we would try to derive a mixed security model, using the follow two models as basis.

### 2.1.4 Confidentiality Model

The Bell-LaPedula (BLP) model places subjects and objects in a set of security clearance $l$ arranged in a linear ordering, with the higher the ordering the more sensitive the data [13, p. 134]. On such a system of security clearances, the following rules [13, p. 134] are applied to subjects accessing objects :

$S$ can read $O$ if and only if $l_o \leq l_s$ and $S$ has discretionary read access to $O$

$S$ can write $O$ if and only if $l_s \leq l_o$ and $S$ has discretionary write access to $O$

The model focuses on confidentiality, and is mainly applied for military setting for highly classified information. In order to maintain confidentiality a subject can only read information at or below the subject's security clearance level. While it can pass information to objects at same or higher security clearance. If instead it writes to a lower clearance, than it may cause leaking some of its own information. As would if a lower clearance subject is allowed to read information at a higher level.

### 2.1.5 Integrity Model

The Biba Integrity Model is a formal state transition system of computer security policy that describes a set of access control rules designed to ensure data integrity. Data and subjects are grouped into ordered levels of integrity. The model is designed so that subjects may not corrupt objects in a level ranked higher than the subject, or be corrupted by objects from a lower level than the subject. It is dual to the Bell-LaPadula Model, and its rules are [13, p. 155]:

$s \in S$ can read $o \in O$ if and only if $i(s) \leq i(o)$

$s \in S$ can write $o \in O$ if and only if $i(o) \leq i(s)$

$s \in S_1$ can execute $s_2 \in S$ if and only if $i(s_2) \leq i(o_2)$

The Biba model is a the complement of the BLP model, and if both are strongly applied to any system, then that would mean that subjects can only read and write to objects at its own security level. In the Biba model, a subject is allowed to read at a higher level, but write only at equal or lesser levels. This is because, if a subject is allowed to read from a lesser trusted source than its own trustworthiness will be compromised.

## 2.2 System Privacy

We now look at another facet of security, which is privacy. Information systems privacy is to protect individual user against the system's misuse or failure to protect the user's the private data [15, p. 595]. Privacy threats can be of the form of data mining where aggregation of data about individual make large scale correlations easy, user profiling where user data is gathered off a device piece by piece to recreate a users identity (Salami attack), as well as identity theft which involves a malicious party disguising in another users identity to use against information systems [15, p. 595]. Common controls protecting privacy are authentication (pins, passwords, challenge-response systems, tokens, biometrics and one-time passwords), anonymizers which do not leave trail of activity, computer voting, pseudonymity and legal controls [15, p. 601].

With smartphones, there is personal data starting to be leaked as soon as the bare device is switched on, even without privacy violating application add-ons. Even if the device is disconnected from the data network and is being used as a simple feature phone, it is holding a host of user information which if retrieved can be traced back the user's identity. It is thus important that user privacy is given sufficient consideration when discussing smartphone security.

### 2.2.1 Information Flow

When a system has a security policy regulation information flow, the system must ensure that the information flows do not violate the constraints of the policy [13, p. 407]. We define it in terms of entropy [13, p. 407], let $c$ be a sequence of commands taking the system from state $s$ to state $t$. Let $x$ and $y$ be objects in the system under the two different states. Then a command sequence $c$ causes a flow of information $x$ to $y$ provided:

$$H(x_s|y_t) < H(x_s|y_s).$$

Stated another way, information flow occurs if value of $y$ allows one to deduce information about the value of $x$. In a system which is sensitive to user privacy, it is required that all information flow is secured and does not cause data leaks.

# 3. CURRENT SMARTPHONE PLATFORMS

In this chapter we present an overview of current popular smartphone models, namely Android, Blackberry, iOS, Symbian and Windows Phone 7. For each, the description is broken down in terms of key security abstractions, in order to contrast the different mechanisms employed by each smartphone.

The *Trusted Computing Platform* (TCB) consists of the operating system, application framework, and core applications. The Trusted Computing Base (TCB) chamber has the greatest privileges. The TCB chamber can modify policy and enforce the security model [16].

The *Trusted Computing Environment* Figure 3.1 is the rest of the device that needs to be protected.

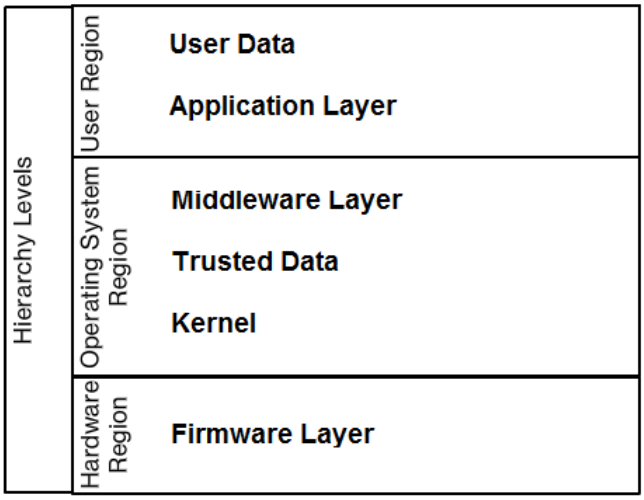


Fig. 3.1.: Trusted computing environment

*Processes Capabilities* is the system of assigning, changing, and checking process access and permissions.

*Data Confinement* is a method of program isolation, where for each application a visibility level can be set at component level hence keeping data private to the application.

*Installer* encompasses all the different channels of entry for outside software to be loaded and given privileges to execute on the device.

## 3.1 Android

**Trusted Computing Base:** The Android software stack is built on the Linux kernel which manages device drivers, networking and memory and process management [17]. The TCB utilizes several Linux security mechanisms such as each application running under a separated POSIX (Portable Operating System Interface) userID, and the application resources being available to only processes of its own userID [18] [17]. With the userIDs, Linux Discretionary Access Control (DAC) mechanisms can be used for access control [19].

Also permissions that are granted at install time cannot be changed, hence following a Mandatory Access Control scheme. Even though the Android Source code is publicly available, the Linux kernel is a highly protected entity and cannot be altered without manipulating hardware [18].

**Trusted Computing Environment:** On top of the kernel, the Android computing environment consists of a layer containing native C-library, SQL database engine, 2D and 3D graphic libraries, native web browser engine (WebKit) and media codecs [17]. This is followed by the next layer, which is the a virtual machine for running the applications. Next the Application Framework layer, includes Google-supplied tools as well as proprietary extensions or services, such as a tool for managing the lifecycle of applications on the next (top) layer [17].

The Android system files contain: operating system relevant files like device files, drivers, libraries, system binaries etc, Android configuration files, Android framework relevant files (e.g. android.awt, android.policy, services, etc.), Android base applications (e.g. Launcher, Browser, Phone, Contacts, etc.) [20]

**Process Capabilities:** Core components and some system processes run with root privileges. Also, there are no fine grained access control mechanisms for system processes [18].

At install time signature/system level permissions are granted based on signature checking. While normal to dangerous protection level permissions are given after user approval. [18]. Each application has its own userID and it is authorized its permissions at install time [18].

**Data Confinement:** Each application runs in its own virtual machine, as a preventive mechanism against buffer overflow, remote code execution and stack smashing [18].

However, Android applications can set an "exported" property label of a component to true, making it visible to external components that have gained permission to access it [18].

**Installer:** Developers are required to sign all applications and package it with an enclosed public key - the signed application is valid until the developer certificate is valid [18] [20]. Some permissions require user authorization while others are granted automatically via the adb install feature. Also, user approval is not fine grained to a subset of permissions that the application request. Lastly, applications from the same developer have a mechanism of sharing permissions [18]. Also, the user is able to use self-signed certificates to sign applications and no central certificate authority is needed [20].

After installation, the application cannot request anymore permissions at runtime.

## 3.2 iOS

**Trusted Computing Base:** The XNU kernel, and core libraries comes from the same code base as the Darwin Operating System developed for the Mac OSX [21] [22]. iOS uses Mandatory Access Controls (MAC) for restricting the capabilities of applications. Also, sandbox is a kernel extension that restricts a set of features from being used for some processes. Once a sandbox is applied, a program cannot access resource out of it [23].

**Trusted Computing Environment:**

**Process Capabilities:** To grant process permissions require writing policy files that describe what permissions an application should have. Users can create new policies to sandbox applications on their system [23]. Permission granting for specific functionality is granted via pop-ups to the user at the run-time of API [23].

**Data Confinement:** The sandbox is used to partition applications from each other and to prevent a malicious application from modifying the underlying system or reading data meant for other applications. Sandbox restrictions are typically enforced at resource acquisition time. The default seatbelt template can be examined in /usr/share/sandbox/SandboxTemplate.sb.

Applications are installed under the 5 documented profiles: TCP/IP networking is prohibited, all sockets-based networking is prohibited, file system writes are prohibited. File system writes are restricted to the specific/temporary folders, and all operating system services are prohibited.

**Installer:** All applications come from the Apple Store. Each application is installed under its own directory and is identified as a User ID. Applications are allowed limited read access to some system areas, but are not allowed to read or write directories belonging to other applications in /private/var/mobile/Applications. Access to the Address Book and Photos is explicitly allowed.

### 3.3 Symbian

**Trusted Computing Base:** In Nokia Symbian, the trusted computing base consists of the kernel, the filesystem, and the software installer. Once again the trusted computing bases is software that enforces permissions and data confinement. [24]

**Trusted Computing Environment:** The trusted computing environment consists of the following software layers in the specified order: 1) cellular platform; 2) adaptation layer, that integrates the generic platform with the phone's cellular platform; 3) operating system layer - from communications, networking, graphics, multimedia to frameworks, libraries and utilities; 4) middleware layer which are domains such as multimedia, networking and location service that serve the application layer, and 5) the application layer which has application specific UI and engine components. [25]

**Process Capabilities:** Capabilities are granted to APIs rather than applications, and they are permission on which applications can access them. The four categories of capabilities are: 1) open to all - meant for generic basic applications, all applications can use this API; 2) user granted install time permissions - user can grant permissions at installation stage; 3) capabilities passed during application certification/signing; 4) manufacturer granted permissions for access to filesystem, DRM or trusted computing base. [24]

**Data Confinement:** Applications can access their own private directories and those directories that are marked as open. The four different access types are: 1) resources folder - every application can read this for icons and bitmaps; 2) system folder - applications are only allowed to be written here at installation time, and the folder can only be read for backup; 3) private folder- one for each applications; and 4) the rest of the filesystem, especially containing user data such as music and documents are open to all applications. [24]

**Installer:** The software installer is part of the trusted computing base, and it only installed symbian signed applications.

### 3.4 Windows Phone 7

**Trusted Computing Base:** The kernel and kernel-mode drivers run in the TCB chamber [16].

**Trusted Computing Environment:** User Space *chambers* have fixed permission sets. The three user space *chambers* are: 1) the Elevated Rights Chamber that can access all resources except security policy and its meant for services intended for use by other phone applications; 2) the Standard Rights Chamber which is the default chamber for pre-installed applications; and 3) the Least Privileged Chamber (LPC) which is the default chamber for all non-Microsoft applications that are available through the Windows Phone Marketplace [26].

**Process Capabilities:** Applications in the fourth chamber type have capability requirements, that the developer creates, that are applied at installation and at run-time [16]. Once installed capabilities cannot be elevated at run time [16].

**Data Confinement:** The security model has four different types of virtual chambers, each of which has different privileges, to isolate the applications such that they cannot access memory used or data stored by other applications. Each chamber has its own policy and the application in it can be further configured to isolate them from each other within the chamber [26].

Every application is granted a basic set of permissions are granted to all applications, including access to an isolated storage file [16]. All applications run in a least-privileged chamber created specifically for the application, and controlled by a policy system that assigns capabilities based on what the application needs. Each application gets what it needs, and when application run they are strictly isolated from each other [16].

**Installer:** Developers use Microsoft.NET tools in accordance with specified standard practices. All applications undergo certification tests by Microsoft and are code-signed and made available through the Windows Phone Marketplace Hub. The Windows Internet Explorer Mobile browser cannot install programs or plugins from other websites, which greatly reduces potential exposure to malware [16]. All applications installed from theMarketplace Hub, into least privileges access rights which can be expanded using capabilities [26].

The above information is summarized in Table 3.1

Table 3.1: Comparison of different smartphones

| | *Android* | *iOS* | *Symbian* | *Windows Phone 7* |
|---|---|---|---|---|
| **Trusted Computing Base** | Linux kernel and core libraries | XNU kernel and core libraries | kernel, the filesystem, and software installer | kernel and kernel-mode drivers |
| **Install time permissions** | Signature checking and user approval | Only Apple signed applications | Only Symbian signed applications | Only Windows from theMarketplace Hub |
| **Runtime permissions** | Cannot request more permissions at runtime | User prompted for more permissions at runtime | User can grant permissions only during installation | Capabilities cannot be elevated at run time |
| **Core Applications** | Run with root privileges | Run with root privileges | Have manufacturer granted access to TCB and DRM | stored in Standard Rights Chamber |
| **User processes** | Component level MAC | MAC, fixed user space access profiles | Permissions granted to APIs | Fixed permission sets |
| **Data Confinement** | Sandboxed, runs in their own memory | Sandboxed | Applications have their own private directories | Stored and run strictly isolated from each other |
| **File Sharing** | Can make components visible externally | none | Can access open marked directories | No communication other than the cloud |

# 4. SECURITY SENSITIVE SMARTPHONE APPLICATIONS

The smartphone is already hosting a multitude of applications, and the security concern with each next generation application is growing. This is because every next application has improves on the smartphone's unique functionalities, but also increases the risk with information passing and storage.

In this chapter we look at two such applications, which are bound to grow even more in near future, but which raise the security sensitivity of the smartphone to a new level. Since the focus of this thesis is to propose very high and even stringent security solutions, this chapter discusses yet another reason why strict security is required on smartphone.

## 4.1 Smartphone as a Server

Using the smartphone as a server for personal usage is a natural direction for the the smartphone, even more so than using it as a workstation. With the smartphone increasing in computational complexity, as well as housing

### 4.1.1 Secure Shell (SSH) Protocol

Several how-to tutorials on the Internet, guide users to connect to their smartphone device from another computer using the SSH protocol. A key reason for doing so is that especially once jailbroken, this allows the user to use a big screen and real keyboard to work on the device, managing files or using terminal that Linux based smartphones have.

In installing a SSH server-side application on to a smartphone (which is currenlty only possible on jailbroken phones), the user has opened the device to receive a connection request from any other device on the internet which has knowledge of its IP address. Often these SSH applications that are hacked to be installed on a smartphone, install with default login key and security setting, common across all installations. A common user, who is may even have used SSH before, is often familiar with the client-side application, and does not realize the added hazard of installing a server side application with minimum protection. Though there are instructions available on how to create private keyfiles, a user who may only be interested in file transfer, and is downloading the application via an installation package, would find it difficult to change security settings via a Unix terminal as in most of these applications. On the other hand, any can broadcast ping to devices on a subnet, searching for smartphones and finding their IP addresses.

Moreover, SSH application runs in the background, sometimes without notifying or warning the smartphone user that the application is running, or has even established a connection.

Hence, installation currently SSH server-side violates the policy of introducing a vulnerability to attacks via the network, without warning to the user. By installing an application not tailored for a smartphone, but rather an actual server, the smartphone is exposed to severe security and integrity issues, and hence it is a case of incorrect implementation of a solution to a functionality sought by smartphone users.

### 4.1.2 Remote Desktop Protocol (RDP)

Third party RDP server-side applications are available for the iPhone devices. RDP has the advantage over SSH to be able to provide a graphical user interface to the iPhone. User can view their device screen, touch controls and even push the lock and menu buttons from another computer. However once again there are no passwords or toggle for the server, risking unauthorized connections. This is a similar policy

violation of opening a severe vulnerability as a side effect to a sought functionality. Currently there are Remote Desktop Client available for iOS and Android that enables the user to connect to their Windows computer across the Internet from the mobile smartphone devices.

## 4.2 Smartphone as an E-Wallet

The concept of e-wallet or digital wallet on the smartphone, is to further take advantage of the people always carrying their smartphones on them. By serving as an e-wallet, the smartphone could carry quick and secure electronic commerce transactions. This would combine the wallet functionality to the already merger of cell phone and a lightweight PC functionalities.

### 4.2.1 On-Device Banking Application

A technology called Near-Field Communication (NFC) which allows any enabled device to access the cash register through a secure radio frequency. It is a short ranged communication technology for exchanging digital content. Of most interest to us, a smartphone with an NFC chip could make a credit card payment,serve as keycard or ID card. When a phone is enabled with near-field communication technology, shoppers can load bank and credit card information onto their phones and then scan them to buy goods at the grocery store, gas station, subway or any other place set up to read the device.

Currently, Google has already announced to be using NFC as a mobile payment system on Android [27]. An Android device with NFC hardware will typically act as an initiator and will actively look for NFC tags and start activities to handle them [28].

### 4.2.2 E-ID

NFC is also useful in simplified transactions, data exchange, and connecting electronic devices with a touch. As promoted [29] NFC devices can read NFC tags on a museum or retail display; and can pair with Bluetooth and share contacts, photos, songs, applications or videos.

However, the same concept of quick short range communication can be expanded to support functionalities such as e-ID, e-passport, etc. With more complex cryptographic authentication, and safe storage, smartphones could include scannable identification information. Eye scans and fingerprints can make phone IDs more secure. The ID technology might can work as identification papers or security badges, pulls up personal information when scanned.

### 4.2.3 Remote Lock

Already cars (such as in the case of the 2011 Chevrolet Cruze) have started featuring with a smartphone enabled remote lock that allows the owner to check the fuel gauge, lock and unlock the car, set off the horn and lights alarm and perform onboard diagnostics, such as checking tire pressure, by remote. To do so, the manufacturers installs a 13.56MHz, cryptography-protected NCF2970 chip while building the car. [30] This enables car owners to check status and run diagnostics, once again via NFC, from their smartphones. Also, of great utility is that the smartphone is now a location aware set of keys. It can records the car's last parked position, and display it on a map application for the owner.

Stepping back, and looking from a security perspective, it is easy to see why it is now even more critical to protect the smartphone when it enables remote access to a vehicle. Moreover, a Swedish company is already user testing NFC-enabled phones as hotel keys [27].

### 4.2.4 E-Cash

E-cash is an cryptographic application and was introduced by David Chaum [31] as an anonymous electronic cash system. The system uses blind signatures to decouple transactions from the spender/withdrawer [32]. It is based on RSA blind signatures and fulfils all condition of a physical monetary cash. However, the concept has not entirely been adopted for the average users. This is why, given that the smartphone can so takeover all uses of the wallet, it will make e-cash more widely used. This would bring up the need for secure storage and method to carry out cryptographic operations securely on the device. There are already applications for storing credit card information under 256-bit AES encryption.

# 5. SMARTPHONE THREATS AND VULNERABILITIES

The Smartphone is vulnerable to all kinds of attacks that a desktop is vulnerable to and more. It uses several off-device communication channels, has always on connectivity, and is used in as data synchronization and transfer. Each of these channels pose as an attack vector, and they are shown in Figure 5.1. In this chapter we categorize all threat and vulnerability in classes. We base our categorization on the knowledge we have of security malfunction in current smartphones.


Fig. 5.1.: Threat channels

### 5.1 Wide Attack Surface

An attack surface is the a general term referring to user inputs, protocols, interfaces and services, which altogether comprise the different vectors through which a device can be threatened. Attack surface be reduced by turning off unnecessary functionality, having less code available to unauthenticated users, reduce entry points available to un-trusted users, reduce privilege levels as much as possible, eliminate services requested by relatively few users, reducing the amount of running code, reducing access to entry points by un-trusted users, reducing privilege to limit damage potential, looking out for anonymous code paths and watching out for code being open to less secure protocols (many applications that use TCP also listen for UDP traffic) [33] [34].

For instance in most cases of smartphones seen so far, applications are sandboxed and in some cases, installation of third party software is completely disallowed. Also iPhone mobile safari browser is disallowed to have plug-in like Flash, the ability to download certain types of file, and the ability to access the file system from the browser. On the other hand Android does allow Flash plugin, and a recent Flash vulnerability if exploited on Android can lead to system crashes and code execution [35].

Furthermore, platforms such as Android and iOS are borrowed from Linux foundations, and hence one major flaw when building a smartphone is to reuse an operating system from a desktop platform onto the smartphone platform. Though the two systems may seem to have similar functionalities, just but reducing a much larger operating system leaves open redundant and possibly harmful functionalities. For instance ' setreuid' and 'setreguid' previously existed but was later removed from the iPhone OSX kernel to prevent processes from even requesting to change user and group IDs [23].

In addition, whenever the smartphone is open to an external communication channel, the more vulnerable it is to attacks due to discovering addresses during commu-

nication, compromise of link key database, software errors and incorrect protocol handling [36]. For instance, iPhone uses a proprietary mechanism called XYMPKI based off Push IMPA to authenticate to server for Yahoo mail accounts. It was discovered that this custom protocol infact led to a man-in-the middle/replay attack since it did not support Transport Layer Security (TLS) [37]

**Threat 1** *The larger the attack surface for a smartphone, the more likely that it will be exploited.*

## 5.2 Disclosure of Information

Smartphones are used for storing and publishing a significant amount of personal information. Information may be contained in secure storage if they are of the form of passwords, financial information or security keys. However, information often exists close to the attack surface of the smartphone and can be easily exposed. In some cases even if measures are taken to minimize the amount and type of personal information disclosed through a smartphone, some information is always leaked which the user will not be aware of by use of simple functionalities. For instance, most smartphones transmit the unique device ID to outside of the device whenever a new application is installed [38]. A survey by the Wall Street Journal [38] showed that the top 101 most popular smartphone third party applications transmit private data such as current location, phone numbers, owner's name, and device IDs to outsiders without owner?s consent, hence compromising privacy and anonymity. Although, manufacturers such as Apple and Google require that third party applications obtain users' permissions before transmitting any data, often such a policy is not enforced on the devices, and several applications violate this rule.

Information can be disclosed via third party applications, which should have had passed better scrutiny before installation. As seen in the case of the Symbian smartphone private data such as music and documents are accessible by applications of any trust level. In case of the Android smartphone, self-signed applications can be

installed which can cause unaware users installing less tested applications on their devices.

Information can be disclosed via peripheral devices such as connection to a workstation or connection to a bluetooth device. Any channel transmitting information outside the device via an insecure link, and without encrypting the data is open to data interception. At the same time, information stored externally/off the device, whether in a workstation backup or on the cloud, without encryption or sufficient authorization mechanism can be read via unauthorized parties.

Information can be disclosed if the components of the device computing environment do not correctly read, write and execute only on permitted levels. Hence if data is shared between applications of different trust levels, or if an top application level entity such as a browser has direct kernel mode access, then it would compromise confidentiality and integrity.

**Threat 2** *With the growing amount of sensitive data being stored on a smartphone, and the corresponding lack of security in storage, communication and application installation, the smartphone is open to data theft.*

### 5.3 Deception with False Data

For each security mechanism in place on a device, there is the threat of the mechanism being fooled with false data. The biggest threat for spoofing with false data, is to override/trick security checks. In current times, just as hackers have violated device policies by downgrading the operating system, they have also been successful in by passing SIM authorization leading to device 'unlock' and in installing applications without proper signature.

As an instance of unauthorized OS downgrade, an Apple security mechanism validates and verifies every device OS restore via a challenge/response protocol. A 'partial digest' of the firmware files are sent to the server which verifies and signs it of. This way the device is protected from being booted with custom/modified firmware,

as well as old firmwares that have uncovered security vulnerabilities. However, any such verification model is subject to a simple replay attack, where one just stores a copy of Apple's sign off and then returns it at a later point.

Some phones are locked to a particular carrier's SIM card. This is done via a code on the SIM that corresponds to the users account in the mobile carrier's database, three set digits of which is the Mobile Network Code (MNC) number identifying the carrier. When the phone baseband is loaded in memory, one of the things it verifies is the MNC against its network lock state, before activating the cell radio. For a smartphone with a more protected verification code, a simpler way is to 'unlock' the carrier is with a simple hardware clip that fits between the SIM chip and the reader. This clip spoofs information to bypass the authorization check.

All smartphone platforms have an official manufacturer controlled code signing and certification mechanism. Some have a tightly confined system of only allowing signed code from select authorities. However, several means of bypassing code-signing through pseudo-certification, unauthorized self-signing, and bypassing certification checks are already well employed. iPhone hacker Jay Freeman, has published instructions on his blog [39] on obtaining self-signed developer certificates and generating SHA1 hashes that are checked by the kernel. Both of which are examples of deception with false data.

**Threat 3** *Security mechanisms are easier to bypass if authentication and Verification is not done more rigourously.*

## 5.4 Disruption of Correct Operation

A frequent hack, especially for the purpose of jailbreaking (see Section 7.1) is to place the device in firmware upgrade mode and to then skip code verification during the bootloading process to mount an unsigned kernel.

On start-up, the iPhone calls upon three stages of boot-loading, which is a boot-strapping process that starts the iOS. The first stage boot-loader is called the BootROM

or SecureROM. It calls the Low Level Bootloader (LLB) that runs several setup routines and checks the signature of the third stage bootloader or iBoot before calling it [40, LLB]. This brings up the Recovery/Restore Mode, which is run from Apple RAM-disk during a restore or update. It has an interactive interface which can be used over Universal Serial Bus (USB) or serial to connect with the Apple iTunes interface which can re-flash the device with a new OS [40, Recovery Mode]. A critical jailbreak is that while executing the LLB bootloader, signature checking fails causing the device to stay in Device Firmware Update (DFU) mode. The DFU mode unlike Restore/Recover Mode bypasses the current installed operating system and allows the device to be upgraded or downgraded [40, DFU Mode].

**Threat 4** *Malicious parties can seize control by disrupting the secure flow of operations.*

### 5.5 Unauthorized Control of Part of the Device

User space hacks can often lead to an application gaining access of the filesystem in part of whole, which it was previously not authorized to. Once again, taking the example of the iphone, user space exploits have been executed to gain access to the entire filesystem, thus overthrowing mechanisms such as data confinement. In the case of the iPhone, hackers having gained access to a kernel file called fstab, can modify it to mount the System partition as read-write [40, Restore Mode]. The fstab (/etc/fstab) (or file systems table) file is a system configuration file commonly found on Unix systems. The fstab file typically lists all available disks and disk partitions, and indicates how they are to be initialized or otherwise integrated into the overall system's file system [41].

In cases where rootkits are installed, malicious parties can take over kernel functions. Bickford, et. al. [42] demonstrated how rootkits can enable tracking the user over GPS, hearing into private conversations, stealthily switching on bluetooth and draining battery on an OpenMoko phone.

As we attempted by ourselves, server side SSH installation with known login that can allows an unauthorized user to remotely launch applications on a jailbroken iPhone.

**Threat 5** *If the system is partitioned into different security levels, a vulnerability in any one of them can allow for access and control of all code running at that level.*

### 5.6 Threats to the Cellular Network

Smartphones are connected both with the Internet and the cellular network. This makes it possible to move botnets as data packages which over the internet or over SMS. Such malware propagation would be similar to the case of viruses and trojan for workstations over the internet. Attackers can use Smartphone to compromise other devices, computers, or networks by running network or port scanners, SMS/MMS/email worms, as well as compromise internal/protected network. Though in the case of the data network we can have secured/firewalled enterprise networks monitoring network activity, as well as protected data channels; such mechanisms are missing in the cellular network. Also, core network targets, such as DNS in case of the internet, are often hit with denial of service attacks. However as noted in [43], even though the internet is more resilient to such attacks, cellular networks have more rigid hierarchical dependencies and are more vulnerable.

Traynor, et al in [43] point out that a relatively small number cellular botnets can collapse a targeted cellular core network. They theoretically measure the potential impact of a hypothetical botnet through a combination of measurement, simulation and analysis showing that a botnets infecting 11,750 compromised mobile phones can degrade service to area-code sized regions by 93%. They simulated denial-or-service attacks, on network bottleneck regions. The attackers can evaded users' attention by make service requests instead of active calls, for flooding network centerpoints.

Additionally, Mulliner, et al in [44] describe the architecture and even implement a cellular botnet for the iPhone. They tried a combination of scenarios for communi-

cation from the botmaster to the botnet: over SMS only and over SMS and IP. They thus show that with the presence of internet connectivity botnets can be more advance transporting larger data to and from device, stealing information or upgrading the botnet. This makes smartphone botnets more dangerous than those on feature phones.

**Threat 6** *The smartphone is vulnerable to being highjacked via cellular network botnets.*

## 5.7 Threat from Very Restrictive Application Marketplace

Each smartphone vendor have a different mechanism of allowing application distribution and installation. It ranges from a very restricted marketplace to a no restriction on application installation at all. With full control of third party software installation, as with the iPhone, the vendor can closely check for quality and security adherence, enforce code signing, and revoke applications if they turn out to be malicious. With a totally free market place, as with the Android, the vendor sees much higher growth in number and variety of applications being developed for their platform. As an in-between mechanism, the vendor can delegate security guardian responsibilities to another entity, such as mobile phone carrier or enterprise system administrator [45] which then maintain a blacklist of revoked application.

However, going to either a close monopolistic or unmonitored free market are high threats to smartphones. While platforms with no restriction on application installation, see most number of malware and phising attacks, being a closed market as seen more hazardous attacks. In case of the iPhone, restrictions on application installation, lead to many user 'jailbreaking' their iPhones. Apple's policy on adopting applications for the official AppStore is based on two criteria: functional restrictions and content restrictions [46]. Functional restriction are where concrete security measures should be applied. For instance one functional restriction is not allowing applications that synchronize data using the wifi protocol [46]. Another restriction on earlier iPhones

were to not use the camera to record video. This lead to an unauthorized third party application which could record videos on jailbroken phones. Content restrictions are even more discretionary and may not always be appreciated by the users. One famous case is Apples rejection in 2009 of the app NewsToons by the cartoonist Mark Fiore, which criticizes the White House. But the application was accepted in 2010 only after the author won the Pulitzer Prize [46]. There are other forms of checking such as user interface aesthetics and checks for security violation, but no further details are provided on the actual approval mechanism. Popular iPhone hacker Jay Freeman estimates that more than 10% of all iPhones are jailbroken [39].

**Threat 7** *By now allowing certain amount of flexibility in application development and installation, users may instead willing bypass security mechanisms introducing more vulnerabilities.*

### 5.8 Application Level Malware

For security mechanisms to work, it should be a viable assumption that the security services can rely on the kernel to supply correct data. Malware can find entry to a device via social-engineer techniques or by communication vectors on the smpartphone. Once there, a malware can remotely control a device, modify the filesystem, expose confined data, install rootkit, and breach the device security policy in general.

After this, permission escalation is to maliciously using the permissions granted to an installed application. One attack that effected several smartphone was the Webkit web browser attack, which was done through a buffer overflow in an outdated native library and a cross-site scripting vulnerability, which allowed the hacker to run malicious code on the device using the browsers high privileges. This was able to be done and lead to a userspace jailbreak (complete control of the filesystem) because the browser application was running at the kernel trust level.

**Threat 8** *The smartphone is vulnerable to application layer malwares*

### 5.9 Kernel Level Malware - Rootkits

Kernel level malware exploit a vulnerability in the operating system kernel or system libraries. Rootkits can do malicious activities such as stealthily placing a call, listening into confidential conversation, read and send location data, and drain resouces such as battery [42]. At the microkernel level, the rootkit gains full access of the target system, by being able to inspect all communication between the operating system and the hardware, as well as evade detection with this more control [47].

**Threat 9** *The smartphone is vulnerable to rootkits*

### 5.10 Insecure Data Transfer

Malware infections can spread from a smartphone to other devices that it is connected to via peripherial links and vice versa, showing a crossing-over behavior. In 2005 Cardtrap.A was a Symbian SIS file Trojan not only disabled application on the cell phones, it also installed three Windows worm on the device's memory card which would move to the workstation once the card in inserted in it [36]. In 2006 Crossover virus moved from the Windows workstations to Windows Mobile Pocket PC [36]. Similarly [48] showed three attacks via the USB link in the three areas of device to computer, computer to device and device to device. Such attacks were possible because the syncing, backup and removable media mechanisms are not well secured in most smartphones.

**Threat 10** *Without proper security data being moved outside the device can lead to both contamination/unauthorized modification of data, as well as loss of sensitive information.*

# 6. EXISTING MECHANISMS

In Chapter 5, we presented a discussion on common smartphone threats and vulnerabilities. In this chapter we present related works done on countering some of the mentioned threats. We discuss existing mechanisms in the broad catergories of ensuring kernel integrity, preventing data leak, and detecting and preventing malware. We note that these solutions are ad-ons, often like security enabling applications, to the existing smartphone platform. In later chapter, we will present out solution which is a redefinition of the smartphone security platform itself.

## 6.1 Ensuring Kernel Integrity

The NSA created Security-Enhanced Linux (SELinux) by implementing Mandatory Access Control (MAC) on Linux. MAC access decisions are based on labels which are enforced over all subjects (processes) and objects (e.g. files, sockets, network interfaces) in the system. MAC can support a wide variety of categories of users on a system, and it can contain the damage that can be caused by flawed or malicious software.

The iPhone permission system is based on the TrustedBSD framework.Following the Trusted BSD framework, to grant process permissions require writing policy files that describe what permissions an application should have. Users can create new policies to sandbox applications on their system. [23]

Similar to it since smartphone operating systems such as the OSx, iOS and Android are subsystems of the Linux kernels, it is possible to apply SELinux to the these them. SELinux can limit the abilities of root processes and otherwise potentially vulnerable or high-priority entities, so that even if they are compromised the attack is cause less damage. The SELinux policy can support separation policies that

can enforce restrictions on data, establish well defined user roles, or restrict access to classified data, containment policies useful for such things as restricting web server access to only authorized data and minimizing damage caused by viruses and other malicious code, integrity policies that are capable of protecting unauthorized modifications to data and applications, and invocation policies that can guarantee data is processed as required.

In [49] the authors claimed that Linux Security Moducles based approaches such as SELinux as it focuses on enforcing least privilege, and its policies on personal computer systems are too complex to understand integrity completely.Hence, they developed a and modified and compacted SELinux policy to build a high integrity phone system, and later tested the integrity of a phone system using the Policy Reduced Integrity Measurement Architecture (PRIMA) approach.

## 6.2 Preventing Data Leaks

In order to secure the operating system on a smartphone, there needs to be a custom access control policy. Unix systems use Discretionary Access Control (DAC) mechanisms, where any program executed by the user inherits all of the privileges associated with that user. Only two categories of subjects are supported: the administrator and normal user. In all this is a coarse-grained policy, and any malicious program which has obtained super user privileges can change permission associated with all object and disclose them for other programs. DAC access decisions are only based on user identity and ownership, ignoring other security-relevant information such as the role of the user, the function and trustworthiness of the program, and the sensitivity and integrity of the data. Also such a policy is not mindful of the fact that all application may need to be classified into more than just two security levels depending on their source, signing authority, and objects such as processes, sockets, files and other resources that they need.

Data leaks can be internal as well as to the outside of the device. Some of the most sensitive as well as commonly requested permissions is to use geolocation features or read data from the camera. Such permissions should be discerenly granted once again depending to the overall security required by the user as well as the source of the application code. GPS, camera and audio should not be accessible directly without using a trusted API created to transfer information.

The likelihood of privilege escalation can be reduced by means of a memory management unit (as done in Android), which sequesters processes in memory space, so that a process is unable make its own code run in a privileged mode by means of overwriting the private OS memory (protect integrity) [17]. Also this way a process is unable to read the memory pages of another process (protect confidentiality/ information disclosure), or flood its memory (protect availability).

Sensitive private data (such as messages, contacts, emails, notes, audio/video, images and documented) can be encrypted using a user input key so that the information is secure even if an attacker steals the device and has full access to it. This way by not having the device alone encrypt such data, the key is stored not on the device itself, but is with the user of the device.

On the Windows phone there are no communication channels between applications on the phone other than through the cloud [16]. Though this may be effective, its is a highly restrictive security measure.

To prevent data loss in case of phone theft, user authentication and authorization can be done by the SIM containing a secret known by the card and the operator.

Data elements type that are passed by the Android-specific Inter-Process Communication (IPC) mechanism, is defined by the developer on compile time to ensure that types are preserved across process boundaries [17].

Resource management consists of fairly allocating resources to applications according to their needs and importance (for example, the phone application is very important and should thus receive more CPU than a game). In this case, unsupervised resource drainage is not possible. If a resource management solution maintains

disk storage quotas and disk and network I/O are rate limited and permitted up to a certain quota, then it can fully mitigate a DoS attack [18].

## 6.3 Preventing and Detecting Malware

Even for smartphones that are Linux based, off the shelf security from malware cannot be utilized most times owing to issues with dependencies, size and resource usage. Even then anti-malware tools are being developed. For instance, the authors is [20] statically compiled an anti-virus open source unix tool called *Clam AntiVirus2* for the Android phone and found that 28MB space required exceeded the 21MB system space, and the virus checking database had to placed in a different location. There are several approaches to host based intrusion detection rootkit detection which are effective against the malware threats.

The Linux application level provides all the functionality needed for monitoring and storing device and operating system information. On Java application level, anomaly detection, detection collaboration, and detection response are realized where the corresponding states can be visualized in a user interface. [20]

### Signature Based Malware Detection

Signature-based anti-virus solutions for mobile device are useful for postinfection cleanup, and to match signatures to a malware database they may seek too much of constrainted mobile resources. Also they are prone to malware signature changing by obfuscation, polymorphism and packing techniques.

### Behavior based Malware Detection

It [36] an alternative to signature-based mobile malware detection in form of behavioral detection where the run-time behavior of an application (e.g., file accesses, API calls) is monitored and compared against a set of malicious and/or normal behav-

ior profiles. This involved first reconstructing higher-level behavior signatures online from lower-level system calls and file accesses; second reconstructing these signatures during run-time by monitoring system calls and resource accesses.

### Mobile Trusted Module

The Mobile Trusted Module (MTM) is a security element for use in mobile and embedded devices [50]. A hardware based counter-measure to both jailbreaking and rootkit would be a Mobile Trusted Module (MTM) [50] [51]. One suggestion is that the MTM takes integrity measurement of software stack as SHA-1 hashes and store it in secure hardware which can be called ready by a verification authority such as a service provider [52].

Kim, et al. [51] show the design and implementation of a Mobile Trusted Module (MTM) is presernted which satisfies small area and low-power condition.

### Coprocessor

A coprocessor based technique is proposed in [52], where the authors suggest deploying malware or virus detection/prevention software on to a workstation that the smartphone often synchronizes its content with using the USB connection. This would mean that the workstation is capable of accessing the phone's filesystem, as well as store hash of all files onto the workstation. On subsequent connections, the phone would compute and send hashes of all files to the computer, which in turn would copy and scan those files whose hash values have changed. Using a keyed hash technique which requires a key from the workstation would prevent any rootkit on the phone from storing hash values of files before modification.

Additionaly, [52] makes the suggestion of detecting rootkit by placing a challenge onto the phone in the form of a complex function call or computation which would give a higher running time on a phone with rootkit. Similarly, a memory challenge

of asking the phone to temporarily copy data equivalent to expected free space could signal if there are any other rootkit code occupying space in memory.

A coprocessor-based strategy, however, places significant trust on an untrusted device i.e. the workstation by giving it access to the filesystem. Smartphone memory access via PCI card [53] and Firewire has been suggested. A coprocessor is thus a kernel integrity monitor that does not rely on the kernel for access to main memory and requires no modifications to the protected host's software.

# 7. SMARTPHONE HACKS, ATTACKS AND JAILBREAKING

All code has a nonzero probability of containing vulnerabilities and although minimizing threats and patching known vulnerabilities prevent security failures, it does not mitigate the amount of damage an attacker could inflict once a vulnerability is found. In this chapter, we duplicated a couple of simple but severe attacks, which help us understand common security negligence which can severely compromise the security framework of the smartphone.

## 7.1 Jailbreaking

Jailbreaking is the process of removing restrictions on application installation and directory access, set by the manufacturer on the device. It is also a precedent step for SIM-unlocking, the act using the device with any carrier SIM card. For this reason, even phones with no restriction on application installations maybe jailbroken if they is not flexibility in choice of service provider when purchasing the device. Jailbreaking, opens the device to uncertified market place where application developer enthusiasts can create functional applications that the manufacturer is not providing, even though these application may have severe vulnerabilities. More importantly, jailbreaking also disables all on-device security mechanisms and leave the device open for attack and installation of unverified code. Jailbreaking is most popular with the iPhone platform, and so in this section we look at how jailbreaking (which in a way is user consented attack to the device, without sufficient premeditation from a security viewpoint) has been carrier out on the iPhone.

For security mechanisms to work, it should be a viable assumption that the security services can rely on the kernel to supply correct data. The iPhone uses a

signed kernel to prevent tampering, however all iPhone kernel version till date have been exploited via different security flaws [54]. The hacking community surrounding smartphones has devised several tools and techniques to assist with the process of jailbreaking the device to allow installation of third party software and unlocking to use an unapproved carrier SIM card, dating back to 2007 when the iPhone was first released closed to to third party application installations [54].

Table 7.1 is a list of a few critical jailbreaks on the iPhone, alongside the exploit and the vulnerability. The data is taken from what is shared with public by the informal jailbreaking hacker community behind the iPhone.

## 7.2 Data Exported to Insecure Devices

The smartphone has to be often synced to backup personal information and application files. This is important in case the device data on device is lost. Often a software is trusted to sync running on a personal workstation, eg. for iPhone it is iTune and for Windows Phone it is Microsoft Zune. One reason for having a specific software to sync the device is to automate quick backups as the software detects and backups data from set file partitions. The off-device software is also used to detect operating system version and administer OS upgrades. Another utility could have been to secure data transfer with more security, though as from the first userspace attack in Table 7.1 we know this is not the case.

An important consideration to be made here, is that even though security mechanisms are added to data transfer and storage; a syncing-software can not be considered trusted since it resides on an untrusted workstation. Any malicious software can mimic the protocol even if security authorizations were present, since the workstation itself can be malicious. Hence not all files can be allowed to leave the device in raw form. Also, storage on an untrusted device is inherently untrusted. Even after encrypted, if the encryption keys are stored with the file, then it is a security flaw.

Forensic analyst, Jonathan Zdziarski was the first to demonstrate in 2008, a low level but shocking experiment behind iPhone backups, reveling that they are infact

Table 7.1: List of important jailbreaks

| **Name** | **Exploit** | **Vulnerability** |
|---|---|---|
| UserSpace Exploit | Modified the Apple File Connection (AFC)service which syncs with iTunes, by adding a service that runs as root | The (AFC) lied in the non-OS partition |
| UserSpace Exploit | Used the command cp iBoot to copy the fstab(unix files listing disk partitions) to userspace | Redundant functionality: the command cp iBoot had filesystem access |
| Pwnage | In the bootloading, though iBoot signature checked the kernel,Low Level Bootloader (LLB) did not check iBoot and bootrom did not check LLB | bootrom did not signature check LLB |
| ARM7 Go | Two iBoot debug commands left behind: arm7 stop and arm7 go, which could have the coprocessor execute code | allows unsigned ARM7 code on coprocessor partition |
| JailbreakMe | buffer overflow in libtiff file modifies kernel space data to circumvent security checks, calls setuid(0) to get root access | Webkit browser had access to kernel memory with write privilege |

unencrypted. In our attack, we manually retrieved the backup files located on the PC disk. Each application has a folder which is named same as the application ID from Apple Store. We then recovered the data from backedup files by restoring them to their correct formats. We noted that the fact that iTunes sets a password for these backups had no security benefit to prevent uncovering the data outside of iTunes. This experiment infact proved two attack concerns: 1) that we are able to backup information from the smartphone and then extract this information with a second application that was non trusted to the smartphone. This undermines any security

functionality of iTunes. 2) It also showed that a quick backup to a foreign workstation can infact be of the the simplest ways to steal data from the device. This is especially true, given that a few of the files that were found in the backups of even unjailbroken phones were the location database, personal multimedia, application backups and passwords.

### 7.3 Exposing Private Data

It was discovered in 2011 by two software hackers [55] that an unencrypted file resided in Apple iOS called "consolidated.db" containing extensive location information. This is critical not only because it is done unknown to the user, but because it is easily accessible on the iOS userspace directory. The official Apple statement in defense was that Apple as the manufacturer stores information about nearby cell towers and WiFi access points on the device which was later collected by Apple after anonymizing the source from the data. The data gets transferred outside the device without user consent, whenever the device connected to a Wifi network.

In our experiment we were able extract this file both from iPhone backups and by access to an open directory location on the device, accessible to the user by jailbreaking it. It was noted that the log was extensive, and spanned several months. The "consolidated.db" file contained several tables, the most prominent being two tables called Wifi location harvest and Cell location harvest. For both tables, each entries had timestamp, latitude and longitude information besides other fields. For the Wifi location, the table also had a MAC address field for each entry. Further research into older statements from Apple revealed to us, that Apple was developing a 'quasi-GPS' method for location detection called Skyhook [56]. In order to populate Skyhook with location data, users in 2008 were encouraged to submit latitude and longitude information by revealing their addresses as well as the WiFi point associated with the designated location and detected by the smartphone. It then seems, that Apple since

moved to secretly and autonomously make devices collect and transmit this data. However, this scheme has two major security concerns.

First, Wifi location refers to discovery of routers and this is transmitting data not only about the user or device, but about other devices in the environment. Also, cell locations are always known by the base stations and the user is aware of that. Also the service provided is authorized to have cell location, but it is not clear why should the manufacturer should be authorized to know cell location. Hence this is a violation of the principle of least privilege.

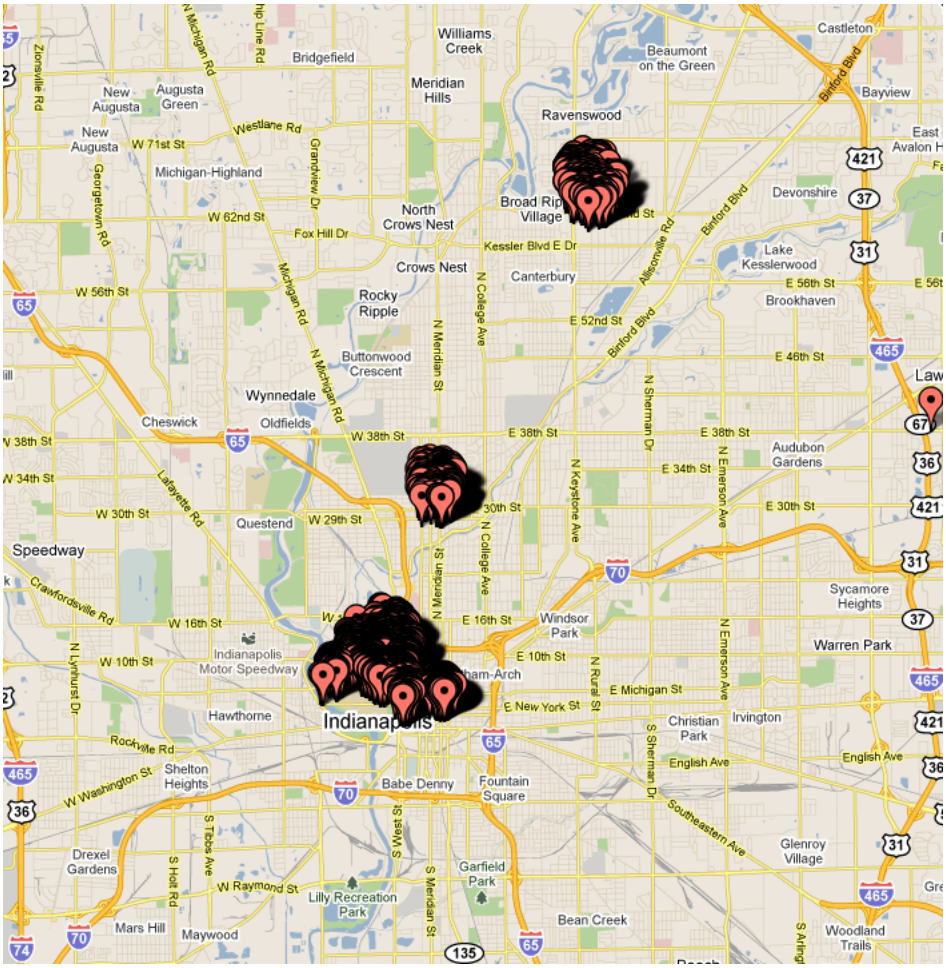


Fig. 7.1.: Location based attack

Second, we plotted Wifi and cell location data from several iPhones to assess how much information is reyvealed. In Figure 7.1 we see the latitude and longitude entries plotted from one of the tables from one of the iPhones. As apparent, this map gives a lot of information about the party. Not only does it present clusters of regions where the user was mostly present, it leaks significant information on the users activities and locations over a some months.

### 7.4 Detection Over Network and SSH Attack

We see that smartphones are easily detected smartphones over a subnet. In our experiement, we sequentially scanned the subnet for smartphones, identifying them by their TTL (Time to Live) value. We knew that jailbroken iPhones get a default root password assigned and often have the secure shell server installed. Hence, after identifying a device, we used its public IP to attempt to create an SSH connection with each device, looking for plausible jailbroken devices with SSH server installed. Once a connection was established, we found that with server access and root control of a smartphone, we could remotely start and exit applications as well as view the entire filesystem. We also noted that smartphones, especially in unmonitored public networks can be attacked via constant pinging. This is a denial of service attack not only because it effects network traffic to the device, but also because it drains its power.

For the network detection attack technique, smartphones are vulnerable to be detected by network malware, which can be in the form of worm and viruses (some recent ones being named Code Red, Nimda, and Slammer). These are characterized as bot networks or topological worms. In [36] it was demonstrated how the Cabir virus can spread among the cellular network via Bluetooth, and how another worm can exploit email and peer-to-peer file sharing to infect the enterprise network. This can happen because, once one device is affected, the malware can spread throughtout the network via 1) proximity scanning for close range wireless channels such as bluetooth

or Infrared; 2) history scanning monitoring phone and sms history; 3) topological scanning by searching address books, URLs, application data cache, etc; 4) sequential subnet scanning.

As for the other half of the attack involving SSH server, we see that if one can also make a connection with a server on the smartphone, we can remotely access its filesystem and control applications. In November 2009 a hacker had exploited this vulnerability to create a worm was named Ikee.A [57] which infected around 21000 iPhones within two weeks by simply copying itself via secure copy (part of ssh)from iPhone to iPhone. Later somebody added a very simple command and control mechanism to Ikee to turn it into a botnet, this botnet was called Ikee.B [57]. The command and control mechanism was simply polling a webserver to download and run a shell script.

# 8. SMARTPHONE SECURITY POLICIES

On any smartphone device the parties each with some guarantee of access are the smartphone owner(s), the person presently in access of the smartphone, application framework developers, third party application developers, operating system developer/community, local subnet (if open to bluetooth or WiFi area), the device manufacturer, network carrier/service provider and the government which owns the network bandwidth.

The types of resources and information stored on a smartphone range from hardware device drivers, system/kernel files, application framework files, permission files, password files, application files and user data. What's open for the users are their own data and executables for installed application. By default the kernel has the highest security level access. Kernel and system files are unaccessible and can be invoked only through the given API, even though they can be directly patched by permission held by the device manufacturer, or reistalled with newer version as provided by the OS provider. Applications are given restricted acess to anything beyond their own data set, though more than one applications can combine to leverage into higher access permissions. The guarantee for the network service provider's SIM recognition is handled by hardware modules, or as in current iPhone version with Verizon, the SIM is eliminated and carrier capabilities are hard coded in the hardware. In terms of priority in terms of amount of control held by each party, the user can be seen to be at the end of the chain with least control or device modification capabilities. This may serve to prevent misuse of device to harm the technology or network, however with least privileges the user can be being put at a disadvantage or even risk.

A smartphone security policy would define whether a smartphone is secure or insecure. A Smartphone should have explicitly stated security requirements, which are statements about what and how should it function. These requirements would be

a collection of well-defined, consistent, and implementable rules that are clearly and unambiguously expressed [15]. A policy set defines rules such as what generic security rule should be invoked by default and what device functionalities are important and should be protected. Also, given that a smartphone has various stakeholders, namely user, manufacturer, cellular service provider, and well as any network that the device is being used in, each of these different viewpoints have different security rights.The purpose for such policies are that if the device implements these requirements then

1. It can be said to have met the security expectations from the view of the different stakeholders.

2. It would have secured itself from being exploited owing to traits such as ubiquity, computing complexity, portability, single user ownership, and connection to data and telecommunication networks.

After having viewed the different smartphone platforms, we now discuss the different policies that are or should be followed under the different components for the smartphone; each having its own set or rules.

## 8.1 Policies for Application Installations

Beyond the static kernel model, at application installation and runtime the kernel as well as installation medium should provide certification as well as enforce them for the applications. Applications should be installed and run under well defined security levels. Since the number of applications on a single smartphone market can go to an amount of hundreds of thousands, certainly not all applications can be treated with same trust level. Hence, there should be multiple trust levels for applications, and information on an applications minimum basic security permission as well as the range up to which its security level can be elevated should be set at the time of installation. Trust level on applications can be differentiated on the grounds of provider, role and installation source.

**K1:** Applications should be granted security permissions based on source, installation mechanism, security level of installing userID, sensitivity of application and its data, amount of resource requested, amount or privileges requested and proximity to the kernel. For this applications should be signed by the developer validating its security and integrity, by the user and by the installer. Also application should then undergo multiple confirmation before they can escalate their security privileges.

**K2:** When the kernel allows an application executable to be run, it must ensure that its certification is valid.

**K3:** For some applications, the kernel must transform them from one security level to another.

**K4:** For every application, an allowed tuple (application, data item) should be tracked. Thus putting it under a security level.

**K5:** The kernel must authenticate every application at run time. This is per application call, not system startup.

**K6:** As suggested in the Clark-Wilson security model, the kernel should append a log of all application calls, of enough information to reconstruct the operation.

**A1: (Application)** Only the certifier of an application may change its policy list.

**A2:** Any application that takes a data item may only perform valid operations as defined by its policy file.

**A3:** Application which provided added functionalities requiring an escalation of trust level should prompt user requesting such permissions at runtime. Also users must have the capability of canceling such previously granted permissions. The ability to grant and cancel up to a level of privacy permission is part of providing the user the flexibility of choosing between utility and privacy.

**A4:** Applications should be given only those privileges that it needs in order to complete a task (least privilege principle).

**A5:** Applications should be explicitly given access to an object, else they should be denied access to that object (fail-safe defaults principle).

**A6:** Application mechanisms used to access resources should not be shared (least common mechanism principle).

**A7:** Application installation mechanisms, between application of different security levels should not be shared.

**A8:** Applications developed by third party should have direct write access to critical files such as location, keychain, contact, etc databases.

**A9:** Critical applications should be protected and should not be able to be called by applications of lesser trust or remotely. For instance, be able to switch telephony on/off remotely.

**A10:** Applications should be packaged with a easy to read privacy policy. The user should be able to have a view of every installed applications privacy policy via a build in viewing application which parses and displays the policy file content. This is because, during its lifetime on a device an application may go several run time permission changes, and the user must be able to view at what access level is the current application running. Also, all applications which are available publicly should have their privacy policy displayed in a form which is graspable to the average user (such as a matrix) on a public location. Such an application policy requirement is similar to the W3C standard on privacy policy for websites.

## 8.2 Policies for Maintaining Kernel Security

**K7:** The device kernel should have multiple signature checking when installing any files to modify the operating system

**K8:** The device kernel security mechanisms should not make the resource more difficult to access than if the security mechanism was not present (psychological acceptability).

**K9:** The device kernel should enforce simple as possible security mechanisms (economy of mechanism).

**K10:** The device kernel should not depend on the secrecy of its design or implementation as a security mechanism (open design).

## 8.3 Policies for Off-Device Communication

For a smartphone the different communication channels are:

1. cell communication part of service provider channel (SIM card)
2. manufacturer channel (net/tethered based)
3. physical channel
4. general communication channel (IR, GPS, 3G/4G (VPN + Service Provider), Wifi, Radio, bluetooth)

**C1 (Communication):** The communication channels should be differentiated between secured and unsecured channels.

**C2:** The communication channels security mechanisms should not make access to valid communication channels or network more difficult to access than if the security mechanism was not present.

**C3:** The communication channel should not establish link based on a single condition.

**C4:** The different communication channels should have separate and independent resources.

**C5:** The communication channels should not authorize transmission of any data on device outside the device without user authorization.

**C6:** Presence of a communication channel should revel no new information about the device. Stated in term of entropy:
H ( device | cell communication, another channel, UserNACK)
= H ( device | cell communication)

**C7:** The communication channels should have security mechanisms for hosting applications such as augmented reality, context aware intelligence and multi-users gaming, without causing privacy leaks regarding the user or the device. Such applications make not just an emerging direction for future of the smartphone, they also require huge overlapping communication among several devices. In order for such applications to take off, there should be a protocol guaranteeing an acceptable level of anonymity as well as flexibility to modify user's permissions regarding them.

## 8.4 Policies for On-Device Information Flow

The device should treat process launched by the following parties in a decreasing level of trust: Kernel (and all OS provider signed patches); core application, virtual machines and software libraries; user installed applications; authorized/trusted online repository(ies); external device connections. Hence no operating system software patch/code that can do kernel level modifications can be allowed to be installed via an external device, since by their nature they have been classified as untrusted. Also, no core applications, virtual machines and software libraries can be be installed via an external device, for the same reason as above. An alternative to this is that all

of the above item should come pre-installed from the manufacturers, or should be available via secure direct download onto the device.

**K11: (Kernel)** The device kernel should have secure encrypted storage for high privacy user data. Such storage should not only be unreachable by any unauthorized on-device application, external device or been transmitted. Any such implementation of such a storage is needed for smartphone applications such as digital wallet, passport or ID.

**K12:** The device kernel should have separate storage and handling of data that is locked to the device, and that which can be accessed or synchronized with external devices or the cloud.

**K13:** The device should not be able to update any system file via a medium whose trust level is lesser than the kernel itself.

**K14:** The device kernel should not grant permission based on a single condition.

**K15:** The device kernel must be able to enforce the separation of information based on confidentiality and integrity requirements to provide system security.

**K16:** The device kernel to check all access to objects to ensure that they are allowed (complete mediation).

# 9. FORMAL SECURITY MODEL AND MECHANISMS

A formal policy model is a highly abstract set of rules and settings which allows us to both define and state rules to maintain a secure smartphone environment. As seen in Chapter 2, the smartphone, and all its components, needs to be secured from confidentiality, integrity, availability and privacy viewpoints, as well as all the information flow on the device needs to be regulated. In this chapter, we make use of lessons learnt from Chapters 4, 5 and 6, to construct a mathematical model which builds upon the basic confidentiality and integrity models from Chapter 2, to a customized and comprehensive security model for the smartphone.

## 9.1 Formal Definitions

First, we distinguish all resources and participants as *subjects* and/or *objects*, denoted by $S$ and $O$ respectively. Where in,

$$S = \{\text{set of all } \textit{subjects}\} \text{ and}$$
$$O = \{\text{set of all } \textit{objects}\}.$$

*Subjects* can be an entity which makes (resource) requests,it can be a controlling party such as user, manufacturer, service provider, etc. or an active process such as telephony, SMS, GPS, browser, camera, e-wallet, microphone, near-field detection, etc.

*Objects* can be any passive (requested) resource such as log files, memory, control switches, etc. and user data such as images, memos, contact book, audio/video, etc. with one or more copies as well as a multithreaded or unthreaded subject that is requested, for example a process that is invoked by another process. Thus since $S \subset O$, we use the term "object" to reference to both subjects and/or objects that are trying to be accessed.

To each object we then assign confidentiality, integrity and availability levels for each $s \in S$.

$$\text{let all } \textit{Confidentiality Levels} \text{ be represented by } \mathsf{C} \in \mathbb{Z}^+$$
$$\text{let all } \textit{Integrity Levels} \text{ be represented by } \mathsf{I} \in \mathbb{Z}^+$$
$$\text{let all } \textit{Availability Levels} \text{ be represented by } \mathsf{A} \in \mathbb{Z}^+.$$

For each object $x \in S \cup O$

$$con(x) = ((\mathsf{C} \times \mathcal{P}(S)) \times (\mathsf{C} \times \mathcal{P}(S)) \times \cdots \times (\mathsf{C} \times \mathcal{P}(S))).$$
$$int(x) = ((\mathsf{I} \times \mathcal{P}(S)) \times (\mathsf{I} \times \mathcal{P}(S)) \times \cdots \times (\mathsf{I} \times \mathcal{P}(S))).$$
$$avl(x) = ((\mathsf{A} \times \mathcal{P}(S)) \times (\mathsf{A} \times \mathcal{P}(S)) \times \cdots \times (\mathsf{A} \times \mathcal{P}(S))).$$

where $con(x)$, $int(x)$ and $avl(x)$ are the confidentiality, integrity and availability values for $x$. In particular,

$$con(x) = ((c_1, \mathcal{C}_1), (c_2, \mathcal{C}_2), ..., (c_{n_1}, \mathcal{C}_{n_1}))$$
$$int(x) = ((i_1, \mathcal{I}_1), (i_2, \mathcal{I}_2), ..., (i_{n_2}, \mathcal{I}_{n_2}))$$
$$avl(x) = ((a_1, \mathcal{A}_1), (a_2, \mathcal{A}_2), ..., (a_{n_3}, \mathcal{A}_{n_3}))$$

where the $\mathcal{C}_i$ are pairwise disjoint subsets of $S \cup O$ whose union is $S \cup O$. Similarly $\mathcal{I}_i$ and $\mathcal{A}_j$ are pairwise disjoint subsets of $S \cup O$ whose union is $S \cup O$. As an example let $browser \in O \cup S$ and $user$ , third part application (tpa) $\in S$, using these entities a mock security levels would be:

$$con(browser) = ((\text{LUKE}, user, tpa), (\text{NIL}, other))$$
$$int(browser) = ((\text{HIGH}, user), (\text{LOW}, tpa), (\text{NIL}, other))$$
$$avl(browser) = ((\text{HIGH}, user), (\text{LUKE}, tpa), (\text{NIL}, other))$$

We define function $\mathcal{F}$, such that for all $s \in S$, $x \in S \cup O$

$$\mathcal{F}(con(x), s) = t \text{ where } (t, s) \in con(x)$$
$$\mathcal{F}(int(x), s) = u \text{ where } (u, s) \in int(x)$$
$$\mathcal{F}(avl(x), s) = v \text{ where } (v, s) \in avl(x)$$

For example, we use the case of the browser object:

$$\mathcal{F}(con(browser), tpa) = \text{LUKE}$$
$$\mathcal{F}(int(browser), tpa) = \text{LOW}$$
$$\mathcal{F}(avl(browser), user) = \text{HIGH}$$

For any object, prior to being allowed access by a subject, the security monitor mediates on the current security level of the object. three matrices have to be decided and assigned to them by the security monitor. When a object is currently being used by one or more processes, then each usage by subject $S_j$ will have an associated confidentiality level $c_j$, integrity level $i_j$, and availability level $a_j$. If there are $k$ active instances of object $x$, then the current use of $x$, denoted by $cur(x)$, is a $k$-tuple where each "coordinate of the $k$-tuple" is itself a 4-tuple. Thus $cur(x)$ is

$$cur(x) = ((c_1, i_1, a_1, S_1), (c_2, i_2, a_2, S_2), ..., (c_k, i_k, a_k, S_k)).$$

As an example, suppose the browser is being currently run by a third party application, thus

$$cur(browser) = ((\text{LUKE}, \text{LOW}, \text{LUKE}, user)).$$

Now assume that the user requests for access, and owing to higher availability metric, is given access. In the case where the user subject is given access besides the active tpa, the user entry is (appended) to $cur(browser)$:

$$cur(browser) = ((\text{LUKE}, \text{LOW}, \text{LUKE}, tpa), (\text{LUKE}, \text{HIGH}, \text{HIGH}, user)).$$

Else if user is given access over tpa, and the tpa access is terminated, $cur(browser)$ will change to

$$cur(browser) = ((\text{LUKE}, \text{HIGH}, \text{HIGH}, user)).$$

Lastly, each entity also has a property file associated with it, which is created during installation, and modified at each access. We denote this file by $prop(x)$ for each $x \in S \cup O$. The content of $propx$ are given in Tables 9.1 and 9.2.

Table 9.1: Property list for subjects

| **Subjects** | |
|---|---|
| signature | $Sig$ |
| Signer | SIGNER_PK |
| multi-thread | (yes/no) |
| thumbprint of $x$ | $\mathcal{H}(x)$ |
| installed time | TIME |
| log of accesses | LOG FILE |

Table 9.2: Property list for objects less subjects

| **Objects less Subjects** | |
|---|---|
| signature | $Sig$ |
| owner | OWNER_PK |
| access table | TABLE |
| time created | TIME |
| time modified | TIME |
| log of accesses | LOG FILE |
| number of allowed copies | N |
| duplication allowed | (yes/no) |

Each subject has a thumbprint which is the hash of the binary file, this is to authenticate that the software has not be modified. The subject is signed by the developer, and the property file contains the public key of the signing source. Applications can be multithreaded, in which case several other subjects can access it simultaneously, though with shared memory. Time of installation and log of accesses are stored so that at a later point they can be retraced in order to validate its authenticity, all actions done on a subject can be recreated.

All passive objects are signed by the owner, so that owner access can be authenticated. The access table (detailed in Section 9.2 provides other subjects that can access it. Time created, time modified and log of accesses is to retrace security attacks. If two subjects need write access to it, multiple copies of the object be made up to N copies if duplication is allowed.

## 9.2 Access Table for Entities Unique to the Smartphone

Unlike a generic computer setting and its corresponding security model, a smartphone is specialized to support certain necessary functionalities, and so a smartphone security model must address these functionalities/applications as special cases.

Table 9.3 lists the applications special to the smartphone, along with some of their key attributes. Applications such as telephony, texting, global positioning system, electronic wallet and bluetooth, are hardware constrained and only one instance of each can be active at any given time. However, it is still possible for multiple subjects to be allowed access to them each sharing a time multiplexed form of the resource. For this reason, these applications are written as having multiple active copies.

Our goal here is to model security on a smartphone, thus we MUST protect those subjects and objects that are unique to the smartphone. In Table 9.3 we distinguish and categorize the different objects that are often present on smartphones, and which need to be protected. For each of these objects, we ask ourselves that consider a subject $s \in S$ which wishes to access an object/ subject $x \in \{S \cup O\}$, under what criteria will $s$ accessing $o$, be allowed?

We now consider access rules based on these special applications as listed in Table 9.3 all of which belong to $S \cup O$, and see under what conditions can they be accessed and by which subjects.

### Telephony

The most important task for a smartphone is the telephony process which allows users to make and receive phone calls. The two actions associated with telephony (TEL) are to make call or receive call. Both sending and receiving a call can cause money being charged to the user. Thus the user needs to sign off on the calls and the permissions for doing so cannot be handed off to applications without explicit user (USR) consent each time. Further a great concern would be that some rouge application was created and distributed that spammed the cellular telephone network,

Table 9.3: List of subjects and objects

| **Entity** | **Abbr.** | **Type** | **Multi-Threaded** | **Copies** *time-sliced |
|---|---|---|---|---|
| telephony | TEL | $S$ | $No$ | $Yes^*$ |
| texting | TXT | $S$ | $No$ | $Yes^*$ |
| manufacturer | $S$ | S | $No$ | $No$ |
| kernel | KRN | $S$ | $No$ | $No$ |
| user | USR | $S$ | $No$ | $No$ |
| cellular service provider | CSP | $S$ | $No$ | $No$ |
| global positioning system | GPS | $S$ | $No$ | $Yes^*$ |
| electronic wallet | EWA | $S$ | $No$ | $Yes^*$ |
| core device applications | CDA | $S$ | $Yes$ | $Yes$ |
| trusted third party applications | TTA | $S$ | $Yes$ | $Yes$ |
| third party applications | TPA | $S$ | $Yes$ | $Yes$ |
| user data item | UDI | $O \setminus S$ | $Yes$ | $Yes$ |
| bluetooth | BLU | $O \setminus S$ | $Yes$ | $Yes^*$ |
| physical link: Firewire/USB | PHL | $O \setminus S$ | $Yes$ | $Yes^*$ |

thus limiting usage to others. By limiting subjects that can make calls via telephony (TEL), we can preserve the cellular network from attacks via the device even if all other applications on the device has been compromised.

One application that may need to access telephony is the record (REC) application, which would make a copy/transcript of a call. In such cases, and in using TEL in general, the calling subject can obtain access of resources such as call logs, contact book and voicemail, which includes reading and deleting them.

Also observe that the user (USR) consent should be explicitly required before an incoming call is picked up, as it could otherwise lead to attacks such as remote attacker stealthy listening into a user's environment. Hence, telephony (TEL) on the

device should be a protected functionality and its access should be restricted the user (USR) or any application working on behalf of the user such as handsfree application (HSF).

Moreover subjects such as manufacturer (MAN) and service provider (CSP) should be barred access and control of telephony on the device, even though they each can still make changes to the software installed and the call connections respectively. The access set that is allowed to access telephony is: {user, Handsfree Application, Call Recording Application}.

Table 9.4 outlines the subjects that can access TEL, as well as what access types are granted. and the corresponding $cur(TEL)$ before and after access is granted. Figure 9.1 illustrates the access policy.

Table 9.4: Access to telephony

| **calling_entity** | **access type** | **permission** |
|---|---|---|
| user | make/receive call | allowed |
| handsfree app. | make/recieve call | allowed |
| call record app. | make call | disallowed |
| call record app. | receive call | allowed |

**Texting (SMS)**

Another important task for a smartphone is the SMS/TXT process which allows users to make and receive text messages. The SMS application is by default always allowed access to objects such as Address Book and SMS logs. Also, the SMS application has the right to call or invoke the System Alert object in order to notify the user. However, given the instances of potential malware attacks via SMS not all subjects calling SMS can be expected to given an object handle to the SMS application running in its full capability. Here Core Device Applications (CDA) is a reference

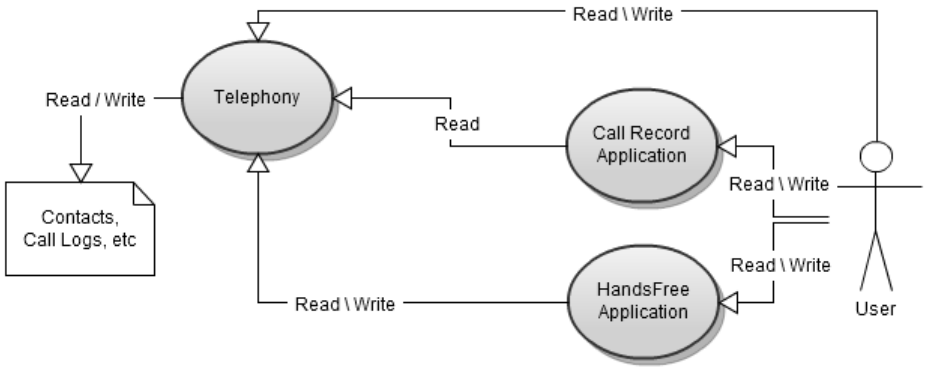


Fig. 9.1.: Accessing telephony

to applications that include camera, browser and email clients. While Trusted Third Party Applications (TTA) can include applications such as for social networking. Not unlike the TEL functionality, the SMS or simply texting (TXT)application can cause charges to the user when texts are sent or received. However, contrary to telephony where the authorized subjects can choose to take or decline a call. The application is always open to receiving a message unless the application is disabled. Also the texting (TXT) application and in turn any subject calling it has hold of objects such as contact book and sms/mms logs, which includes reading or deleting them. Hence once again TXT on the device should be a protected functionality and its access should be restricted the user (USR) or any application working on behalf of the user such as handsfee application (HSF). Once again subjects such as manufacturer (MAN) and service provider (CSP) should be barred access and control of telephony on the device, even though they each can still make changes to the software installed and the call connections respectively.

The texting send nor the receive feature should not be multi-threaded nor duplicated. The texting send feature is reserved primarily for user, but allowed under certain circumstances, as detailed below. The receive feature is reserved as well primarily for user, all other allowed received functions to TXT are described below.

Table 9.5 outlines the subjects that can access TEL, as well as what access types are granted. and the corresponding $cur(TEL)$ before and after access is granted. Figure 9.2 illustrates the access policy.

Table 9.5: Subject access to texting

| **calling_entity** | **access type** | **permission** |
|---|---|---|
| user | send/read | allowed |
| handsfree app. | send/read | allowed |
| CDA | send | allowed |
| CDA | read | disallowed |
| TPA | send | allowed |
| TPA | read | disallowed |
| other | send/read | disallowed |

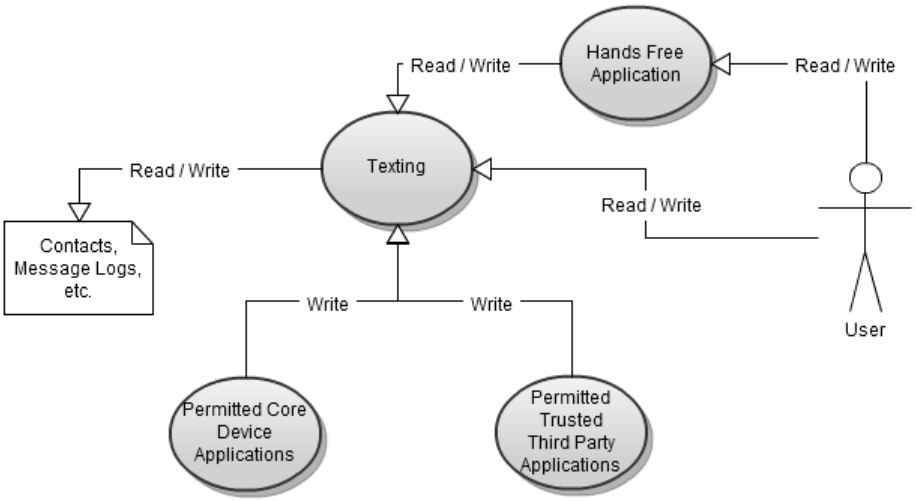


Fig. 9.2.: Accessing texting

### Kernel/Trusted Computing Base

One of the many functions of the Kernel is that it enforces all rules and policies, thus it is the security monitor of the smartphone. Clearly, any breach to kernel access will impact the security of the device. The kernel is the trusted computing base of the smartphone and its confidentiality, integrity and availability must be preserved at all times by policy. In terms of confidentiality, no unauthorized subject can read kernel level data. By preserving integrity, no unauthorized subject can read write to kernel level data. Regarding availability, no low-trust process can lock/reserve subjects and objects that a kernel process requires. The kernel cannot be modified, except under strict process initiated by the manufacturer or manufacturer authorized software. The access set that is allowed to access the kernel is:: { MAN }.

In regards to $prop(KRN)$, there is no threading nor duplicate copies available of the kernel. The following table outlines the subjects that can access KRN, as well as under what circumstances it is granted. and the corresponding $cur(KRN)$ before and after access is granted.

Table 9.6: Access to kernel

| **calling_entity** | **access type** | **permission** |
|---|---|---|
| MAN | read/write | allowed |
| other | read/write | disallowed |

### Global Positioning System/Location Database

Global Positioning System (GPS) has the capability of identifying current location information, as well as other functionalities, as well as reading and writing this information to the location database. Though other applications may request current location information, they should do so by invoking a feature of the GPS application

rather than directly access the location database. Consequently, access to the location database should be restricted to the GPS subject. Access to the GPS subject would need to be allowed to a category of third party applications. However when called upon by subjects of different capabilities, the GPS itself would escalate or reduce its own capabilities and access to records, dependent on the subject's trust level who invokes GPS. Hence altering the portion of records allowed to be accessed. In all GPS can be invoked from a range of subjects, the user (USR) and manufacturer (MAN) to permitted trusted third party applications (TPA). Lastly, the GPS application is the sole subject that can use the GPS-Satellite channel, which too must be considered to be at the same trust level. The access set that is allowed to access GPS is: {MAN, KRN, USR, EWA, CDA, TTA}.

Location based information is very sensitive information and should not be revealed nor should it flow to other parties as seen in Section 7.3. Thus access to GPS services needs to be closely monitored.

Table 9.7 outlines the subjects that can access GPS, as well as under what circumstances it is granted. and the corresponding $cur(GPS)$ before and after access is granted.

Table 9.7: Access to GPS

| **calling_entity** | **access type** | **permission** |
|---|---|---|
| MAN | read | allowed |
| KRN | read | allowed |
| USR | read | allowed |
| EWA | read | allowed |
| CDA | read | allowed |
| TTA | read | allowed |
| other | read/write | disallowed |

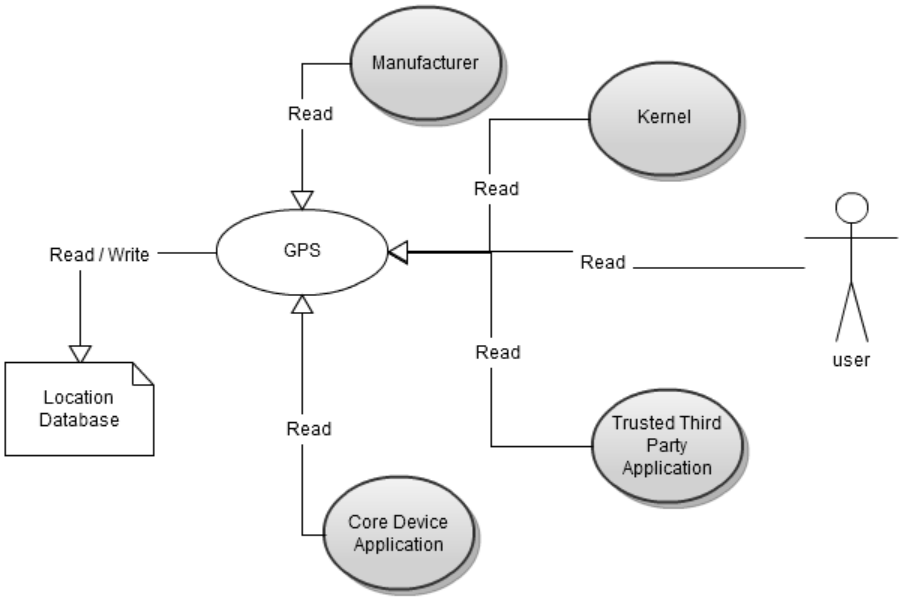


Fig. 9.3.: Accessing GPS

**Electronic Wallet**

Electronic Wallet, also called e-wallet, (EWA) will consist of a number of features– banking, credit cards, keys (car, home,...), personal identifications (driver license, passport,...), etc., providing capability of electronic transactions, as well as a number of other features. Sensitive files will be accessible to this feature, thus access to the e-wallet functionality must be closely safeguarded. The only way to seal of the e-wallet (EWA) from being invoked by less trusted applications requiring its services, is to allow only the e-wallet to access other applications, and never the other way around.

The e-wallet should only be accessed by the user (USR) or processes can ask the kernel to invoke it on its behalf. In effect, for every transaction the user invokes the e-wallet and directs it to the application requiring the transaction - may it be some webpage or application store portal.

The e-wallet (EWA) thus acts in highly trusted user mode, and communicates off device via its own SSL connections. The access set that is allowed to access electronic wallet is: { KRN, USR }.

Table 9.8 outlines the subjects that can access EWA, as well as under what circumstances it is granted. and the corresponding $cur(EWA)$ before and after access is granted.

Table 9.8: Access to electronic wallet

| **calling_entity** | **access type** | **permission** |
|---|---|---|
| KRN | write | allowed |
| USR | read/write | allowed |
| other | read/write | disallowed |

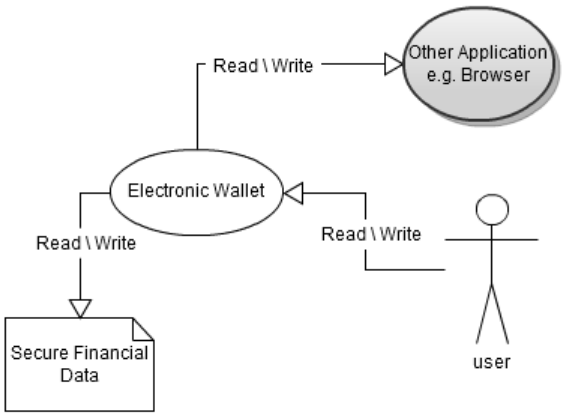


Fig. 9.4.: Accessing electronic wallet

### Core Device Applications

Applications such as the browser, email client, camera and audio player are examples of core device applications (CDA). These applications all belong on the same confidentiality, integrity and availability levels, since they consistently require each

others' services and data from one another, they can be accessed to be used by another core device application. When data is being accessed from any of these applications by another (one of the CDA is acting as a subject), these CDA objects need to all be in a high trust status. When a higher ranked application such as telephony, SMS/MMS, GPS or EWA is the authorized subject accessing a CDA object, then such applications can only a read access to a portion of the CDA resource. This is because, even though the subjects have higher rankings, in case they are compromised they should not be a allowed to execute or write all other applications. The access set that is allowed to access core device applications is: {KRN, MAN, USR, GPS, EWA, TTA}.

Table 9.9: Access to core device applications

| **calling_entity** | **access type** | **permission** |
|---|---|---|
| KRN | write | allowed |
| MAN | read/write | allowed |
| USR | write | allowed |
| GPS | read/write | allowed |
| EWA | write | allowed |
| TTA | read/write | allowed |
| other | read/write | disallowed |

### Third Party Applications

Third Party Application can be come from various sources, and in a generic smartphone model they can be installed via several different channels. They can be under different credibility ranking, which we will simply generalize as Trusted Third Party Applications (TTA) or Less Trusted Third Party Applications (TPA), though all lower than Core Device Applications. In general particular data items associated

with an application are to be sandboxed by default unless they are explicitly declared to be shared with different applications and had obtained permission to do so. Also, to preserve credibility, lower applications cannot read, modify or execute higher applications. To preserve integrity, lower applications cannot write to higher applications. And to preserve availability, higher applications have higher priority in accessing resources than lower applications. The access set that is allowed to access trusted third party applications (TTA) is: { KRN, USR, EWA }.

Table 9.10: Access to trusted third party applications

| **calling_entity** | **access type** | **permission** |
|---|---|---|
| TEL | write | allowed |
| KRN | write | allowed |
| USR | write | allowed |
| EWA | write | allowed |
| other | read/write | disallowed |

Table 9.11: Access to third party application

| **calling_entity** | **access type** | **permission** |
|---|---|---|
| KRN | write | allowed |
| USR | write | allowed |
| EWA | write | allowed |
| TTA | read/write | allowed |
| other | read/write | disallowed |

### User Data Items

Private Data Items belonging the user consists of passive objects such as pictures, songs, saved Wifi settings, contacts, themes and wallpapers, password file, games scores, 'clipboard', etc. As it can be seen all of these data items are likely to be associated with one or more subjects, and must thus operate at the same level as the uniformly ranked subjects. The subjects have to be brought to the same level when accessing a common resource, which cannot be sequestered. The access set that is allowed to access user data item is: { any subject with the same trust level}.

Table 9.12: Access to user data item

| **calling_entity** | **access type** | **permission** |
|---|---|---|
| any authorized subject | read/write | allowed |

### Bluetooth

Bluetooth is to be used by trusted third party applications such as wireless headsets. These application must use application layer encryption to create secure communication To remote controlling no core application use this channels. The kernel must access the bluetooth channel to monitor its activity as well as periodically switch it on or off. The user too ofcourse can physically turn it off or on The access set that is allowed to access bluetooth is: { USR, KRN, TTA}.

### Physical Link: Firewire/USB

The physical link is to be used for both system and data backup The backup application is expected to to establish a secure connection and encrypt data being transferred The access set that is allowed to access physical link is: {KRN, USR, UDI, TTA, TPA}.

Table 9.13: Access to Bluetooth

| **calling_entity** | **access type** | **permission** |
|---|---|---|
| KRN | write | allowed |
| USR | write | allowed |
| TTA | read/write | allowed |
| other | read/write | disallowed |

Table 9.14: Access to physical link

| **calling_entity** | **access type** | **permission** |
|---|---|---|
| KRN | write | allowed |
| USR | write | allowed |
| UDI | write | allowed |
| TTA | read/write | allowed |
| TPA | read/write | allowed |
| other | read/write | disallowed |

## 9.3 Mechanism: Installer

The installer is part of the Trusted Computing Base, and can not be modified by any party but the manufacturer. It is single threaded and only one copy exists at any given time. It is a special application because it needs to access objects belonging to all trust levels. Also, the only party invoking the installer is the user. The user not only calls the installer for application installation, but also for providing install time user permission. These are permission that the application requests for, by the device is not the authority in approving them on behalf of the user.

As seen, all applications need a to be signed. This necessities the presence of a Public Key Infrastructure, which means that all application developers, from the very

trusted manufacturer to the least trusted third party application market developer, need to first obtain a signing key. It is possible that the user may want to install an application directly to the smartphone, in which case the application will be installed using the users keys. For this the user will obtain his/her key at device registration with manufacturer.

Once the installer unpacks an application $x$, it creates $prop(x)$ reading its attributes. For fields in the property file where values are unknown, it defaults to minimum values. The installer also reads the requested permissions, and generates $con(x)$, $int(x)$ and $avl(x)$, setting the context for all subjects $s \in S$ in which they can access $x$

The installer also looks for a privacy policy file in the application package. The installer can then process the privacy information and display it in a user-friendly format of a privacy grid or highlight privacy concerns.

The above installer actions are illustrated in Figure 9.5

## 9.4 Mechanism: Security Monitor

For any subject $s$ to access any subject/object $o$, the access control is performed by the security monitor, which is part of the Trusted Computing Base. The role of the security monitor is to authenticate each access following a three stage verification, namely: 1) check for authenticity, 2)check for availability, and 3)check for confidentiality and integrity. In Figure 9.6 we see the overall flowchart on monitoring access control. Each stage has an associated algorithm and each fits into a category of checking for authenticity, availability or confidentiality and integrity. We now discuss the purpose and mechanism behind these three classifications.

### Checking for Authenticity

To check for authenticity, the security monitor first looks at the access Table 1 associated with object $x$ to ensure that the access is permitted by the $x$'s policy

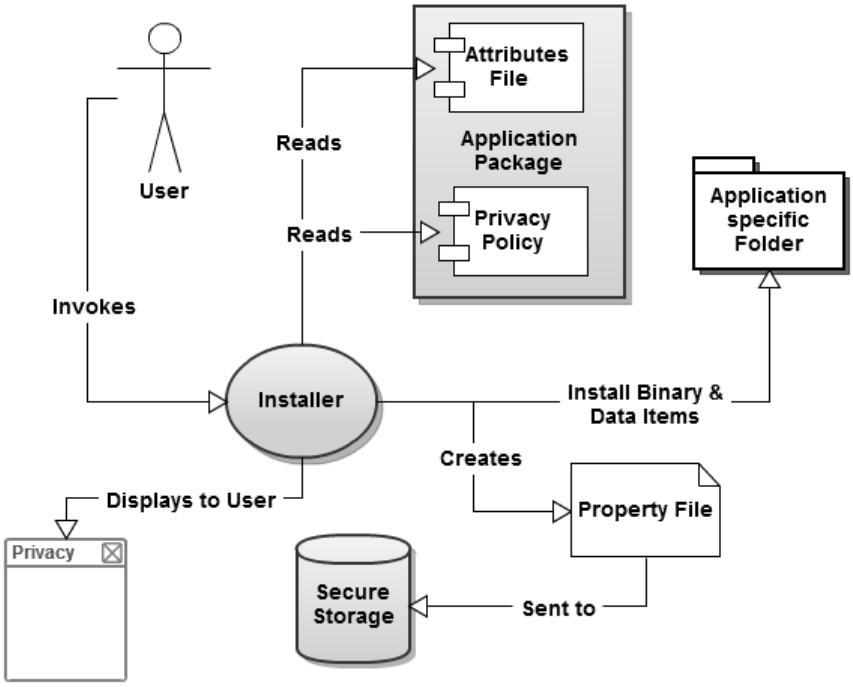


Fig. 9.5.: Installation mechanism

developer. The access tables for all distinct smartphone objects are given in Section 9.2.

**Algorithm 1** Table Lookup

**Input** $x$, calling_entity, $x$.access_table, access_type

**Output** allow/disallow

1: return $x$.access_table(calling_entity, access_type).permission

The second step is to create a hash of the application binary and compare it with the pre-stored hash (see Algorithm 2) to see if it has been modified. Applications are allowed to me modified, but when they are modified by authorized parties, their property file's also modified.

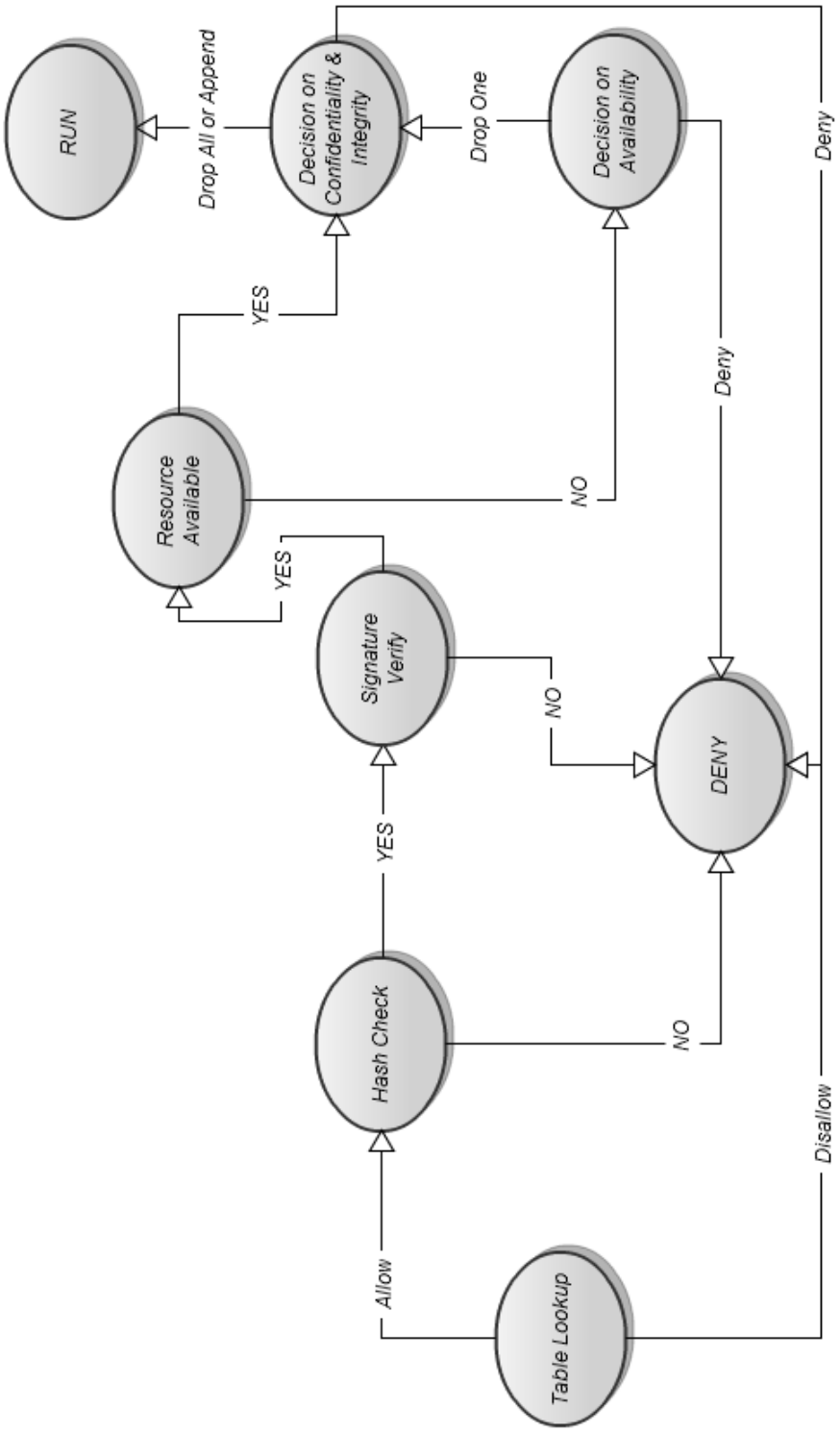


Fig. 9.6.: Access control state diagram for security monitor

**Algorithm 2** Hash Check

**Input** $x$, $prop(x)$

**Output** yes/no

1: $y = \text{Hash}(x)$

2: return $y \oplus prop(x).\mathcal{H}(x)$

Lastly, the signature of the developer/modifier is verified by the signers public key as per Algorithm 3. Though signature checking happens during installation, this is part of the policy for complete mediation.

**Algorithm 3** Signature Verify

**Input** $x$, $prop(x)$

**Output** yes/no

1: return Signature_Verify($Sig$, $x$, $Signer\_PK$)

### Checking for Availability

To check for availability, the security monitor first checks if any copies of resource $x$ is available, or if resource $x$ is multithreaded. This check is done by Algorithm 4. If the resource is multi-threaded, then the resource is always available. If it has a limit on the number of copies that can be made, and that limit has not exceeded then it is still available, else it moves on to the Algorithm 5 for decision on availability.

**Algorithm 4** Resource Available

**Input** $cur(x)$, $prop(x)$

**Output** yes/no

1: **if** $prop(x).multithreaded = TRUE$ **then**
2: return $TRUE$
3: $N$ = number of calling_entities in $cur(x)$
4: **if** $N > prop(x).num\_of\_copies$ **then**
5: return $TRUE$
6: return $FALSE$

Before arriving at this later algorithm, it is already decided that all available copies of object $x$ have been taken up. It is also know that these copies are sequestered and independent, and can be shared by subjects of varying trust levels. Algorithm 5, thus decides if the *availability index (avl)* of the current subject dominates the *availability* of the lowest ranked subject. If true, then the subject with lowest availability ranking is dropped and the current subject is moved to next sequence of decisions; if false then the current subject is dropped.

**Algorithm 5** Decision on Availability

**Input** $x$, $cur(x)$

**Output** yes/no

1: $tmp = HIGH$
2: For all $y \in cur(x)$.calling_entities
3: **if** $avl(y) < tmp$ **then**
4: $tmp = avl(y)$
5: **if** $avl(y) < avl(x)$ **then**
6: Drop $y$
7: return $TRUE$
8: return $FALSE$

**Checking for Confidentiality and Integrity**

For all subject/objects, one or more copies can be active at the same time, each independent and sequestered from each other. The knowledge of how many copies can be available and and how they are to be rationed is to be used to determine availability.

We also observe that some subject/objects, can be 'multi-threaded'. We use the term multi-threaded to mean that different instances can it be created with memory shared in-between them, such that multiple subjects can grab different threads of an object simultaneously. However, if all subjects accessing an threads of an object do not belong to the same security level, then information flowing between different security levels will reduce all subjects to the level of the subject with lowest security. Figure 9.7 which illustrates the importance of sequestering. The illustration shows the potential risk if two subjects (one low level the other high level) access the same object. Depending on the security metric, information flow from high confidential object to one of low confidentiality, degrades the high level objects. Similarly information flow from low integrity to high integrity brings the high level object to the low one's integrity level.

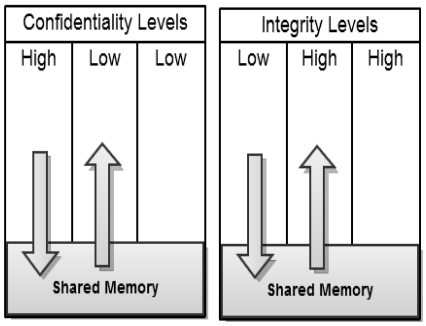


Fig. 9.7.: Threaded object shared between subjects of different trust

Before making decisions based on confidentiality and integrity in Algorithm 6, it is known that either a thread or a copy of object $x$ is available. However, this availability could have been decided on either multiple copies or multiple threads being available. If it is the case of multiple sequestered copies, then any allowed subject irrelevant of confidentiality and integrity levels can access it. Else, it is know that all the the subjects currently accessing the threads are of the same confidentiality and integrity levels. If the current subject has either lesser confidentiality or integrity, then it is dropped. If it is higher in both trust indices, then all subjects presently accessing the object's threads are dropped.

## 9.5 Mechanism: Applications Marketplace

Unauthorized third-party code can be prevented by making code mandatory to be signed, this way any file an exploit is able to write out to disk will not be allowed to be executed by the kernel.

There should always exist certificate authority who will sign the code before it can be distributed. Prior to installation on device, the software package should be validated by verifying the digital signature, confirming the legitimacy of permissions requested and check the files contained in the package. It should also look for a privacy policy file which is sufficiently completed as per standard guidelines and see that it can be viewed by the user.

The package installer on behalf of the user should be able to decline a subset of requested permissions, depending on a system wide device security settings that the user wants the device to maintain. By approving permissions or small groups of permissions individually there is less risk of an application maliciously misusing the permissions granted to it.

**Algorithm 6** Decision on Confidentiality & Integrity

**Input** $x$, $cur(x)$, $calling\_entity$

**Output** yes/no

1: **if** $prop(x).multithreaded = FALSE$ **then**
2: append log in property file
3: return $TRUE$
4: ▷ Current calling entities are sharing memory and thus have equal *con* & *int*
5: For any $y \in cur(x)$
6: **if** $\mathcal{F}(con(x), calling\_entity) \neq \mathcal{F}(con(x), y)$ **then**
7: **if** $avl(calling\_entity) > avl(y)$ **then**
8: Drop all active processes in $cur(x)$
9: append log in property file
10: return $TRUE$
11: append log in property file
12: return $FALSE$
13: **if** $\mathcal{F}(int(x), calling\_entity) \neq \mathcal{F}(int(x), y)$ **then**
14: **if** $avl(calling\_entity) > avl(y)$ **then**
15: Drop all active processes in $cur(x)$
16: append log in property file
17: return $TRUE$
18: append log in property file
19: return $FALSE$
20: Append $calling\_entity$ to $cur(x)$
21: append log in property file
22: return $TRUE$

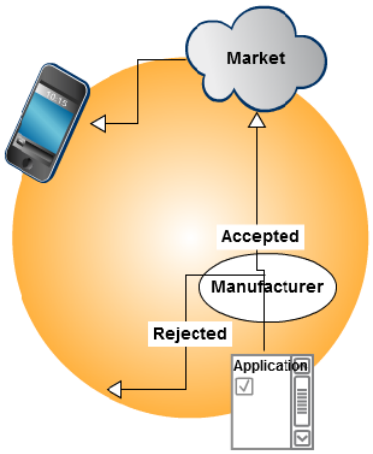


Fig. 9.8.: Closed application market

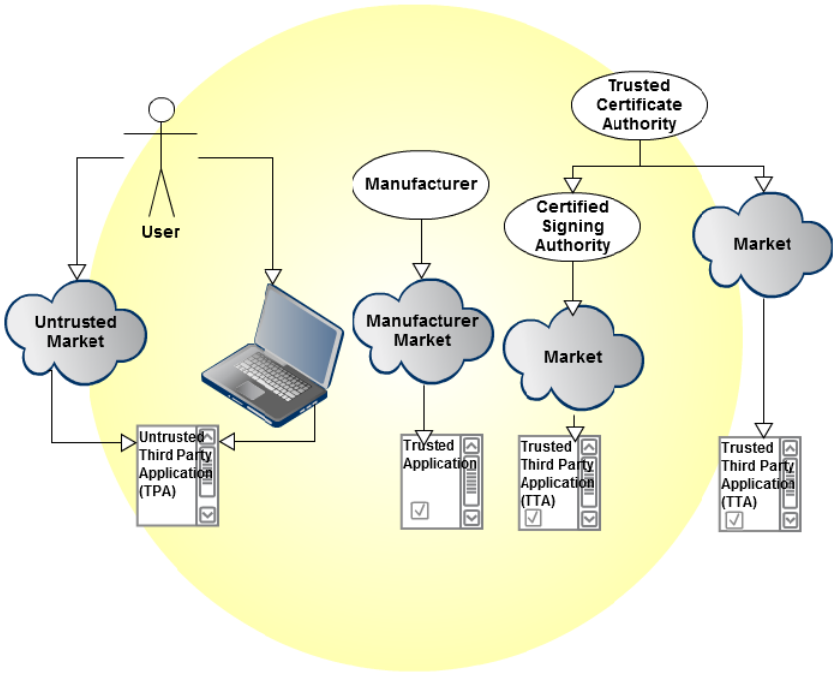


Fig. 9.9.: Public key infrastructure for application distribution

## 9.6 Mechanism: Securing Backup, Syncing and Data Transfers

Most smartphone have their own desktop software such as iTunes and Windows Zune to provide interface for the smartphone to the PC. However, as seen

in Section7.2, such software cannot be relied on to provide necessary security features. All smartphones also do not allow access the the phone's filesystem externally. While some like Windows Phone, can allow external communication via the cloud. Whatever the case, security features should be implemented by the device itself.

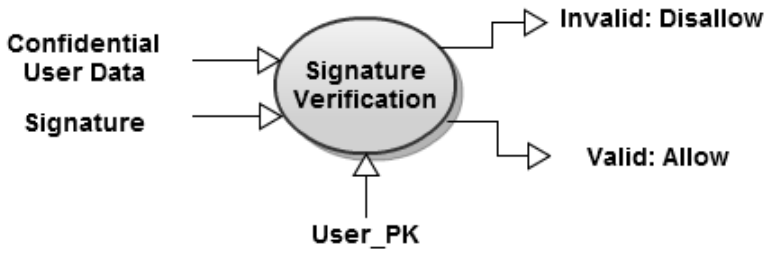


Fig. 9.10.: User signing confidential user data

Data on the device can be broken into application data, user data, and confidential user data. In case of application data, it should always be backed up as a signed package. Upon installation the signature is then verified. User data in general should be always encrypted before being sent outside the device, to secure it on the communication channel as well as external storage. Confidential user data include financial, location, and other privacy sensitive information used by special applications such as E-Wallet and GPS. Since confidential user data needs higher security and is also tied more the the individual rather than the device, it should be encrypted and signed by the a personal special set of user keys while on device. When transferred outside the device, user keys are no longer available, and so the data should be encrypted and locked under a user passphrase. While on device, when the device is handed off to another user, this personal secret key is overwritten with a newly generate user key, upon users prompt, and the confidential user data encrypted with it are deleted/overwritten by newer data. Data access to confidential data should follow the given verification as shown in Figure 9.10.

# 10. RESULT ANALYSIS

In Chapter 5 we categorized and discussed smartphone threats in ten categories. We then discussed security policies, model and mechanisms which can counteract against these threats. In this chapter we analyze the means and effectiveness of our proposed measures. For each, we revisit the threat, discuss how it is subverted, briefly state a sample attack and follow up with a series of security steps that show why the attack cannot happen. The table below each threat, is hence a sequence of security measures counteracting the attack at every stage.

Before that we recall our assumptions that trusted computing base is a protected entity, consisting of the kernel, filesystem and installer, that all proposed security mechanisms are implemented and the manufacturer has correctly handled secure bootloading and firmware security. We also maintain that there is secure storage on the device to be storing property files for all objects loaded on the smartphone.

**Threat 5.1:** *The larger the attack surface for a smartphone, the more likely that it will be exploited.*

The smartphone attack surface is the collection of inputs, protocols, interfaces and services. The challenge was thus to limit the attack surface without decreasing functionality. For this we designed a secure smartphone operating system at an abstract level, which unlike prevailing smartphones is not reduced from a bigger operating system. We also defined security in terms of the special smartphone applications, giving each a unique consideration in terms of access and modification rights. We now run an sample attack on how the threat of wide attack surface can be violated. One of the trusted native applications which is most open to outside parties is the browser. Suppose an attacker wishes to execute untrusted code on the browser, she

would need to bypass the security steps in following attack example of a nefarious program trying to install software via the browser.

We now follows the sequence of countermeasures against Threat 5.1:

1: program runs code on browser to get installed
2: browser request for access to installer to security monitor
3: security monitor checks if the installer is not previously engaged as per Figure 9.6
4: the installer can only be called by the user, hence the user is notified as per Figure 9.5
5: if the user agrees, the program is installed under user signature with minimum privileges of untrusted third party application, that user signed code receive as per Figure 9.9
6: if the user see that the installation is unintended then the installation is rejected, a log entry is made of the browser breach as per Algorithm 6

**Threat 5.2***: With the growing amount of sensitive data being stored on a smartphone, and the corresponding lack of security in storage, communication and application installation, the smartphone is open to data theft.*

In order to prevent disclosure of information or privacy leaks via applications we decided on two policies and mechanisms. First we decided that the installer should read and display the privacy policy for any application prior to installation. We also stated that the user has the choice of agreeing to a subset of privacy rights that the application is seeking. As a second mechanism, we paired each confidential data file with a secure application, such as financial data with e-wallet and location data with the GPS application, and made the application the sole interface for data access. We also defined access rules for special applications limiting read and write to them, while allowing them to read and write to approved subjects/objects.We now demonstrate an example where a nefarious application attempts to obtain trusted third party application status and leak location data from the device.

We now follows the sequence of countermeasures against Threat 5.2:

1: Application tries to obtain certification from a trusted market place as per Figure 9.8
2: Application is verified and its privacy policy is cross checked with its binary execution
3: Application is packaged and signed along with the certificate
4: Installer verifies the signature and signer, unpacks the application, displays privacy policy to user, as per Figure 9.5
5: User approves to a subset of permissions.
6: Installer create property file and install application as per Figure 9.5

**Threat 5.3***: Security mechanisms are easier to bypass if authentication and Verification is not done more rigourously.*

We stated in Threat 5.3, that security mechanisms are at risk of being fooled with false data and that the biggest threat for spoofing with false data, is to override/trick security checks. In response we outlined the security monitor to check every subject/object against stored hashes as well as the signatures on them. We also stated the concept of the secure storage for property files and of an independent non-tamperable security monitor. Let us take the example of a third party application (TPA) trying to access a user data item (UDI). Let us assume that this TPA had asked for such access rights at installation as per Table 9.12.
We now follows the sequence of countermeasures against Threat 5.3:

1: Application passes request to security monitor which will authorize it through a multi-step verification from Figure 9.6
2: Security Monitor verifies authenticity and compares against Access Table lookup (Algorithm 1), signature checking (Algorithm 3) and hash check (Algorithm 2)
3: Security Monitor ascertains if a copy of the data item would be available through Algorithms 4 and 5

**Threat 5.4***: Malicious parties can seize control by disrupting the secure flow of operations.*

Once again to avoid disruption of correct operation, we define the actions of the security monitor to be atomic and secure. We also make every access to be communicated down to the security monitor. Also, adhering to the principle of fail-safe defaults, we deny all access unless approved on every access basis by the monitor. An example of this is a trusted third party application trying to access a single threaded application while the later is in use.

We now follows the sequence of countermeasures against Threat 5.4:

1: Application passes request to security monitor which will authorize it through a multi-step verification from Figure 9.6

2: Security Monitor verifies authenticity and compares against Access Table lookup (Algorithm 1), signature checking (Algorithm 3) and hash check (Algorithm 2)

3: Security Monitor ascertains if the object would be available through Algorithms 4 and 5

4: Security Monitor ascertains if application is in high enough confidentiality and integrity level for the object through Algorithm 6.

**Threat 5.5***: If the system is partitioned into different security levels, a vulnerability in any one of them can allow for access and control of all code running at that level.*

For unauthorized control of part of the device, we prevent user space hacks can that can let a malicious code allow access to the entire filesystem by making the filesystem a part of the protected trusted computing base. The trusted computing base functions at kernel trust level, and the kernel access table and protections are applied to it. The example for this follows closely with that for the last two threats. In this case the object being accessed is the filesystem, and being part of the trusted computing base, it follows the same access rules as for the kernel. Hence, only the manufacturer is allowed to modify it.

**Threat 5.6***: The smartphone is vulnerable to being highjacked via cellular network botnets.*

We stated that smartphones botnets can exploit both cellular and internet connectivity in tandem increasing threats to the cellular network. For this we made special access rules for telephony making it accessibly only by user permission. It could be argued that indirect access by certain applications may enhance functionality. However given that the cellular network causes charges to the user, that the cellular network is more rigid and vulnerable to botnet attacks, and that cellular connection are not behind protected networks, it is necessary that the user is the sole accessor to the telephony functionality. As an example, let us see how a handsfree application working on behalf of the user operates.
We now follows the sequence of countermeasures against Threat 5.6:

1: The user calls the hands-free application
2: Hands free application obtain access to telephony
3: Contacts, call records and other telephony user data files are preserved behind the telephony application
4: The hands-free application quits once user gives up control

**Threat 5.7***: By now allowing certain amount of flexibility in application development and installation, users may instead willing bypass security mechanisms introducing more vulnerabilities.*

We noted that each smartphone vendor has different mechanisms of application distribution in Threat 5.7. Also that very restrictive marketplace lead to jailbreaking, while no system for certification and quality checking is plan in harmful for the device widening the attack surface. For this we designed an application market place designed upon public key infrastructure distinguishing between trusted and uncertified third party applications. We also incorporated manufacturer certified applications, making them most trusted and user approved ones, which though allowed have least

privileges. How the user can authorize unsigned applications by themselves or other developers, and install them as an untrusted third party application is shown in Figure 9.9.

**Threat 5.8***: The smartphone is vulnerable to application layer malwares*

We stated that application layer malware is any piece of code that opens up vulnerabilities - remotely control the device, modify the filesystem, exposing confined data or installing rootkit. Though each of these threats have been addressed separately. A more general solution against malwares of all kinds was proposed to maintain a security access log in the security monitor so as to later be able to analyze malware activities. This is shown in Figure 9.6 and Algorithm 6.

**Threat 5.9***: The smartphone is vulnerable to rootkits.*

Kernel level malware or rootkits exploited vulnerabilities in the operating system level. For this we kernel only accessible by the manufacturer (see Table 9.6), and give no read or write access to any other party to the kernel.

**Threat 5.10***: Insecure data transfer.*

Lastly, we stated that while data transferring outside the device, could lead to both contamination/unauthorized modification of data, as well as loss of sensitive information. For this we define mechanism for applications, data and confidential data traveling outside the device under any communication channel and protocol. Recall that for data to travel outside the device it should fulfil the following steps. We now follows the sequence of countermeasures against Threat 5.10:

1: The application should notify user for invoking permission to transfer
2: The data should be encrypted
3: If it is being backed up, then it should be stored under a user passphrase
4: If it confidential user data, then it should be signed with user keys

# 11. CONCLUSION AND FUTURE WORK

In this thesis we have analyzed smartphone security and have proposed new security model and mechanisms for the smartphone platform, based on privacy, integrity and availability. We analyzed our model, to show it circumvents previously experienced and future foreseeable threats and attacks on the smartphone.

We introduced the smartphone and its applications from a security perspective. Specifically, we emphasized the growing popularity and changing trends in smartphone applications which increase the need for security features protecting user information and identity. Then, we discussed and compared how prevailing smartphone brands address security on their devices. We collected, consolidated and summarized ten categories of smartphone threats and vulnerabilities as witnessed in the smartphone security scene since its inception in the consumer markets. We based the resolvement of these ten threats as the goal for our policies, model and mechanism in the chapters that follow. After that, we briefly discussed the existing research focus on enforcing security on smartphones, and we see that almost all works focus on adding security enhancements on top of an existing platform. We distinguished our work as a bottom-up-approach in redefining a secure smartphone environment, with emphasis on its special applications. We developed security policies which are targeted to address the security shortcomings and need that were previously presented. We used these policies as the basis to develop a formal model defining how security measures can be systematically built into the system. We also proposed accompanying security mechanisms, which will ensure that all applications and stakeholders and protected. Finally, we analyzed the effectiveness of our work in the light of the threats that were discussed.

For the future, this work can be expanded by creating and testing a small scale implementation of the model. Though we believe the model is at par with current smart-

phone performances, through implementing the proposed model and mechanisms the empirical efficiency and viability of the design can be tested. Also, a method of faster signature checking, such as elliptic curve cryptography can be added to the model. Lastly, decisions on the precise implementation of our methods which are described at an abstract level need to be made based on various situational, corporate and market factors.

# LIST OF REFERENCES

# LIST OF REFERENCES


[1] PhoneScoop, "Glossary:smartphone." http://www.phonescoop.com/glossary. Last accessed: July 2011.

[2] A. Nusca, "Smartphone vs. Feature Phone." http://www.zdnet.com/blog/gadgetreviews. Last accessed: July 2011.

[3] R. Ballagas, J. Borchers, M. Rohs, and J. Sheridan, "The Smartphone: A Ubiquitous Input Device," *IEEE Pervasive Computing*, vol. 5, no. 1, pp. 70–77, 2006.

[4] M. Brownlow, "Smartphone statistics and market share." http://www.email-marketing-reports.com/wireless-mobile/smartphone-statistics.htm. Last accessed: July 2011.

[5] R. Wilson, "TI shows four ARM core smartphone processor." http://www.electronicsweekly.com/Articles/2011/02/10/50456/ti-shows-four-arm-core-smartphone-processor.htm. Last accessed: July 2011.

[6] comScore, "comscore Reports U.S. Mobile Subscriber Market Share." comScore.com. Last accessed July 2011.

[7] F. Y. Rashid, "Symantec Reports Targeted Threats, Mobile Attacks increased in 2010." http://www.eweek.com/c/a/Security/Symantec-Reports-Targeted-Threats-Mobile-Attacks-Increased-in-2010-191684/. Last accessed: July 2011.

[8] F. Rashid, "Mobile Malware, Hacktivism top List of Major Security Concerns." eWeek, April 6 2011. Last accessed: July 2011.

[9] L. Cassavoy, "Skype for Android security flaw: What you need to know." http://www.cio.com.au/article/383511/skype_android_security_flaw_what_need_know/. Last accessed: July 2011.

[10] B. Levine, "Apple Issues iOS Update to Fix Some Problems." http://www.newsfactor.com/news/Apple-s-iOS-Update-Fixes-Problems/story.xhtml?story_id=122000A8US0A. Last accessed: July 2011.

[11] B. X. Chen and M. Isaac, "The database makes a tempting target for law enforcement." http://www.wired.com/gadgetlab/2011/04/iphone-location/. Last accessed July 2011.

[12] A. Efrati and D. Searcey, "U.S. probes smartphone apps on privacy." http://online.wsj.com/article/SB10001424052748703806304576242923804770968.html. Last accessed: July 2011.

[13] M. Bishop, *Computer Security*. Addison Wesley, 2003.

[14] D. F. Ferraiolo and D. R. Kuhn, "Role-Based Access controls," in *15th National Computer Security Conference (1992)*.

[15] P. C. Pfleeger, S., *Security in Computing*. Prentice Hall, 3 ed.

[16] Microsoft, "Windows Phone 7 security model," *Windows Phone 7 Guides for IT Professionals*, 2010. Last accessed: July 2011.

[17] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, and S. Dolev, "Google Android: A State-of-the-Art Review of Security Mechanisms," *Neural Networks*, p. 42, 2009.

[18] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google Android: A Comprehensive Security Assessment," *Security Privacy, IEEE*, vol. 8, pp. 35–44, March-April 2010.

[19] J. Anderson, J. Bonneau, and F. Stajano, "Inglorious Installers:Security in the Application Marketplace," 2010.

[20] A.-D. Schmidt, H.-G. Schmidt, J. Clausen, K. A. Yksel, O. Kiraz, A. Camtepe, and S. Albayrak, "Enhancing Security of Linux-based Android Devices," in *in Proceedings of 15th International Linux Kongress*, Lehmann, 2008.

[21] MacRumors, "iphone:OSX is not Mac OSX." http://forums.macrumors.com/archive/index.php/t-268363.html. Last accessed: April 2011.

[22] Apple, "Porting UNIX/Linux Applications to Mac OS X: Glossary." http://developer.apple.com/library/mac/#documentation/Porting/Conceptual/PortingUnix/glossary/glossary.html. Last accessed: July 2011.

[23] Programming4us, "Mobile Application Security: The Apple iPhone-Permissions and User Controls." http://programming4.us/mobile/1750.aspx. Last accessed July 2011.

[24] Forum Nokia Library, "Platform security." http://www.developer.nokia.com/Community/Wiki/Platform_Security. Last accessed: July 2011.

[25] Forum Nokia Library, "Introduction to symbian3." http://library.developer.nokia.com/index.jsp?topic=/GUID-E35887BB-7E58-438C-AA27-97B2CDE7E069/GUID-13987218-9427-455E-AC77-ADE6B0E9CD7E.html. Last accessed: July 2011.

[26] Microsoft, "Windows Phone 7 security and management," *Windows Phone 7 Guides for IT Professionals*, 2010. Last accessed: July 2011.

[27] J. Newman, "How Android Phones Will Use Near-Field Communication." http://www.cio.com.au/article/371015/how_android_phones_will_use_near-field_communication/. Last accessed: July 2011.

[28] Google Android, "Near field communication." http://developer.android.com/guide/ topics/nfc/index.html. Last accessed: July 2011.

[29] NFC Forum, "About NFC." http://www.nfc-forum.org/aboutnfc/. Last accessed: July 2011.

[30] B. Chacos, "NFC-Enabled car keys can help you find your lost car." http://www.maximumtech.com/nfc-enabled-car-keys-can-help-you-find-your-lost-car. Last accessed: July 2011.

[31] D. Chaum, A. Fiat, and M. Naor, *Untraceable Electronic Cash*, vol. 403.

[32] D. Chaum, "Blind Signatures for Untraceable Payments," in *International Crytology Conference*, pp. 199–203, 1982.

[33] M. Howard, "Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users." http://www.geekarticles.com/article/Mitigate-Security-Risks-by-Minimizing-the-Code-You-Expose-to-Untrusted-Users.html. Last accessed: July 2011.

[34] P. K. Manadhata and J. M. Wing, "An Attack Surface Metric," in *IEEE Transaction on Software Engineering*, 2010.

[35] S. Ragan, "Flash vulnerability opens new attack surfaceAndroid." http://www.thetechherald.com/article.php/201037/6148/Flash-vulnerability-opens-new-attack-surface- Last accessed: July 2011.

[36] A. Bose, *Propagation, detection and containment of mobile malware*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 2008.

[37] D. Cridland, "iPhone/yahoo: Too cool to do standards, too hip to do security." http://blog.dave.cridland.net/?p=32. Last accessed: July 2011.

[38] S. Thurm and Y. I. Kane, "Your Apps are Watching You." http://online.wsj.com/article/SB10001424052748704694004576020083703574602.html. Last accessed: July 2011.

[39] J. Freeman, "Bypassing iPhone Code Signatures." http://www.saurik.com/id/8. Last accessed: July 2011.

[40] theIphoneWiki, "Restore mode." http://theiphonewiki.com/wiki/index.php?title=Restore_Mode. Last accessed: July 2011.

[41] Linux man page, "fstab(5)." http://www.die.net/. Last accessed: July 2011.

[42] J. Bickford, R. O'Hare, A. Baliga, V. Ganapathy, and L. Iftode, "Rootkits on smart phones: attacks, implications and opportunities," in *Eleventh Workshop on Mobile Computing Systems & Applications*, pp. 49–54, 2010.

[43] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, P. D. McDaniel, and T. L. Porta, "On cellular botnets: measuring the impact of malicious devices on a cellular network core," in *ACM Conference on Computer and Communications Security*, pp. 223–234, ACM, 2009.

[44] C. Mulliner and J. Seifert, "Rise of the iBots: 0wning a telco network," in *Proceedings of the 5th IEEE International Conference on Malicious and Unwanted Software (Malware)*, 2010.

[45] D. Barrera and P. Van Oorschot, "Secure Software Installation on Smartphones," *Security Privacy, IEEE*, vol. 9, pp. 42 –48, May-June 2011.

[46] P. Magaudda, "Hacking Practices and their Relevance for Consumer Studies: The Example of the 'Jailbreaking of the iPhone," *Consumers,Commondities & Consumption*, 2010.

[47] J. M. F. da Trindade, C. Pham, and N. Dautenhahn, "Poster: BeR: A Microkernel Based Rootkit for Android Smartphones," 2010.

[48] Z. Wang and A. Stavrou, "Exploiting smart-phone USB connectivity for fun and profit," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, 2010.

[49] D. Muthukumaran, A. Sawani, J. Schiffman, B. M. Jung, and T. Jaeger, "Measuring integrity on mobile phone systems," in *Proceedings of the 13th ACM symposium on Access control models and technologies*, (New York, NY, USA), pp. 155–164, ACM, 2008.

[50] J. Ekberg and M. Kylanpaa, "Mobile Trusted Module (MTM)-an introduction." http://research.nokia.com/files/tr/NRC-TR-2007-015.pdf, 2007. Last accessed: July 2011.

[51] M. Kim, H. Ju, Y. Kim, J. Park, and Y. Park, "Design and implementation of mobile trusted module for trusted mobile computing," *Consumer Electronics, IEEE Transactions on*, vol. 56, pp. 134–140, February 2010.

[52] B. Dixon and S. Mishra, "On rootkit and malware detection in smartphones," in *International Conference on Dependable Systems and Networks Workshops*, 2010.

[53] N. L. Petroni, J. Timothy, F. Jesus, M. William, and A. Arbaugh, "Copilot - a coprocessor-based kernel runtime integrity monitor," in *In Proceedings of the 13th USENIX Security Symposium*, pp. 179–194, 2004.

[54] J. Zdziarski, *IPhone Forensics: Recovering Evidence, Personal Data, and Corporate Assets*. O'Reilly Media, 2008.

[55] B. X. Chen, "iPhone Tracks Your Every Move, and Theres a Map for that." http://www.wired.com/gadgetlab/2011/04/iphone-tracks/. Last accessed: July 2011.

[56] B. Wilson, "Apple: submit WiFi location information to Skyhook." http://reviews.cnet.com/8301-19512_7-10115417-233.html#ixzz1TGZann8i. Last accessed: July 2011.

[57] P. Porras, H. Sadi, and V. Yegneswaran, "An Analysis of the iKee.B iPhone Botnet," in *Security and Privacy in Mobile Information and Communication Systems*, vol. 47 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 141–152, Springer Berlin Heidelberg, 2010.