A SECURITY FRAMEWORK FOR MOBILE HEALTH APPLICATIONS ON ANDROID PLATFORM

MUZAMMIL HUSSAIN

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

A SECURITY FRAMEWORK FOR MOBILE HEALTH APPLICATIONS ON ANDROID PLATFORM

MUZAMMIL HUSSAIN

THESIS SUBMITTED IN FULFILMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

UNIVERSITY OF MALAYA
ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: **Muzammil Hussain**

Matric No: **WHA130038**

Name of Degree: **Doctor of Philosophy**

Title of ~~Project Paper/Research Report/Dissertation/~~Thesis ("this Work"):

A SECURITY FRAMEWORK FOR MOBILE HEALTH APPLICATIONS ON ANDROID PLATFORM

Field of Study: **Network Security (Computer Science)**

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name: **Prof. Dr. Miss Laiha Binti Mat Kiah**

Designation: **Professor**

ABSTRACT

The advent of smartphones dramatically changed the way of communication, computation, and the model of many services, including healthcare delivery. The adoption of smartphones in the healthcare system is rapidly growing, and enormous number of apps are being developed to monitor patient health, access patient records, test results, prescribe medications, and for numerous related purposes under the collective term of mobile Health (mHealth). These apps are readily accessible to the average user of mobile devices, and despite the potential of mHealth apps to improve the availability, affordability and effectiveness of delivering healthcare services, they handle sensitive medical data, and as such, have also the potential to carry substantial risks to the security and privacy of their users. Developers of apps are usually unknown, and users are unaware of how their data are being managed and used. This is combined with the emergence of new threats due to the deficiency in mobile apps development or the design ambiguities of the current mobile operating systems. A number of mobile operating systems are available in the market, but the Android platform has gained the topmost popularity. However, Android security model is short of completely ensuring the privacy and security of users' data, including the data of mHealth apps. Despite the security mechanisms provided by Android such as permissions and sandboxing, mHealth apps are still plagued by serious privacy and security issues. These security issues need to be addressed in order to improve the acceptance of mHealth apps among users and the efficacy of mHealth apps in the healthcare system. The focus of this research is on the security of mHealth apps, and the main objective is to propose a coherent, practical and efficient framework to improve the security of medical data associated with Android mHealth apps, as well as to protect the privacy of their users. The proposed framework provides its intended protection mainly through a set of security checks and policies that ensure protection against traditional as well as recently published threats to mHealth apps. The design of the framework comprises two layers: a Security Module Layer (SML) that implements the security-check modules, and a System Interface Layer (SIL) that interfaces SML to the Android OS. SML enforces security and privacy policies at different levels of Android platform through SIL. The proposed framework is validated via a prototypic implementation on actual Android devices to show its practicality and evaluate its performance. The framework is evaluated in terms of effectiveness and efficiency. Effectiveness is evaluated by demonstrating the performance of the framework against a selected set of attacks, while efficiency is evaluated by comparing the performance overhead in terms of energy consumption, memory and CPU utilization, with the performance of a mainline, stock version of Android. Results of the experimental evaluations showed that the proposed framework can successfully protect mHealth apps against a wide range of attacks with negligible overhead, so it is both effective and practical. Furthermore, this framework is available to other researchers for research purposes as well as for real-world deployments.

ABSTRAK

Kemunculan telefon pintar secara mendadak mengubah cara komunikasi, pengiraan, dan pelbagai model perkhidmatan, termasuk penyampaian penjagaan kesihatan. Penggunaan telefon pintar dalam sistem penjagaan kesihatan berkembang pesat, dan sejumlah besar aplikasi yang sedang dibangunkan untuk memantau kesihatan pesakit, rekod akses pesakit, keputusan ujian, menetapkan ubat-ubatan, dan untuk pelbagai tujuan berkaitan di bawah istilah kolektif Kesihatan mudah alih (mHealth). Aplikasi ini adalah mudah diakses oleh pengguna purata peranti mudah alih, dan walaupun potensi mHealth aplikasi untuk meningkatkan ketersediaan, kemampuan dan keberkesanan penyampaian perkhidmatan penjagaan kesihatan, mereka mengendalikan data perubatan yang sensitif, dan oleh itu, mempunyai juga potensi untuk membawa besar risiko kepada keselamatan dan privasi pengguna mereka. Pemaju aplikasi biasanya tidak diketahui, dan pengguna tidak tahu bahawa bagaimana data mereka diuruskan dan digunakan. Ini digabungkan dengan munculnya ancaman baru kerana kekurangan dalam pembangunan aplikasi mudah alih atau kekaburan reka bentuk sistem operasi mudah alih semasa. Beberapa sistem operasi mudah alih yang terdapat di pasaran, tetapi platform Android telah mendapat populariti yang paling atas. Walau bagaimanapun, model keselamatan Android adalah pada masa ini belum sepenuhnya mampu memastikan privasi dan keselamatan data pengguna, termasuk data aplikasi mHealth. Walaupun mekanisme keselamatan yang disediakan oleh Android seperti kebenaran dan kotak pasir, aplikasi mHealth masih berhadapan dengan isu-isu privasi dan keselamatan yang serius. Isu-isu keselamatan perlu diberi perhatian dalam usaha untuk meningkatkan penerimaan aplikasi mHealth dikalangan pengguna dan keberkesanan aplikasi mHealth dalam sistem penjagaan kesihatan. Fokus kajian ini adalah pada keselamatan aplikasi mHealth, dan objektif utama adalah untuk mencadangkan rangka kerja yang jelas, praktikal dan berkesan untuk meningkatkan keselamatan data kesihatan yang berkaitan dengan aplikasi Android mHealth, serta untuk melindungi privasi pengguna mereka. Rangka kerja yang dicadangkan memperuntukkan perlindungan yang dimaksudkan terutamanya melalui satu set cek dan dasar-dasar yang memastikan perlindungan terhadap tradisional serta ancaman baru-baru ini diterbitkan untuk aplikasi mHealth keselamatan. Reka bentuk rangka kerja terdiri daripada dua lapisan: lapisan Modul Keselamatan Layer (SML) yang melaksanakan modul keselamatan cek, dan Layer Interface System (SIL) yang mempunyai ruang kaitan SML untuk OS Android. SML menguatkuasakan dasar keselamatan dan privasi pada tahap Android Platform yang berbeza melalui SIL. Rangka kerja yang dicadangkan itu disahkan melalui pelaksanaan prototypic pada peranti Android yang sebenar untuk menunjukkan praktikal dan menilai prestasinya. Rangka kerja ini dinilai dari segi keberkesanan dan kecekapan. Keberkesanan dinilai dengan menunjukkan prestasi rangka kerja terhadap set serangan yang dipilih, manakala kecekapan dinilai dengan membandingkan overhed prestasi dari segi penggunaan tenaga, ingatan dan CPU, dengan pelaksanaan laluan utama, versi stok Android. Keputusan penilaian uji kaji menunjukkan bahawa rangka kerja yang dicadangkan berjaya boleh melindungi aplikasi mHealth daripada pelbagai serangan dengan overhead diabaikan, jadi kedua-dua ia adalah berkesan dan praktikal. Tambahan pula, rangka kerja ini disediakan kepada penyelidik lain untuk tujuan penyelidikan dan juga untuk pergerakan dunia sebenar.

ACKNOWLEDGEMENT

All Praise be to Allah, Almighty, Lord of the worlds, The Most Gracious, The Most Merciful, and peace be upon His Messenger. First and foremost, I am thankful to Almighty Allah for enabling me to complete this challenging task.

I would like to express my heartiest gratitude and appreciation to my supervisors, Prof. Dr. Miss Laiha Binti Mat Kiah, and Dr. Nor Badrul Anuar Bin Jumaat for their invaluable guidance, supervision, unparalleled patience, exemplary kindness, support, and encouragement to me throughout this research. Their keen, meaningful and kind supervision empowered me to pursue this research with commitment and dedication. I wish to extend my thanks to all those who have contributed directly or indirectly to the completion of this work or the building of my career. I would like to thank specifically Prof. Dr. Shaukat Iqbal, Associate Prof. Dr. Muhammad Shujaat Mubarik, Dr. Bilal Bahaa Zaidan, and Dr. Aws Alaa Zaidan for all they have done; their deeds to me are beyond any attempt of detailing. Especially, I am deeply indebted to Dr. Ahmed Al-Haiqi for his support, advices, guidance and consistent help on each stage throughout this research. I would also like to thank my friends for encouraging me and supporting me at every occasion. Special thanks go to Bright Sparks Program, University of Malaya, for the direct contribution to the accomplishment of this goal. I further thank University of Malaya and its entire people, in particular the friendly staff of the Faculty of Computer Science and Information Technology.

I give my deepest gratitude and genuine love to my parents for their love, care, prayers and support throughout my entire life. I have no words to thank my father; his constant support, motivation, and encouragement kept me passionate to complete this task. And of course, my mother, her never-ending prayers for me kept me safe and elevated in every sphere of life. Finally, my grateful thanks and heartfelt appreciation go to my valued brother Eng. Mussadiq Hussain for his support throughout my study.

TABLE OF CONTENTS

ABSTRACT .... **III**

ABSTRAK .... **V**

ACKNOWLEDGEMENT .... **VII**

TABLE OF CONTENTS .... **VIII**

LIST OF FIGURES .... **XII**

LIST OF TABLES .... **XIV**

LIST OF SYMBOLS AND ABBREVIATIONS .... **XV**

LIST OF APPENDICES .... **XVII**

CHAPTER 1: INTRODUCTION **1**

1.1 Introduction 1
1.2 Research Background 3
1.3 Problem Statement 5
1.4 Research Questions 8
1.5 Objectives of the Research 8
1.6 Research Scope 10
1.7 Thesis Outline/Organization 11

CHAPTER 2: LITERATURE REVIEW **13**

2.1 The Landscape of Research on Smartphone mHealth Apps 13
2.1.1 An Overview 13
2.1.2 A Taxonomy of Literature Works on mHealth Apps 15
2.1.2.1 Class 1: Review and survey articles 16
2.1.2.2 Class 2: Studies conducted on mHealth apps and their use 20
2.1.2.3 Class 3: Reports on actual attempts to develop mHealth apps ... 22
2.1.2.4 Class 4: Proposals of frameworks to develop and operate mHealth apps 23
2.1.3 Articles by Medical Specialty of Apps 24
2.1.4 Articles by Purpose and Function of Apps 24
2.1.5 Articles by Indexing Databases 26
2.1.6 Motivations for Smartphone mHealth Apps 26
2.1.7 Challenges to mHealth Apps 28
2.2 The Android Platform and Its Security 32
2.2.1 Android System Architecture 32
2.2.2 The Structure of Android Apps 36
2.2.2.1 Activity 37
2.2.2.2 Service 38

2.2.2.3 Content provider 38
2.2.2.4 Broadcast receiver 38
2.2.3 Android Security Model 39
2.2.3.1 Android permission system 39
2.2.3.2 Application sandboxing 40
2.2.3.3 Application signing 40
2.2.3.4 Secure inter-process communication 40
2.2.3.5 SELinux 40
2.2.4 Android Security Research Trends 41
2.2.5 Issues in Android Security 49
2.3 Security and Privacy of mHealth Apps 51
2.3.1 Threats to mHealth Apps 51
2.3.2 An Empirical Assessment of mHealth Apps' Security 58
2.3.3 Existing Solutions for mHealth Apps' Security 59
2.4 Chapter Summary 63

CHAPTER 3: RESEARCH METHODOLOGY 64

3.1 Phase I: Preliminary Study 64
3.2 Phase II: Framework Design 66
3.3 Phase III: Prototype Implementation 67
3.4 Phase IV: Evaluation 69
3.5 Chapter Summary 70

CHAPTER 4: THE DESIGN OF "MHEALTH APPS SECURITY FRAMEWORK" 71

4.1 MASF Overall Architecture 71
4.2 MASF Layered Components 73
4.2.1 Security Module Layer 74
4.2.1.1 Security checkers 74
4.2.1.2 SML manager 81
4.2.1.3 Policy database 84
4.2.1.4 User interactor 84
4.2.1.5 Action performer 84
4.2.2 System Interface Layer 87
4.2.3 Other External Components 88
4.2.3.1 Android operating system 89
4.2.3.2 Apps 89
4.2.3.3 App store 89
4.2.3.4 Users 90
4.2.4 MASF Policies 90
4.2.4.1 Policy restrictions 92
4.2.4.2 Policy conditions 93
4.2.4.3 Policy definition 97
4.3 Framework Operation 97
4.3.1 Use Case I: Installation of a New App 97

4.3.2 Use Case II: Privacy Enhanced Content Providers and System Services 98
4.3.3 Use Case III: Context-Aware Fine-Grained Access Control 100
4.3.4 Use Case IV: Mitigating the DMB Attacks when Connecting to Devices 101
4.4 Chapter Summary 103

CHAPTER 5: IMPLEMENTATION AND EVALUATION 105

5.1 Implementation Details 105
5.1.1 Implementation of the System Interface Layer 106
5.1.1.1 Kernel space 107
5.1.1.2 Middleware layer 108
5.1.1.3 Application layer 109
5.1.2 Implementation of the Security Modules Layer 110
5.1.3 Enforcement of Fine-Grained Access Control Policies 112
5.1.3.1 Context-aware access control 113
5.1.3.2 One-time checks 113
5.1.3.3 Permission management 114
5.1.3.4 Data shadowing 114
5.1.3.5 Installation checks 115
5.1.3.6 Intent management 115
5.1.3.7 Managing system peripheral state 116
5.2 Experimental Evaluation 116
5.2.1 Experiment 1: Effectiveness 117
5.2.2 Experiment 2: Malware Test Suite 121
5.2.3 Experiment 3: Impact of Permission Restrictions 122
5.2.4 Experiment 4: Impact of Data Shadowing 123
5.2.5 Experiment 5: Impact of Disabling/Blocking Intents 124
5.2.6 Experiment 6: Impact of Enabling/Disabling System Peripherals 125
5.2.7 Experiment 7: ICC False Positives 126
5.2.8 Analysis of the Impact on Android Security 128
5.3 Performance Evaluation 130
5.3.1 SML Performance Overhead 130
5.3.1.1 Performance overhead of permission checks 130
5.3.1.2 Performance overhead of context checks 133
5.3.1.3 Java microbenchmark 134
5.3.1.4 Macrobenchmarks 137
5.3.1.5 System memory overhead 139
5.3.2 SIL performance overhead 141
5.4 Chapter Summary 142

CHAPTER 6: CONCLUSIONS AND FUTURE WORK 143

6.1 Research Summary and Objectives Achievement 143
6.2 Contribution of the Research 146
6.3 Research Limitations 148

6.4 Future Work 149

REFERENCES.. 151

LIST OF PUBLICATIONS 175

APPENDICES 177

LIST OF FIGURES

Figure 2.1: A Taxonomy of Research Literature on Smartphone mHealth Apps 17

Figure 2.2: Number of Included Articles by the Specialty of Apps They Cover 25

Figure 2.3: Number of Included Articles by the Purpose or Function of Apps They Cover 25

Figure 2.4: Number of Included Articles in Different Categories by the Source Digital Database 26

Figure 2.5: Android System Architecture 35

Figure 2.6: Android App Components and Their Interactions 37

Figure 2.7: Taxonomy of Literature on Android Security 41

Figure 2.8: A Three-Dimensional Model for Classifying mHealth Apps in Terms of Security and Privacy Concerns (Plachkinova et al., 2015) 57

Figure 2.9: A Taxonomy of mHealth Apps - Security and Privacy Concerns (Plachkinova et al., 2015) 58

Figure 2.10: Security and Privacy Analysis of Sample mHealth Apps 59

Figure 2.11: Static Analysis System Design Framework 62

Figure 3.2: Conceptual Framework of the Research 65

Figure 4.1: The Overall Architecture of the Proposed Framework MASF 73

Figure 4.2: Data Leakage Detection with Static Analysis 79

Figure 4.3: Static Taint Analysis System Design Framework 80

Figure 4.4: System Interface Layer 88

Figure 4.5: Use Case 1 - Installation of a New App 98

Figure 5.1: Middleware Security Module Structure 111

Figure 5.2: Security Assessment of a 100 mHealth Apps on Stock Android OS 118

Figure 5.3: Number of Apps Attempting to Access Various Resources Containing Sensitive Data 118

Figure 5.4: Impact of Permission Revoking on Applications 123

Figure 5.5: Impact of Data Shadowing on mHealth Apps 124

Figure 5.6: Impact of Intent Disabling on Apps 125

Figure 5.7: Impact of Enabling/Disabling System Peripherals................................ 126

Figure 5.8: Denied ICCs by Different Policy Checks... 127

Figure 5.9: Time Overhead of MASF Permission Check System 131

Figure 5.10: Energy Overhead of MASF Permission Check System 133

Figure 5.11: Comparison of Device Battery Consumption while Checking for Context Updates 134

Figure 5.12: The Result of CaffeineMark 3.0 Benchmark / Microbenchmark of Java Overhead. 136

Figure 5.13: Total Memory Overhead Comparison with and without MASF Policy Restrictions 140

Figure 5.14: Relative Cumulative Frequency Distribution of Microbenchmarks in Stock Android vs SIL 142

LIST OF TABLES

Table 1.1: Research Questions Mapped to the List of Objectives and Contributions .. 9

Table 2.1: A Summary of Existing Android Security Proposals in the Literature 45

Table 2.2: Description of Attack Surfaces (He et al., 2014) 54

Table 2.3: Recommendations to Consumers and Application Developers (Adhikari et al., 2014) 61

Table 4.1: Sample Installation-Time Policy Rules 95

Table 5.1: Malware Test Suite 121

Table 5.2: Macrobenchmark Results of Time Overhead for Modified Core Android Methods 137

Table 5.3: Weighted Average Performance Overhead of Executing Hooked Functions in Stock Android and in SIL. The Margin of Error is given for the 95% Confidence Interval. 142

LIST OF SYMBOLS AND ABBREVIATIONS

API	Application Programming Interface
APP(S)	Application(s)
ASF	Android Security Framework
CPU	Central Processing Unit
DMB	Device Mis-Bonding
DVM	Dalvik Virtual Machine
ECG	Electrocardiogram
EMA	Ecological Momentary Assessment
FDA	Food and Drug Administration
GNU	GNU's Not Unix
GPS	Global Positioning System
HIPPA	Health Insurance Portability and Accountability Act
ICC	Inter Component Communication
IDE	Integrated Development Environment
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
IPC	Inter-Process Communication
IRM	Inlined Reference Monitors
LSM	Linux Security Module
MAC	Mandatory Access Control
MASF	MHealth Apps Security Framework
mHealth	Mobile Health
MMS	Multimedia Messaging Service
NDK	Native Development Kit
NFC	Near Field Communication
OS	Operating System
PDAs	Personal Digital Assistants
PHI	Personal Health Information
PHR	Personal Health Records
PII	Patient Identity Information
PL	Programming Language
RPC	Remote Procedure Call
SD	Secure Digital
SDK	Software Development Kit
SELinux	Security-Enhanced Linux
SIL	System Interface Layer
SML	Security Module Layer
SMS	Short Message Service
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UID	User ID

URI	Uniform Resource Identifier
URL	Uniform Resource Locator
Wi-Fi	Wireless Fidelity
XML	Extensible Markup Language

LIST OF APPENDICES

APPENDIX A :	Sample Set of Applications	177
APPENDIX B :	List of System Interface Layer Functions	182

CHAPTER 1: INTRODUCTION

1.1 Introduction

The advent of smartphone dramatically changed the way of communication, computation, and the model of many traditional and new services, e.g., healthcare and entertainment. In the early days, mobile phones were only used for making phone calls. Nowadays mobile phones have come to be known as smartphones because of their increasing functions and intelligence. Smartphones are equipped with powerful operating systems that enable users to install additional software, more storage and processing capabilities, and multiple options of network connectivity. Due to their improved functionalities and computing capabilities, smartphones are increasingly viewed as handheld computers (M. N. Boulos, Wheeler, Tavares, & Jones, 2011), and their adoption by people is arising due to their ease of use (Y. Park & Chen, 2007).

Among the available smartphone Operating Systems (OS) in the market, Android OS has the topmost popularity, with a market share of above 87.6% (International Data Corporation, 2016), and more than 1.5 million apps available on Google Play ("Number of Android applications," 2015). Categories of those apps range from the basic *trivia* game apps to serious business and financial applications. One active area of smartphone apps that has witnessed an astonishing growth is the healthcare system. Under the category of *medical* apps, Google Play and similar online stores of smartphone apps are providing large collections of apps that can be used for various healthcare-related functions. Adopting the notion of *mobile Health* (mHealth) as a reference to the use of mobile devices in medicine and public health, smartphone medical apps are referred to in this thesis as *mHealth apps.* mHealth apps are hereby defined as software programs that provide health related services through smartphones and tablets.

mHealth apps have the potential to improve the availability, affordability and effectiveness of healthcare services for patients (Mirza, Norris, & Stockdale, 2008). They have become incorporated into the health informatics field as tools that maintain a patient-centred model of healthcare by enabling users to monitor their health related problems, attain personal fitness goals, and understand specific medical conditions. Patients can use smartphones to access and update their medical records, monitor their health, and to view their prescriptions as well (Brennan, Downs, & Casper, 2010). Physicians, on the other hand, can use smartphones to access patient records and test results, monitor patient health and to prescribe medications (Burdette, Herchline, & Oehler, 2008; Luxton, McCann, Bush, Mishkind, & Reger, 2011; Ozdalga, Ozdalga, & Ahuja, 2012). mHealth apps can also improve the way in which physicians interact with patients and provide healthcare services.

Similar to other new trends, mHealth apps have to face a number of challenges despite their compelling benefits. The sensitive nature of these apps' purpose and consequence of use –in relation to human health– impose several questions about their reliability, authority, and compliance to regulations. Aside from the functional requirements, issues related to non-functional requirements have also to be addressed, such as the usability of the apps by users from different age groups. In particular, it soon became clear that mHealth apps carry substantial risks to the security of user's sensitive medical data as well as their privacy (Adhikari, Richards, & Scott, 2014; Dehling, Gao, Schneider, & Sunyaev, 2015; Gill, Kamath, & Gill, 2012; He, Naveed, Gunter, & Nahrstedt, 2014; Plachkinova, Andrés, & Chatterjee, 2015; Y. Zhou & Jiang, 2012). Developers of these apps are usually unknown, and users are unaware of how their data are being managed and used. In mHealth, users can easily enhance the functionalities of their smartphones by connecting them to external devices, such as medical devices, sensors and credit card readers. This introduces many new threats along with the useful applications in various domains, including healthcare information systems and retail (Anokwa, Ribeka, Parikh, Borriello, & Were, 2012; Avancha, Baxi, & Kotz, 2012; Istepanian, Laxminarayan, & Pattichis, 2006; Murthy & Kotz, 2014; Naveed, Zhou, Demetriou, Wang, & Gunter, 2014).

It should be noted that in addition to the traditional threats found in other software and information systems, mHealth apps introduce new security and privacy threats to mobile computing (He et al., 2014). Even when compared to other health information systems, mHealth apps are different in various perspectives. First, mHealth apps have the potential to collect larger amounts of data from patients because mobile devices are always carried by the patients and can collect data over long time intervals. Second, mHealth apps collect much broader range of data besides physiological measurements and direct medical data; this includes patient activities, location, lifestyle, social interactions, diet details, eating habits and so on. Third, the nature of communication between the patient and healthcare professionals is different (He et al., 2014); e.g. healthcare professionals can remotely access and monitor patients' health conditions.

Motivated by the previous facts and observations, the focus of this research is specifically the security of mHealth apps. This thesis aims to improve the security of medical data associated with Android mHealth apps, as well as to protect the privacy of users from threats that might be imposed by such apps.

1.2 Research Background

The main theme of this thesis is the security and privacy of mHealth apps on Android smartphones. This theme involves three main research components: the concept of mHealth apps, the security of Android smartphones, and the incorporation of mHealth apps' security within Android security model. The first two ingredients are themselves separate research fields, while the third element –the focal point of this research– is an emergent field with very few recent contributions. This section introduces these research components briefly, while more elaborate background is provided in the next chapter.

Mobile health is a medical and public health practice using mobile devices, such as smartphones, personal digital assistants (PDAs), patient monitoring devices and other wireless devices (Organization, 2011). mHealth is an emerging field which has the potential to make healthcare professionals more efficient, increase patient satisfaction and reduce the healthcare cost. The general concept of mHealth includes medical apps. There are several types of medical apps, some are using external devices such as medical sensors, and some apps are using smartphone resources, such as the camera for the treatment of the patient. The use of mHealth apps among physicians and patients has grown significantly since the introduction of mobile phones. Physicians can access patients' data and medical knowledge at the point of care, and they can also monitor patient health through mHealth apps.

Android is an operating system based on Linux for mobile devices. Android platform provides a rich application framework that allows developers to build innovative apps in the Java language environment. Android is a multi-user system in which each app is considered an individual user, and is given a unique user ID (UID). Every app runs in its own Linux process and uses a separate virtual machine to be isolated from other apps. In this way, Android platform implements the principle of *least* privilege. That is, each app, by default, can only access those components that are required to do its own work. In order to protect user data, system resources (including the network) and apps themselves, Android platform provides the following extra security features: security at the OS level through the Linux kernel's secure inter-process communication (IPC), application sandbox, application signing, and the Android permission model. The details of these security features are discussed in Chapter 2.

Recently, researchers have been actively involved in the study of mHealth apps, in particular their security and privacy. For example, Mitchell et al. (2013) investigated the security and privacy challenges of mHealth apps; He et al. (2014) raised the security concerns of Android mHealth apps; and Plachkinova et al. (2015) proposed a taxonomy of mHealth apps' security and privacy concerns. Nevertheless, beyond the identification and investigation of the problem itself, there is no actual solution for the security and privacy of mHealth apps specifically, except one policy framework (Mitchell et al., 2013). This framework provides some guidelines to secure mHealth apps; however, these policies are not enough and even not implemented to secure mHealth apps. In addition, Android-provided security features are still insufficient to protect user data against few security attacks that are equally applicable to mHealth apps and their data, such as side channel threats, privilege escalation attacks, sensors-based covert channels and DMB attacks (A. Al-Haiqi, Ismail, & Nordin, 2014; Davi, Dmitrienko, Sadeghi, & Winandy, 2011; He et al., 2014; Naveed et al., 2014).

mHealth apps are a new and revolutionary development in healthcare system, and a huge number of people can access this new system at a very low cost. Considering the great utility and impact of this phenomenal development, and the detrimental effect that security and privacy issues might cause to its successful deployment, those issues need to be addressed to improve mHealth apps' effectiveness and alleviate any barriers to their rapid integration into the healthcare system.

1.3 Problem Statement

Using smartphone apps in the delivery of healthcare is rapidly proliferating. mHealth apps have several potentials that drive this popularity, including the ability to increase patient satisfaction, improve doctor efficiency, and reduce the cost of healthcare (Bishop, 2013). There is still no regulatory protection for mHealth apps similar to that available for traditional health sectors, including PC-based electronic health. For example, the Health Insurance Portability and Accountability Act (HIPPA) is not yet widely applied to mHealth apps (Plachkinova et al., 2015). Similarly, the Food and Drug Administration (FDA) intends to apply its regulatory oversight to only those apps that turn smartphones into medical devices and whose functionality can pose risk to patients' safety if not functioning as intended, which is only a subset of all mHealth apps (Food & Administration, 2015). Several recent studies showed that the lack of standardization, guidelines, security and privacy of user data are the main barriers to the widespread use of mHealth apps (Adhikari et al., 2014; He et al., 2014; Kharrazi, Chisholm, VanNasdale, & Thompson, 2012; Mitchell et al., 2013; Plachkinova et al., 2015).

mHealth apps face the usual security challenges of enforcing confidentiality, integrity, and availability via authentication, authorization, and access control (Adhikari et al., 2014; Dehling et al., 2015; He et al., 2014; Mitchell et al., 2013; Plachkinova et al., 2015). Such protection is necessary to facilitate the adoption of these apps by the healthcare system. Users of mHealth apps are also susceptible to privacy threats, such as identity theft, disclosure threats, privilege escalation attacks and side channel threats, among others (Davi et al., 2011; He et al., 2014; Kotz, 2011; Plachkinova et al., 2015). Leakage of information is a major challenge for mHealth apps (Dehling et al., 2015), where these apps may leak information in numerous ways. For example, apps usually declare their components as public (He et al., 2014), so malicious apps can easily access their information. Besides, apps usually store unencrypted data on smartphone external storage (He et al., 2014; McCarthy, 2013; Mitchell et al., 2013), so any app that has the permission to access external storage can easily access the user's data. Usage of third party services and sharing of information with social networks or other third parties are also raising threats to mHealth apps (Adhikari et al., 2014; Dehling et al., 2015; He et al., 2014; Plachkinova et al., 2015). In addition, mHealth apps use external devices to enhance the functionality of the phone. These devices also impose serious threats to users data, such as external-Device MisBonding (DMB) attacks that include data-stealing and data-injection attacks (Naveed et al., 2014), since Android permission system does not provide permission-based protection for external devices and sensors.

Existing smartphone operating systems, particularly Android, are not sufficient to ensure privacy and security of users' data, particularly in the case of mHealth apps. One major issue in the security model of Android is that the permission mechanism is too coarse-grained and the user might not be aware of the full implications when granting permissions to apps (Y. Zhou, Zhang, Jiang, & Freeh, 2011).

Based on the above facts, there is a need for a better solution to protect the security of mHealth apps, and ensure the confidentiality, integrity and availability of their data. Data associated with mHealth apps are particularly of sensitive nature, and unauthorized leakage or manipulation of these data do not only threaten the privacy of the patients, but might threaten their health or even lives. The intended protection is two-way; meaning it protects the mHealth app and its corresponding data from potential threats on the system, and also protects the system and its resources from installed mHealth apps that can unintentionally or otherwise bring new threats by means of poor design, or ill will. The focus of this thesis is to propose such a solution in the form of a security framework for mHealth apps on Android platform. The proposed framework ought to address the aforementioned security and privacy issues on Android, with a special focus on threats associated with mHealth apps, such as the revealed vulnerabilities in literature, including information leakage; and the published attacks, such as DMB, privilege escalation, and side-channel attacks.

1.4 Research Questions

This research focuses on data security and on privacy issues involved in using mHealth apps within healthcare systems, and proposes a security framework for mHealth apps on Android. The following research questions have been posed to set the direction for this research:

(i) What are the known privacy and security issues associated with using mHealth apps on current smartphones, particularly the Android platform?

(ii) What are the state-of-the-art threats to medical data in the context of mHealth apps on Android?

(iii) Is the original security design of the Android platform capable enough of securing highly diverse and fast-evolving Android-based mHealth apps?

(iv) What are the currently available security solutions for securing Android mHealth apps and protecting their data?

(v) What are the requirements of a security framework for Android mHealth apps?

(vi) How can a security framework resolve the existing security problems of mHealth apps?

(vii) What are the tools needed to implement and evaluate the proposed framework?

(viii) How can we evaluate and analyse the proposed framework?

1.5 Objectives of the Research

The overall objective of this research is to improve the situation of mHealth apps in terms of a practical and implementable security framework on the Android platform. This general objective can be broken down into the following list of detailed objectives:

(i) To investigate the security issues associated with mHealth apps as well as with the Android platform.

(ii) To examine security solutions that are specifically designed for mHealth apps and highlight their weaknesses, so as to help identifying the desired requirements for a better security solution.

(iii) To design a security framework to handle mHealth apps and protect their security, incorporating new security checks on the installation and operation of the apps. The design of this framework is based on the previous analysis.

(iv) To implement the proposed mHealth apps security framework, building a custom Android image that is deployable on a real device.

(v) To evaluate the proposed framework in terms of effectiveness and efficiency. Effectiveness is evaluated by demonstrating that the framework can successfully protect the system from a particular set of attacks, while efficiency is evaluated by measuring the performance overhead in terms of energy consumption, memory and CPU utilization.

Table 1.1: Research Questions Mapped to the List of Objectives and Contributions

Research Questions	Objective	Contribution	Chapter
What are the data privacy and security issues associated with using mHealth apps?	i	i	1 &2
What are the state-of-the-art threats to medical data in mHealth apps?	i	ii	2
Is the original security design of Android OS capable enough to secure highly diverse and fast-evolving Android mHealth apps?	i	-	1 & 2
What are the existing security solutions to secure Android mHealth apps?	ii	-	2
What are the requirements of a security framework for Android mHealth apps?	ii	iii	1 & 3 & 4
How can a security framework resolve the existing security problems of mHealth apps?	iii	iv	4
What are the tools needed to implement and evaluate proposed framework?	iv	v	3 & 5
How can we evaluate and analyse the proposed framework?	v	v	5

Table 1.1 maps the research questions **set forth** in the previous section to the above set of targeted objectives, along with the corresponding actual contributions, which are to be stated later in Chapter 6. This table includes as well the respective chapters in which those contributions are presented and discussed.

1.6 Research Scope

Several assumptions and design selections restrict the scope of the research work within this thesis. The following points list those restrictions:

(i) This research only considers mHealth apps out of the available kinds of apps. For example, it does not include the finance, education, social and other categories, though the same solution would be feasible as well.

(ii) Because Android platform is most popular and open source, it was decided to work on Android OS out of the available smartphone OSs.

(iii) Android 4.3 Jelly Bean has been used to implement the proposed framework.

(iv) Android middleware and the underlying Linux kernel are considered as trusted base, and assumed as not been maliciously designed.

To put the research focus in perspective, Figure 1.1 depicts the scope, where the shaded area is the narrow focus of the thesis.

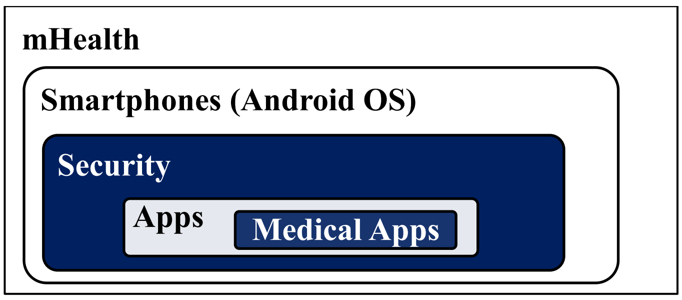


Figure 1.1: Area of Research

1.7 Thesis Outline/Organization

The current chapter is an introduction to the work to be presented throughout the thesis, including the main motivations, research background, the specific problem statement to be addressed, and the main research questions to be answered. This chapter also sets the objectives to be accomplished and maps those objectives to the research questions. The scope of the research is also described based on the problem.

Altogether, this thesis is composed of six chapters. The rest of the thesis is organized as follows:

Chapter 2: Literature Review

This chapter is divided into three main section. First section provides the necessary background of research on mHealth apps and its related areas. A thematic taxonomy is proposed that compactly describes the research on mHealth apps and defines different directions in this field. The second section of this chapter provides an essential background on Android OS and its security mechanism, and it also reviews recent research trends on Android security. Finally, the solutions proposed in the literature to protect the security and privacy of mHealth apps are provided in the third section of this chapter.

Chapter 3: Research Methodology

This chapter outlines the general research methodology adopted in this research study. This methodology is expressed in terms of a conceptual framework that consists of four phases: a preliminary study, the proposed framework's design, a proof-of-concept prototypic implementation, and finally the evaluation. These four phases are briefly described alongside the methods followed in each phase.

Chapter 4: The Design of "mHealth Apps Security Framework"

This chapter presents the concrete design that was generated to achieve the main objective of the research. It outlines and describes the design of the proposed framework, starting from the overall architecture, throughout the individual layers and their components, up to the discussion of few use cases that are representative of the typical operation of the proposed framework.

Chapter 5: Implementation and Evaluation

This chapter presents the results of evaluating a prototypic implementation of the proposed framework. The built implementation is meant to serve as a proof-of-concept that validates the design in the previous chapter and provides an initial seed for further deployments. After describing the implementation choices, this chapter aims to evaluate and analyse the prototype in terms of effectiveness and efficiency. The effectiveness measures the performance and usefulness of the proposed framework in satisfying its purpose of securing users' privacy and protecting their sensitive data. These are evaluated through a set of experiments that are described in the chapter. Another set of experiments measure performance metrics (CPU utilization, memory usage and energy consumption) in order to evaluate the efficiency of the framework in performing its function. In particular, the focus is on the overhead imposed by the framework on the normal operation of the system.

Chapter 6: Conclusions and Future Work

This chapter concludes the thesis by presenting the summary of this research and reporting on the re-examination of the research objectives. Moreover, it lists the main findings of this research work, highlighting the significance of the proposed solution. This chapter also states the limitations of this research study and proposes future directions to improve the produced solution and avoid some of its limitations.

CHAPTER 2: LITERATURE REVIEW

This chapter sets the stage for later chapters by providing necessary background information on the concepts of mHealth apps, the Android platform and the security issues at the intersection of both. The chapter is divided into three major sections. Section 2.1 is first providing a comprehensive literature survey on mHealth Apps, and its related areas. Second, Section 2.2 is providing a complete background on the Android architecture, and what has been done to secure this platform. The third and most important section (Section 2.3) reviews the most relevant works in the literature on the threats to mHealth apps, and provides a critical assessment of their security and privacy. This section also summarizes the existing solutions to address those issues. The focus of the section is directly related to the research in this thesis, which attempts to contribute a novel solution to the said issues.

2.1 The Landscape of Research on Smartphone mHealth Apps

This section provides the necessary background about the research on mHealth apps, how rapidly this field is growing, and what are the main highlights in this new trend of mobile healthcare systems. It surveys the efforts of researchers in response to the new and disruptive technology of smartphone mHealth apps, mapping the research landscape form the literature into a coherent taxonomy, and finding out basic characteristics of this emerging field.

2.1.1 An Overview

Adoption of smartphones in the arsenal of healthcare is coming as no surprise. People have always used available facilities to enhance their most important activities and protect their most valuable assets; and no asset is more valuable than their own health. The utilization of information and communication technology in the practice of healthcare introduced the notion of eHealth, where telecommunications is enabling telemedicine, computers are processing health data, and the Internet is providing the infrastructure to exchange all sorts of medical information and services. When mobility became possible, telecommunications occurred through mobile phones, and computers moved along with people in the form of portable laptops and then handheld devices. The eHealth stretched to include mobile health (mHealth); but still, the phone was a phone and the computer was a computer; until both converged into a single unit known as a “smartphone”.

Smartphones are mobile devices that are *smarter* than earlier generations of cellular phones, usually known as feature-phones. This extra smartness is gained by virtue of closer resemblance to personal computers (PCs). Smartphones possess greater computing power, more connectivity options, sophisticated operating systems, full Internet access, and most importantly the ability to install and run third-party applications, often dubbed as "apps". This last feature extended the smartphone’s versatility into new functions unthought-of before, even by its designers.

However, smartphones are not just scaled down versions of their PC relatives; they depart from traditional PCs in several ways. They are portable, even beyond the portability of laptops, and they are meant to be mobile and used on the move. This introduces the notion of context to smartphones, in terms of location, ambient, and user actions. Smartphones can measure these variables via onboard sensors, such as accelerometers and gyroscopes, which are unique to smartphone platforms. Smartphones also enjoy the ultimate connectivity among computing devices, with multiple wireless interfaces to cellular networks, Wi-Fi access points, Bluetooth peripherals, up to the latest innovations of Wi-Fi Direct and the Near Field Communication (NFC) technologies.

Being this disruptive, smartphones are also the most personal computers so far. They are carried everywhere, and used to run all sort of functions, most of which are intimate to the users. In the context of healthcare, the trend of seeking health information from the

Internet is an obvious option on mobile platforms, but the real change came through the surge of apps written by developers to serve a wide variety of medical and healthcare scenarios, such as health education, intervention and adherence enhancement, as well as medication and diagnosis. Apps targeted both health professionals, patients, and the public, in the form of medical references, calculators, through the way to being attachments or alternatives to medical devices. In essence, what physicians and patients had to access on stationary computers have been brought to them by apps right onto their hands/pockets, augmented by innovative use of the new sensing capabilities that required previously special equipment, external to the computing device.

The unique characteristic of mHealth, and particularly that based on smartphone apps is that it has grown very fast, outpacing the governmental efforts in regulation, as well as the health informatics researchers in study and evaluation. It is not feasible to review, let alone evaluate, the 100000 medical-related apps available online for the major smartphone platforms (Jahns, 2014), but those apps are actually open in the wild for download and use by healthcare professionals as well as the public. Apps are stored centrally in web-based repositories called app stores, a one-stop-shop fashion for marketing apps. The most popular smartphones today, with a market share of 87.6% and 11.7% respectively (International Data Corporation, 2016) are the Android (Google, 2016) and iOS (Apple, 2016b) supported-phones; their corresponding online markets are Google Play (Google, 2016) and Apple Apps Store (Apple, 2016a), respectively.

2.1.2 A Taxonomy of Literature Works on mHealth Apps

A comprehensive survey of research on mHealth apps was conducted, referring to a number of online databases, including ScienceDirect, Web of Science, IEEE Explore, and PubMed by using the following query string: ("health apps" OR "medical apps" OR "medical smartphone apps" OR "health smartphone apps" OR "healthcare apps" OR "healthcare smartphone apps"). This survey resulted in 133 articles that were read thoroughly in the main purpose of finding out a general map for the conducted research on this emerging topic. Most of the articles (51.13%; 68/133) are review and survey papers that refer to actual apps or to the literature in order to describe the existing mHealth apps for a specific specialty, disease, or purpose, or to provide a general overview of the new trend. The next largest portion of articles (32.33%; 43/133) conducted various studies, ranging from seeking to evaluate samples from the flowing current of mHealth apps to exploring the desired features that people would like to have in their newly found helper tools. Quite a few researchers (12.78%; 17/133) moved along the new wave and presented actual attempts to develop their own mHealth apps, or shared their experiences in doing so. The final and smallest portion of works (3.76%; 5/133) included proposals for frameworks or models that address the operation of apps or their development in the more general setting. Observing these patterns, the general categories of research articles can be captured, and then the classification can be refined into the literature taxonomy shown in Figure 2.1. It is possible to distinguish between several subcategories in the main classes, though overlaps do happen. In the following subsections, the observed categories are listed, making simple statistics throughout the discussion.

2.1.2.1 Class 1: Review and survey articles

It comes as no surprise that the earliest and most research works on mHealth apps are review articles that aimed to capture the new phenomena, introduce it to the medical community, and derive some descriptive statistics, trying to understand the implications and potentials along the way. The easiest and largest class to notice is the reviews based on a specific specialty or disease (Al-Hadithy & Ghosh, 2013; Arnhold, Quade, & Kirch, 2014; Aungst, 2013; Baheti & Toshniwal, 2014; Bender, Yue, To, Deacken, & Jadad, 2013; Bhansali & Armstrong, 2012; T. Carter, O'Neill, Johns, & Brady, 2013; Cheng, Chakrabarti, & Kam, 2014; Chhablani, Kaja, & Shah, 2012; Connor, Brady, de Beaux, &

Tulloh, 2013; Dala-Ali, Lloyd, & Al-Abed, 2011; Derbyshire & Dancey, 2013; Deveau & Chilukuri, 2012; Donker et al., 2013; Dubey et al., 2014; Elias, Fogger, McGuinness, & D'Alessandro, 2014; Eng & Lee, 2013; Franko, 2012; Goff, 2012; Gomez-Iturriaga, Bilbao, Casquero, Cacicedo, & Crook, 2012; Goyal & Cafazzo, 2013; Kalz et al., 2014; Khatoon, Hill, & Walmsley, 2013; Kraidin, Ginsberg, & Solina, 2012; H. Lee et al., 2014; Lewis, 2013; Lippman, 2013; Milani et al., 2014; Mohan & Branford, 2012; Moodley, Mangino, & Goff, 2013; Muessig, Pike, LeGrand, & Hightow-Weidman, 2013; Nwosu & Mason, 2012; O'Neill, Holmer, Greenberg, & Meara, 2013; O'Neill & Brady, 2012; Oehler, Smith, & Toney, 2010; Pandey, Hasan, Dubey, & Sarangi, 2013; Robinson & Jones, 2014; Singh, 2013; Slaper & Conkol, 2014; Sondhi & Devgan, 2013; D. J. Stevens, Jackson, Howes, & Morgan, 2014; Tripp et al., 2014; Wallace & Dhingra, 2013; Wang et al., 2014; Warnock, 2012; Workman & Gupta, 2013; Yoo, 2013) (47/68 articles).

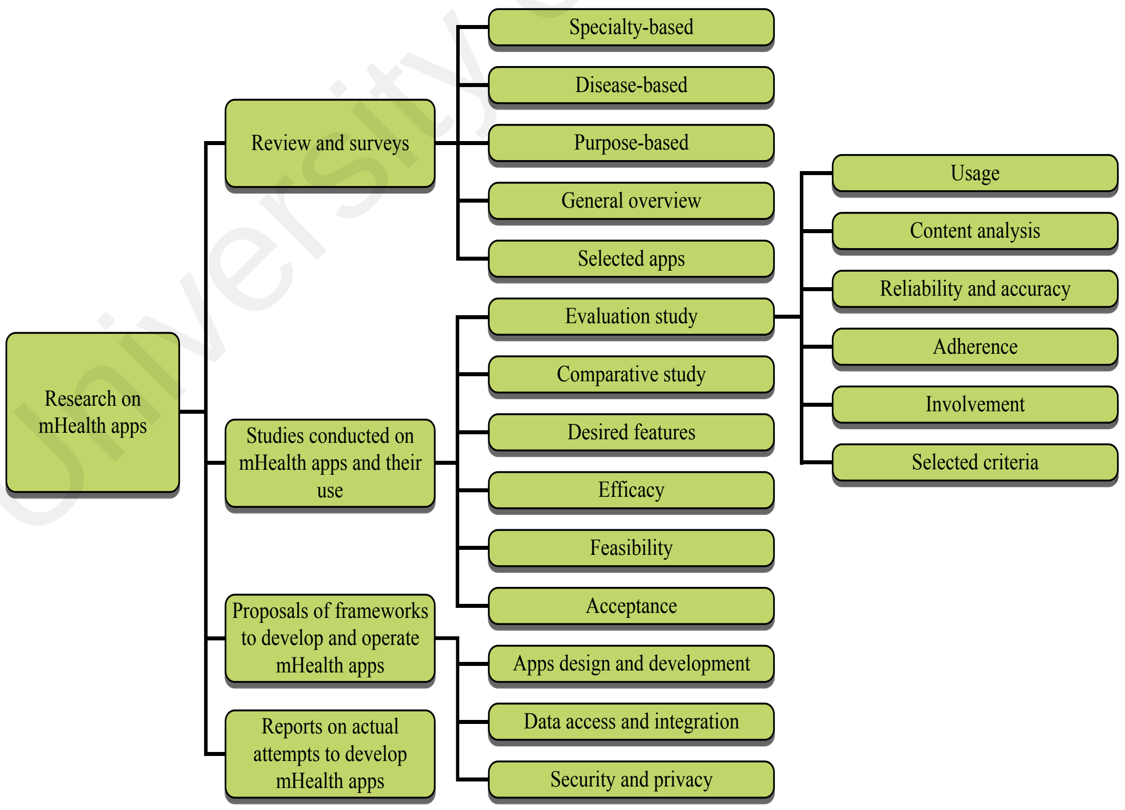


Figure 2.1: A Taxonomy of Research Literature on Smartphone mHealth Apps

Examples of this category include the reviews of apps on Anaesthesia (Bhansali & Armstrong, 2012; Connor et al., 2013; Glassenberg, De Oliveira, Glassenberg, & McCarthy, 2013; Kraidin et al., 2012; Morris, Javed, Bodger, Gorse, & Williams, 2013), Surgery (T. Carter et al., 2013; Dala-Ali et al., 2011; Edlin & Deshpande, 2013; Franko, 2012; O'Neill et al., 2013; D. J. Stevens et al., 2014; Warnock, 2012), Plastic surgery (Al-Hadithy & Ghosh, 2013; Mohan & Branford, 2012; Morris et al., 2013; Workman & Gupta, 2013), Oncology (Bender et al., 2013; Gomez-Iturriaga et al., 2012; Lewis, 2013; Min et al., 2014; Pandey et al., 2013; C. S. Xu, Anderson, Armer, & Shyu, 2012), Palliative medicine (Nwosu & Mason, 2012; B. Rosser & C. Eccleston, 2011; B. A. Rosser & C. Eccleston, 2011; Wallace & Dhingra, 2013), Ophthalmology (Cheng et al., 2014; Chhablani et al., 2012), Dentistry (Baheti & Toshniwal, 2014; Khatoon et al., 2013; Singh, 2013), Pharmacy (Aungst, 2013; Dayer, Heldenbrand, Anderson, Gubbins, & Martin, 2013; Haffey, Brady, & Maxwell, 2013, 2014), Psychiatry (Dennison, Morrison, Conway, & Yardley, 2013; Donker et al., 2013; Elias et al., 2014; Kuhn et al., 2014; Shand, Ridani, Tighe, & Christensen, 2013; Zhu, Liu, & Holroyd, 2012), Paediatrics (Goldbach et al., 2013; Hawkes, Walsh, Ryan, & Dempsey, 2013; Ho et al., 2014; Peck, Stanton, & Reynolds, 2014; Rozenblyum, Mistry, Cellucci, Martimianakis, & Laxer, 2014; Slaper & Conkol, 2014; Sondhi & Devgan, 2013; Wackel, Beerman, West, & Arora, 2014; Wearing, Nollen, Befort, Davis, & Agemy, 2014), Infectious Diseases (Burdette, Trotman, & Cmar, 2012; Goff, 2012; Moodley et al., 2013; Muessig et al., 2013; Oehler et al., 2010; Robustillo Cortés, Cantudo Cuenca, Morillo Verdugo, & Calvo Cidoncha, 2014; Spain, 2014; Visvanathan, Hamilton, & Brady, 2012; Yoo, 2013), Public health (Abroms, Lee Westmaas, Bontemps-Jones, Ramani, & Mellerson, 2013; Arnhold et al., 2014; Årsand et al., 2012; Azar et al., 2013; Bender et al., 2013; BinDhim, Freeman, & Trevena, 2014; Breland, Yeh, & Yu, 2013; Breton, Fuemmeler, & Abroms, 2011; M. C. Carter, Burley, Nykjaer, & Cade, 2013; Choi, Noh, & Park, 2014; Cohn, Hunter-Reel, Hagman, & Mitchell, 2011; Dunton et al., 2014; V. Gay & Leijdekkers, 2012; Goyal & Cafazzo, 2013; Hebden, Cook, van der Ploeg, & Allman-Farinelli, 2012; Kirwan, Duncan, Vandelanotte, & Mummery, 2013; McCurdie et al., 2012; Patel, Nowostawski, Thomson, Wilson, & Medlin, 2013; Pulverman & Yellowlees, 2014; Rabin & Bock, 2011; Savic, Best, Rodda, & Lubman, 2013; Silow-Carroll & Smith, 2013; Wang et al., 2014), Women health (Derbyshire & Dancey, 2013; Robinson & Jones, 2014; Tripp et al., 2014), Dermatology (Chadwick, Loescher, Janda, & Soyer, 2014; Deveau & Chilukuri, 2012; Hamilton & Brady, 2012), Family medicine (Goldbach et al., 2013; Lippman, 2013), Endocrinology (Eng & Lee, 2013), Cardiopulmonary Resuscitation (Kalz et al., 2014), Rehabilitation (Elwood et al., 2011; Milani et al., 2014), Asthma (Huckvale, Car, Morrison, & Car, 2012; McCurdie et al., 2012), Internal medicine (Bierbrier, Lo, & Wu, 2014; Goldbach et al., 2013; O'Neill & Brady, 2012; H.-C. Wu et al., 2014), Cardiology (M. J. Cho, Sim, & Hwang, 2014; Dubey et al., 2014; McCurdie et al., 2012), and Sports medicine (H. Lee et al., 2014). A smaller group of articles provides general overviews of medical apps and their benefits or impacts (M. N. Boulos et al., 2011; M. N. K. Boulos, Brewer, Karimkhani, Buller, & Dellavalle, 2014; Campbell & Choudhury, 2012; Carrera & Dalton, 2014; Fiordelli, Diviani, & Schulz, 2013; Valerie Gay & Leijdekkers, 2011; Liu, Zhu, Holroyd, & Seng, 2011; Mertz, 2012; Moore, Anderson, & Cox, 2012; Y. T. Yang & Silverman, 2014) (10/68). Despite their generality, few of these surveys emphasize special aspects, such as the integration of social networking with medical apps (Valerie Gay & Leijdekkers, 2011), the perspective of developers (Liu et al., 2011), the sensing capabilities of smartphones (Campbell & Choudhury, 2012), or the legal issues and federal regulations of apps (Y. T. Yang & Silverman, 2014). Another few papers (11/68) review apps in the context of specific purposes rather than specific specialties or general views, including apps as references (Haffey et al., 2014; Hilgefort et al., 2013; Zanni, 2013), apps for pain management (B.

Rosser & C. Eccleston, 2011; B. A. Rosser & C. Eccleston, 2011), clinical management (Silow-Carroll & Smith, 2013), pre-operative settings (Brusco, 2010), medical adherence (Dayer et al., 2013), wellness (Handel, 2011), tobacco cessation (Pulverman & Yellowlees, 2014), and even apps for pro-smoking (BinDhim et al., 2014) (to raise awareness of harmful apps).

2.1.2.2 Class 2: Studies conducted on mHealth apps and their use

Despite the frequent complaint in literature about the lack of works that study and assess the phenomena of mHealth apps compared to just reporting on them, around a third of the sample in the above survey (43/133) was articles conducting studies in one form or another (Abroms et al., 2013; Albrecht, von Jan, Jungnickel, & Pramann, 2012; Årsand et al., 2012; Azar et al., 2013; Bierbrier et al., 2014; Breland et al., 2013; Breton et al., 2011; Burdette et al., 2012; M. C. Carter et al., 2013; Chadwick et al., 2014; J. Cho, Park, & Lee, 2014; Choi et al., 2014; Cohn et al., 2011; Dennison et al., 2013; Edlin & Deshpande, 2013; Elwood et al., 2011; Franko, 2011; Franko, Bray, & Newton, 2012; Franko & Tirrell, 2012; Gill et al., 2012; Glassenberg et al., 2013; Goldbach et al., 2013; Haffey et al., 2013; Hamilton & Brady, 2012; Hawkes et al., 2013; Ho et al., 2014; Huckvale et al., 2012; Kalz et al., 2014; Kazi, Saha, & Mastey, 2014; Kuhn et al., 2014; Min et al., 2014; Morris et al., 2013; O'Reilly et al., 2013; Payne, Wharrad, & Watts, 2012; Peck et al., 2014; Rabin & Bock, 2011; Robustillo Cortés et al., 2014; Rozenblyum et al., 2014; Savic et al., 2013; Shand et al., 2013; Spain, 2014; Visvanathan et al., 2012; Wackel et al., 2014; Wearing et al., 2014). The included works in the survey were divided into a large category of evaluation studies (29/43), and a few other smaller categories (14/43). These categories attempt to compare between mHealth apps or between apps and other tools (5/43), explore the desired features sought by users in medical apps (4/43), study the efficacy of medical apps (2/43), check their feasibility in certain situations (2/43), or examine clinician acceptance of using them (1/43). Among evaluation studies, the most popular criteria is the usage patterns of apps by physicians (Elwood et al., 2011; Franko, 2011; Gill et al., 2012; O'Reilly et al., 2013), medical students (Franko & Tirrell, 2012; Payne et al., 2012), or patients (J. Cho et al., 2014; Dennison et al., 2013). Other studies perform content analysis of apps on smoking cessation (Abroms et al., 2013; Choi et al., 2014), asthma self-management (Huckvale et al., 2012), weight management (Azar et al., 2013), addiction recovery (Savic et al., 2013), or references of infectious diseases (Burdette et al., 2012). Arguably, the most sought after studies are those that test the accuracy and reliability of apps. Available studies in this direction are still few, evaluating either the precision of apps measurement compared to traditional tools (Franko et al., 2012; Ho et al., 2014; Wackel et al., 2014), the accuracy their calculations (Bierbrier et al., 2014; Haffey et al., 2013), or the reliability of their assessment (Chadwick et al., 2014). A related class to these studies is the articles that address the adherence of mHealth apps to regulations and established guidelines, especially those related to evidence-based behaviour change (Breton et al., 2011; Cohn et al., 2011; Wearing et al., 2014), and diabetes self-management (Breland et al., 2013). Other evaluation studies examine the involvement of healthcare professionals in the development of mHealth apps (Edlin & Deshpande, 2013; Hamilton & Brady, 2012; Visvanathan et al., 2012), or evaluate apps against a specific set of selected criteria (Albrecht et al., 2012; Robustillo Cortés et al., 2014; Spain, 2014).

Apart from evaluations, few works compare between two mHealth apps (Glassenberg et al., 2013; Morris et al., 2013), between an app and traditional website and paper-based tools (M. C. Carter et al., 2013), or between an app and smartphone-based website access (Goldbach et al., 2013). Another group of studies reported lessons on the design and best practices of developing mHealth apps with features in demand (Årsand et al., 2012; Kazi et al., 2014; Rabin & Bock, 2011; Rozenblyum et al., 2014). A couple of studies examined efficacy of mHealth apps: whether the use of apps can improve performance of trainees in new-born intubation (Hawkes et al., 2013), and effectiveness of apps in suicide prevention (Shand et al., 2013). Another couple of apps addressed the feasibility of using mHealth apps on either daily collection of self-reporting data (Min et al., 2014), or as immunization reminder systems (Peck et al., 2014). Finally, Kuhn et al. investigated the acceptance of mental-health clinician to a future mHealth app based on its description (Kuhn et al., 2014).

2.1.2.3 Class 3: Reports on actual attempts to develop mHealth apps

The literature on mHealth apps includes active attempts to participate in the new trend and develop apps by the researchers themselves (mostly professionals from healthcare disciplines) (17/133). The first such attempt was published in 2010, proposing the use of web apps to collect patients' data (Hamou et al., 2010). A popular choice among articles in this category is to develop physical-activity behaviour change and fitness apps (V. Gay & Leijdekkers, 2012; Hebden et al., 2012; Kirwan et al., 2013). Most papers from IEEE conferences (7/12) appear in this category, reporting on the development of mHealth apps (C. S. Xu et al., 2012; Zhu et al., 2012), proposing the use of hardware capabilities like barcode and RFID tags (Schreier et al., 2013), and the use of data mining (Tseng et al., 2012), or proposing complete designs of apps (Ramachandran & Pai, 2014). A couple of articles demonstrate the use of motion sensors (Aguinaga & Poellabauer, 2013; Dunton et al., 2014). Other options in this category include the development of educational apps (M. J. Cho et al., 2014). The rest of apps-development articles include reports on apps to facilitate public observations collection (Patel et al., 2013), collaboration among researchers (Alexander et al., 2013), or assist international patients by translating medical terms (Hasegawa et al., 2013). One article targets patients of colorectal cancer via early screening service (H.-C. Wu et al., 2014), and the final article in this category reports a large-scale experience with developing 12 health apps in the largest tertiary hospital in Korea (J.-Y. Park et al., 2014).

When talking about development, the choice of platform is pertinent. Most of the first mHealth apps were developed for Apple iOS (through iPhone or iPad devices), as the commencement of this platform predated Google Android (2007 and 2008 respectively). However, most of the research development works in the surveyed sample targeted the Android or both platforms (7/17 and 4/17 respectively). Five articles developed for the iOS, and one article chose to develop a cross-platform, web-based app. As of the target audience of the developed mHealth apps, the majority of apps targeted the patients or the public (12/17), two apps targeted medical staff, and three apps targeted both groups.

Ten of the articles developing mHealth apps explicitly stated the involvement of external professionals of the subject matter in addition to the authors. Those professionals included software developers (Hamou et al., 2010; Kirwan et al., 2013); personal from marketing, nutrition and dietetics, physical activity and information technology (Hebden et al., 2012); two psychologists, a software engineering expert, an Objective-C developer and a media designer (Zhu et al., 2012); software developers, information scientists and end-users (Patel et al., 2013); information technology staff (Alexander et al., 2013); computer scientists, psychologists, epidemiologists, exercise scientists, graphic designers and end users (Dunton et al., 2014); a professional web developer (M. J. Cho et al., 2014); an information technology alliance company (J.-Y. Park et al., 2014); as well as doctors and patients (Ramachandran & Pai, 2014).

2.1.2.4 Class 4: Proposals of frameworks to develop and operate mHealth apps

The final class in the developed taxonomy is articles that cannot be fit in the previous group of articles, since they do not develop new apps, but rather introduce overall frameworks or models for the development or use of them. Articles in this class (5/133) include either works focusing on models and methods for the design (McCurdie et al., 2012) or development of mHealth apps (Paschou, Sakkopoulos, & Tsakalidis, 2013).

Another couple of articles introduce frameworks for data access and integration between apps and other parts of health information systems (Fox, Cooley, McGrath, & Hauswirth, 2012; Mersini, Sakkopoulos, & Tsakalidis, 2013). Finally, a single work addresses the issue of secure exchange of mHealth apps' data, proposing a cooperative environment with data encryption framework (Silva, Rodrigues, Canelo, Lopes, & Zhou, 2013). This category of articles is currently attracting the least attention among researchers. Nevertheless, it is expected that devising frameworks for the production and operation of mHealth apps within the big picture of health informatics would receive more interest as the demand for general and scalable solutions for the current challenges increase.

2.1.3 Articles by Medical Specialty of Apps

It is probably interesting to find out which medical fields are served by the new mHealth apps and to which extent. Figure 2.2 shows the number of articles by the specialty of which their apps cover. The shown articles do not include the full list (133), because the specialties of the addressed apps in 30 articles are not available or not applicable (e.g. the case of proposing general frameworks for apps development). The articles neither add up to 103, since the apps in few articles fall in more than one specialty.

2.1.4 Articles by Purpose and Function of Apps

MHealth apps generally serve particular purposes or functions. Examples include the functions of information reference, education, self-management, clinical practice and diagnosis. Excluding the review articles (since each review usually surveys apps from the whole range of functions), the number of articles from the other categories (detailed by the purpose of apps they address) is shown in Figure 2.3. The value of this figure is to gain an insight into the most visited functions by studies and development efforts, and those that need more attention. The list in Figure 2.3 misses nine articles, where the specific purpose of the subjected apps could not be found.

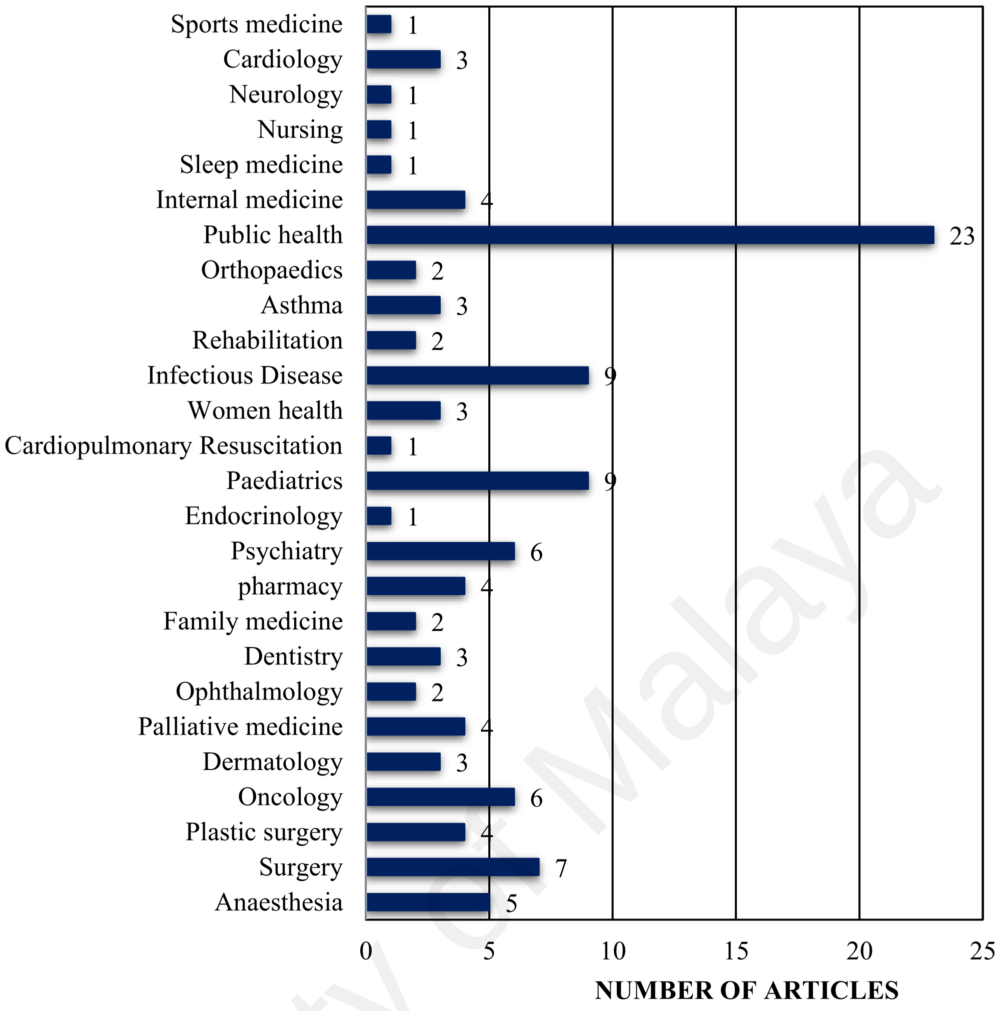


Figure 2.2: Number of Included Articles by the Specialty of Apps They Cover

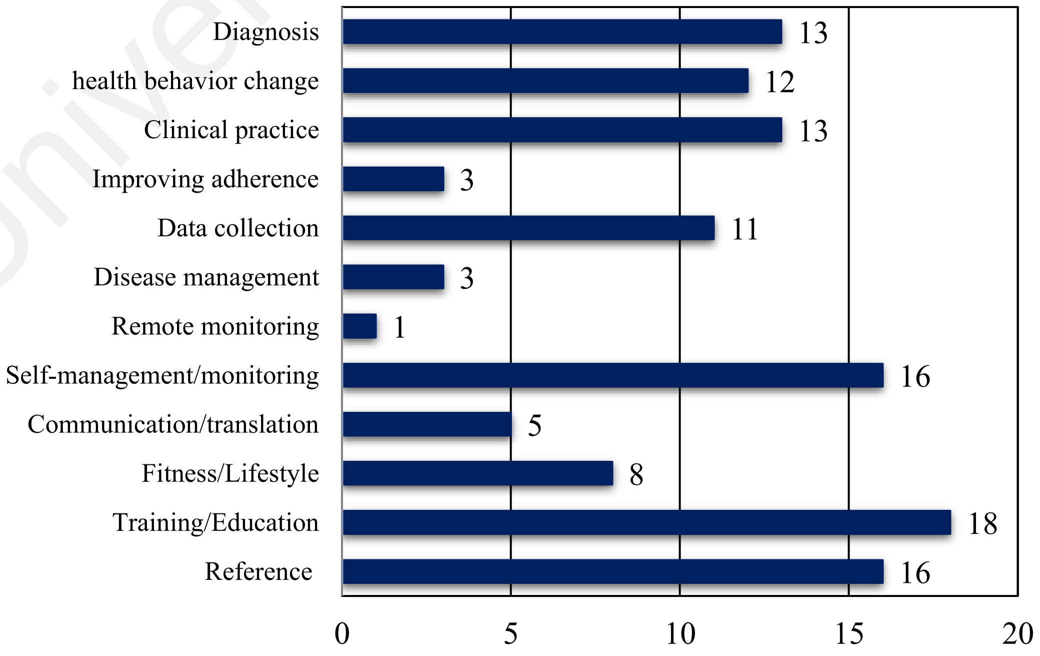


Figure 2.3: Number of Included Articles by the Purpose or Function of Apps They Cover

2.1.5 Articles by Indexing Databases

Figure 2.4 depicts the distribution of articles from different categories over the digital databases in which the search was performed. The purpose is to highlight the potential venues for seeking (as well as publishing) works on mHealth apps. It seems that the disciplines of life and medical sciences are more interested in this subject, which is to be expected. It is also important to note that the figures in this graph are not consistent with the numbers of articles initially found in each database. For example, the initial query against WoS index triggered only 56 results before any exclusion, while Figure 2.4 shows 86 articles of the final set in WoS. The reason for this discrepancy is that the initial query failed to pull out all the relevant articles from this particular database, though the same query yielded more articles from the other database. Because the databases were looked up manually against the final sample set of articles, more articles showed up for each database than it could itself return. This indicates that the individual search engines matter in performing queries in addition to the specific query string.

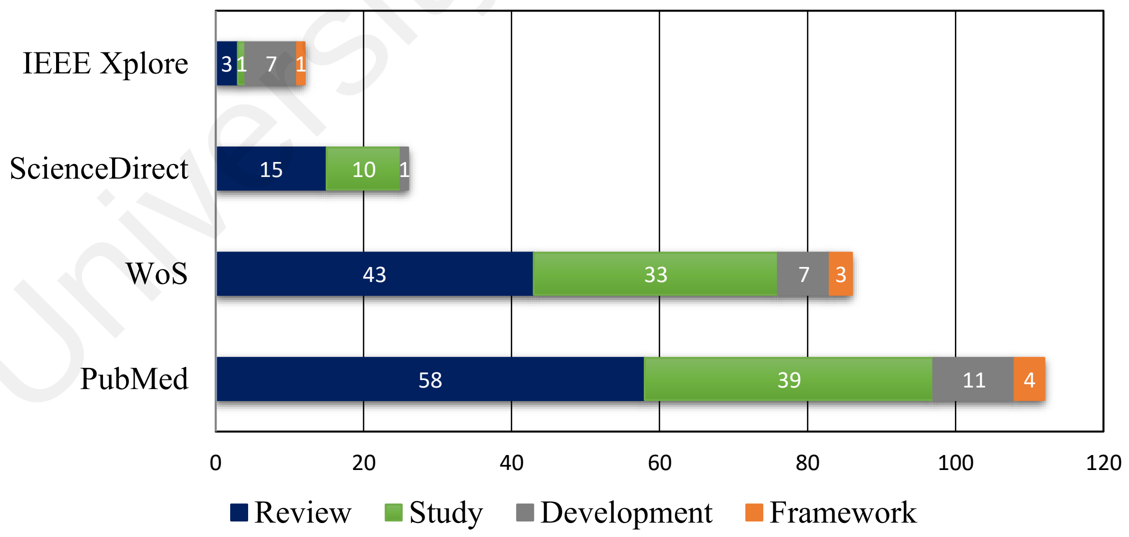


Figure 2.4: Number of Included Articles in Different Categories by the Source Digital Database

2.1.6 Motivations for Smartphone mHealth Apps

The benefits of using smartphone apps in the healthcare domain are obvious and compelling. This section lists but a few of the advantages reported in the literature, grouped into categories of similar benefits and citing the corresponding references for further discussion.

(a) *Benefits related to smartphones portability*

Smartphones are agile, handheld, and can be used on the move (M. N. Boulos et al., 2011). This mobility and portability allow for several benefits. For example, Smartphone apps can provide timely communication (Elias et al., 2014), and are ideal for keeping a symptom diary as they accompany users all the time (Lippman, 2013). From a research perspective, smartphone apps allow conducting ecological momentary assessment (EMA) using MAs, where patients can capture data describing their experience on spot in real time (Tripp et al., 2014), and they also allow for repeated sampling of behaviour over numerous time points and allow the ability to capture less frequent and rare events (Cohn et al., 2011).

(b) *Benefits related to smartphones' capabilities*

As mentioned earlier, smartphones possess several capabilities that enable new possibilities via installed apps. Because they can connect to the Internet, smartphones are useful to keep clinicians up-to-date with the latest medical techniques and advances (M. N. Boulos et al., 2011). The continuous connectedness of smartphones allows the sharing of behavioural and health data with health professionals or peers (Dennison et al., 2013). This ability may also allow telemedicine to replace time-consuming office visits altogether (M. N. K. Boulos et al., 2014). Moreover, Smartphone mHealth apps are specifically created to work well for point-of-care decisions about topics such as drug dosing and conventional treatment regimens, where the needed information are aggregated and presented in an easily digestible format (Goldbach et al., 2013). The increasing ability of smartphones to use internal sensors to infer context such as user location, movement, emotion, and social engagement has raised the prospect of continuous and automated tracking of health-related behaviours and timely, tailored interventions for specific contexts (Dennison et al., 2013). Apart from hardware or software features, the feature of anonymity granted by apps allows patients to ask questions they might otherwise feel embarrassed asking a healthcare professional (Tripp et al., 2014).

(c) *Benefits related to smartphones' market penetration*

The popularity and ubiquity of smartphones allow for access into populations that are difficult to reach and engage (Goyal & Cafazzo, 2013). For patients who cannot access care provision premises, mHealth apps are especially beneficial (Silow-Carroll & Smith, 2013).

2.1.7 Challenges to mHealth Apps

Though attractive, smartphone mHealth apps are (still) not believed to be the panacea of healthcare delivery. The literature indicates that researchers are concerned about many challenges associated with apps and their use in healthcare. Key reported challenges for adoption of mHealth apps are listed below, along with citations to references in which the reader can find the original suggestion and further discussion on those challenges. The challenges are classified into a few groups according to their nature.

(a) *Concerns on quality*

Perhaps the most persistent and crucial challenges are those related to the quality of the developed mHealth apps. Major issues include concerns on the low involvement of qualified professional in app development (Cheng et al., 2014), a lack of extrinsic scrutiny and peer review after publishing (Cheng et al., 2014; Eng & Lee, 2013; Goldbach et al., 2013), lack of evidence of clinical effectiveness, (Eng & Lee, 2013), lack of objective research to evaluate outcomes (Silow-Carroll & Smith, 2013), the absence of content regulation (O'Neill & Brady, 2012) as well as the absence of a regulatory framework that standardizes development (Silow-Carroll & Smith, 2013). Furthermore, many smartphone apps are not based on behavioural change theories or guidelines (M. N. K. Boulos et al., 2014).

(b) ***Concerns on security and privacy***

Privacy of patients and security of their data are also a major and pertinent issue when talking about smartphone apps, and has been frequently raised by researchers (M. N. K. Boulos et al., 2014; Elias et al., 2014; O'Neill & Brady, 2012; Silow-Carroll & Smith, 2013). Elias et al. (2014) note the non-compliance of mHealth apps with the Health Insurance Portability and Accountability Act, unlike the traditional EHRs. They also notice that the apps distribution business model needs caution. Apps and service provided are free to the individual, but privacy is not assured. Information collected about an individual while using the app and its associated services can then be used for targeted marketing either directly by the company or sold to others for marketing or product development (Elias et al., 2014). Furthermore, there are always security risks for less experienced users who might be tricked to download apps that contain malware or offer them dubious medical information and advice (M. N. K. Boulos et al., 2014).

(c) ***Concerns on integration***

It is important to note that mHealth apps are just one (important) aspect of mobile health, which is one form of healthcare delivery; as such, a vital issue is the lack of integration with other parts of the healthcare delivery system (Eng & Lee, 2013). In particular, technical challenges are caused by the lack of seamless interfaces between app platforms and providers' existing information technology systems (Silow-Carroll & Smith, 2013).

(d) *Concerns on usability*

Many researchers also highlighted the problems in using smartphone apps because of the additional involved complexity and the limited usability compared with traditional platforms such as PCs. For example, complexity is introduced to individuals by the need to manage a mix of mobile devices, personal apps, and apps they use for healthcare purposes, each with its own learning curve, possible financial costs, and security and privacy concerns. This burden on consumers could become overwhelming with each organization, provider, and associated businesses requiring use of their own apps (Elias et al., 2014). Moreover, those in rural areas may have limited or no signal and will be unable to benefit from the use of mHealth apps. Beyond access, the patient has to commit to daily use of the app (Elias et al., 2014). Other usability issues relate to the small internal storage capacity, processing power and screen size of the mobile phone, which require apps to be used in a reduced format, potentially reducing clarity (O'Neill & Brady, 2012; Silow-Carroll & Smith, 2013). There are also the prosaic issues such as remembering to recharge a device and the simple maintenance of equipment within a patient's home, which may be problematic (Silow-Carroll & Smith, 2013). Furthermore, older patients in particular may suffer from lack of knowledge or discomfort with technology (Silow-Carroll & Smith, 2013); an app that is perfectly usable by a younger person might be very difficult to manipulate by an older or disabled person (M. N. K. Boulos et al., 2014).

(e) *Concerns on safety*

In a medical environment, apps might cause unexpected effects, such as surface contamination. Smartphones can act as a reservoir for bacteria, and it is possible that doctors using mHealth apps are less likely to perform hand hygiene, thereby increasing the risk of bacterial transmission (O'Neill & Brady, 2012). Electromagnetic radiation from the mobile devices not only could hamper the functionality of patient devices such as pacemakers, but also could interfere with other medical equipment (Gill et al., 2012).

In addition, app descriptions in general contain limited advice or safety information regarding their use as a medical tool (Eng & Lee, 2013).

(f) *Concerns on financial costs*

Despite the pervasiveness of smartphones, certain cost overhead associated with mHealth apps might hinder their wide adoption and use. Examples of the cost overhead include the hidden charges of connection, particularly for apps that automatically connect to other apps or services (Elias et al., 2014). In addition, some patients cannot afford smartphones or the required high-speed Internet connection. For app providers, the development, support, maintenance and regular updating may entail significant costs (M. N. K. Boulos et al., 2014).

(g) *Concerns on administrative and ethical issues*

A less obvious source of difficulties is the reimbursement obstacles caused by communication via smartphones for providers who devote time to these types of activities (Slaper & Conkol, 2014). Providers working in fee-for-service environments will generally expect to be paid for the time they spend on managing healthcare through apps and for associated software or equipment costs. Yet insurers, employers, and other payers are unlikely to reimburse for these costs until there is more robust evidence of their effectiveness (Silow-Carroll & Smith, 2013).

In addition, ethical and medico-legal questions arise when smartphones are used to record patient information via mHealth apps. Informed consent from individual patients would be required (O’Reilly et al., 2013). Moreover, advertising through apps allows companies to target physicians directly, potentially indirectly and unethically influencing prescribing and treatment practices (O'Neill et al., 2013).

(h) ***Concerns on negative effects***

Finally, the use of smartphone apps for medical purposes may entail unwanted effects, many of which have been highlighted in the literature. For example, self-monitoring of certain measures (like glucose) by patients can cause depression and may do more harm than good (Lippman, 2013). Further, apps that provide medical advice based on their own collected data and algorithms could cause unnecessary worry or false reassurance (Eng & Lee, 2013). Other reported challenges include the mistakes and omissions in health care work settings because of distraction and interruptions caused by interaction with apps or their notifications, the impact on inter-professional relationships due to overreliance on communicating by apps, resulting in a decrease in verbal communication, and unprofessional behaviours in the use of smartphones by residents (Gill et al., 2012). Another important issue is the effect on aspects of essential communication between patients and care providers, such as eye contact, gestures, visibility of actions, and verbal and nonverbal contact (Gill et al., 2012; Payne et al., 2012).

2.2 The Android Platform and Its Security

The main goal of this thesis is to protect sensitive information of mHealth apps and their user. MHealth apps operate on mobile platforms, and because the Android operating system is chosen as the target platform in this thesis, there is a need to review its structure as well as its security model in detail. This section briefly presents the details of Android OS, its components, the possible types of communication in the system, and the supported security mechanisms.

2.2.1 Android System Architecture

Android is a Linux based operating system developed primarily for mobile devices (e.g., smartphones or tablets) circa 2003 by Android Inc. (Elgin, 2005). Soon after, Google acquired this company in 2005. Google commenced the first Android device in

October 2008, and thenceforth it maintains the development of the operating system, as well as its marketing and support. Google also releases the source code of Android under open source licenses, and allows vendors to customize new releases of the operating system and to use the customized version in their own devices, subject to the compliance of a special compatibility agreement (Project, 2016).

Android operating system is a stack of software components, as shown in Figure 2.5. Android platform comprises a Linux kernel, a middleware layer and an application layer. Google customized the underlying Linux internals to provide strong isolation between different processes, and then built the whole system upon the modified Linux kernel. The kernel also serves as an abstraction layer between the hardware and other software layers (Rashidi & Fung, 2015). The Linux kernel provides the usual basic facilities, for instance: memory management, device drivers, process scheduling, and a file system.

The Android middleware layer lies on the top of the Linux kernel. This layer contains three main components: the application framework, the native operating system libraries, and the Android runtime environment. The application framework is written in Java and is a collection of services that define the environment in which Android apps are run and managed. These services are offered to apps as Java classes. System applications such as the *system content providers* and *system services* are also part of the application framework. These applications and services provide the essential functionalities and services of the platform, such as *System Settings*, *Clipboard*, *LocationManager*, *WifiManager,* and the *AudioManager*. In Android platform, *system content providers* are essential databases, while *system services* provide the required high-level functions to control the device’s hardware and to get information about the platform state, such as location and network status.

Another part of the middleware is a set of native libraries, which provide functionalities such as graphics processing and multimedia support. These libraries are written in the C/C++ programming language. The final part of Android middleware layer is the runtime environment, which comprises the Dalvik Virtual Machine and core Java libraries. This layer is mostly written in C/C++ except parts of the core libraries, and is customized for the specific needs and requirements of resource-constrained mobile devices.

As illustrated in Figure 2.5, Android Application layer is located at the top of the Android software stack. This contains both the pre-installed apps (i.e., native Android apps) and the third party apps developed by different (unofficial) app developers. Apps are written in Java, but for performance reasons may include native code (C/C++), which is called through the Java Native Interface (JNI). Basically, the Android OS is a multi-user system, in which each app has a unique user ID (UID). All files in an app will be assigned to that apps UID and usually not accessible to other apps. Each app runs in its own Linux process with a unique UNIX user identity and isolated from other apps, so that apps must explicitly share data and resources. In this way, Android platform implements the principle of *least* privilege. Generally, Android app consists of certain components: Activity (User interface), Service (background process), Broadcast Receiver (mailbox for broadcast messages), and Content Provider (SQL-like database) (Bugiel, Davi, Dmitrienko, Heuser, et al., 2011).

The usual path to develop Android apps is to use the special Software Development Kit (SDK) provided by Google, which include an extensive set of tools and API libraries, using the Java programming language. This development option relies on the rich Java application framework in Android, which enable developers to interact with various aspects of the system including hardware and software components, such as sensors, wireless interfaces, telephony services, as well as multimedia and user interface elements. It is also possible to write native apps in the C/C++ language through the special Native Development Kit (NDK) (Android, 2016a). A finalized Android app is not distributed in the traditional Java *bytecode* format within ".class" files or ".jar" archives. Rather, Android development kit includes a tool that converts the Java compiled bytecode into custom bytecode in the form of ".dex" files to be executed by the Dalvik virtual machine. Developers have also to sign their apps, though they can use self-signed certificates (Android, 2016c). The official online store and market for Android apps is Google Play (Google, 2016), where developers can distribute and sell their apps.

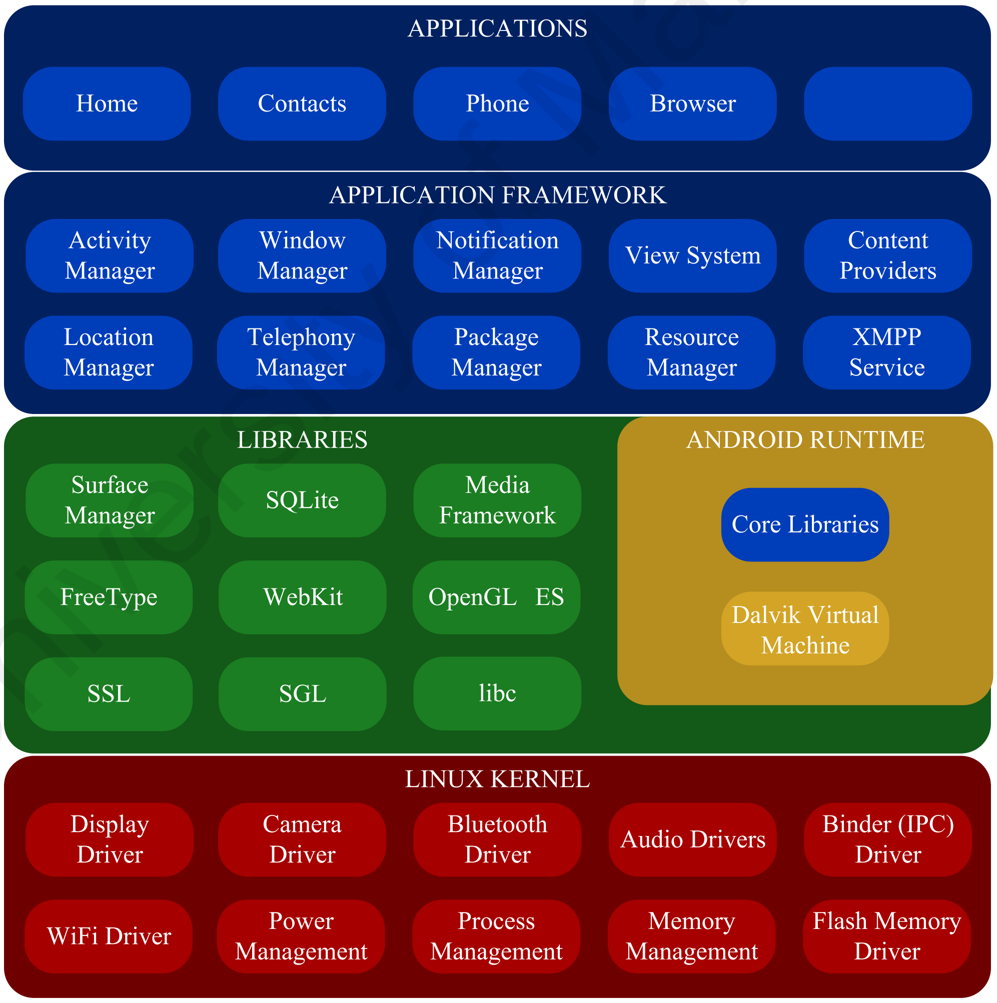


Figure 2.5: Android System Architecture

Android platform is a common target for academic research studies, including the present one. The main reason behind this popularity is straightforward: Android is an open product; its source code is available from Google for interested parties, including other software developers and hardware vendors. This has resulted in an interesting outcome. On the one hand, researchers can dissect the operating system source code, studying relevant parts to their research and potentially modifying the code base accordingly. The possession of a real and mature operating system code at the disposal of researchers to experiment with and implement new ideas proved very attractive and rewarding. Most of the research literature on mobile platforms is directed towards Android, which results in yet more studies that address this platform.

On the other hand, being released under an open license (subject to compliance agreement), Android platform can be adopted by any hardware vendors, including mobile device manufacturers or embedded-system developers. The target devices that employ Android are far more than those employing competing mobile platforms. For example, the market share of Android-based smartphones is almost 87.6% as of 2016 (International Data Corporation, 2016). This means that research studies on the Android platform have much broader potential impact than studies performed on other mobile platforms. Also related to this point is the fact that more hardware devices from various vendors are available to experiment with. This enables researchers to verify their results on a variety of designs and implementations.

The aforementioned reasons justify for the selection of Android platform as the main target in this as well as other studies. Nevertheless, it is worthy emphasizing that most of the results are often applicable to other mobile platforms, with some modifications that depend on the nature of the study.

2.2.2 The Structure of Android Apps

An Android app is an application software that is developed to be run on mobile devices, such as smartphones or tablets (Techopedia, 2016). Each Android app contains one or more of four main components: *activities*, *services*, *broadcast receivers* or *content providers*. With a few exceptions, communication between apps occurs through the middleware layer, which defines the different types of inter-process communication (IPC). There is a direct correspondence between types of IPC and the four main components of apps. In general, IPC is implemented through objects called *intent message* (Enck, Ongtang, & McDaniel, 2009). By definition, an intent is "an abstract description of an operation to be performed" (Developers, 2016b). Intents are addressed either directly to a component using the application's unique namespace, or to an *action string*. For instance, it can be used with the *startActivity()* method to launch an *Activity*, with the *bindService(Intent,ServiceConnection,int)* or *startService(Intent)* methods to communicate with a background *Service*, and with the *broadcastIntent()* method to send it to any interested *BroadcastReceiver* components. Further, developers define *intent filters* based on action strings for different components of apps to automatically start out on corresponding events. For instance, two different Android apps' components and their interaction is shown in Figure 2.6. The details of an app components are given below.

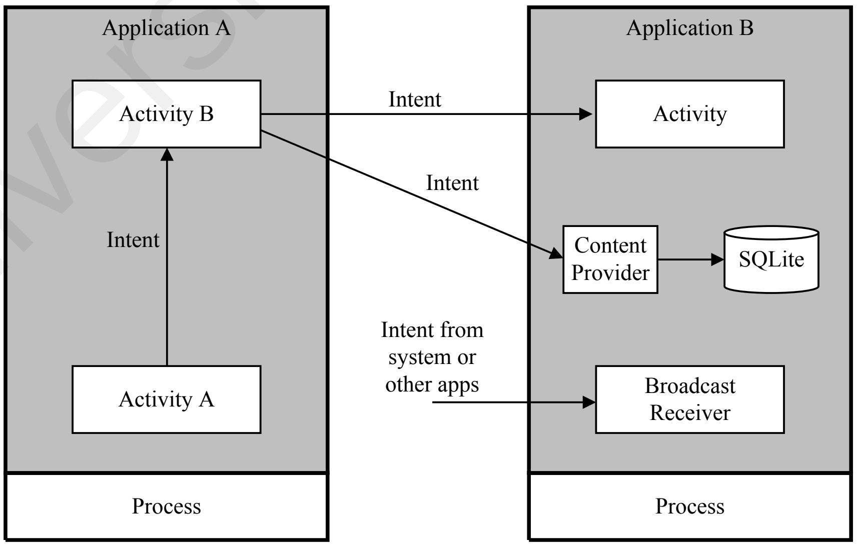


Figure 2.6: Android App Components and Their Interactions

2.2.2.1 Activity

An activity component represents a single screen with a user interface (Developers, 2016a). An activity component interacts with the user through touchscreen and keypad.

Apps usually contain multiple activities, one for each screen presented to the user. for example, in an email app, one activity shows a list of new emails, another activity for reading emails, and another activity to compose an email.

2.2.2.2 Service

A service is “a component that runs in the background to perform long-running operations or to perform work for remote processes” (Developers, 2016a). A service component does not provide a user interface. Furthermore, this component provides background processing that continues even after its app loses focus. Services also define arbitrary interfaces for remote procedure call (RPC), including method execution and callbacks, which can only be called after the service has been *bound*.

2.2.2.3 Content provider

A Content Provider component manages a shared set of app data. This component is a database-like mechanism for sharing data with other apps. An app user can store the data in the file system, an SQLite database, on the web, or any other persistent storage location that apps can access. The interface with content providers does not use intents, but rather is addressed through a *content URI*. It supports standard SQL-like queries, e.g., SELECT, INSERT, UPDATE, through which components in other apps can retrieve, store or even modify the data according to the content provider’s schema (i.e., if the content provider allows it).

2.2.2.4 Broadcast receiver

A broadcast receiver is another Android app component that responds to system-wide broadcast announcements. It is an asynchronous event mailbox for broadcasted intent messages.

Many broadcasts originate from the system, for instance, a broadcast announcing that the battery is low, the screen has turned off, or a picture was captured. Apps can also initiate broadcasts, for instance, to let other apps know that some data has been downloaded to the device and is available for them to use (Developers, 2016a).

Every app package includes a manifest file. The manifest file defines all components in an app that also includes their types and intent filters. Note that Android platform allows apps to dynamically create broadcast receivers that do not appear in the manifest (Enck et al., 2009).

2.2.3 Android Security Model

This section briefly lists the security mechanisms that Android platform uses to secure the application environment. Android implements a number of security mechanisms, the most prominent of which are application sandboxing and a permission framework that enforces mandatory access control (MAC) on inter-component communication calls and on the access to core functionalities. A detailed overview of Android security mechanisms can be found in (Android, 2016b).

2.2.3.1 Android permission system

By default, Android apps have limited access to system's resources and to each other's components. The access to sensitive resources within Android is protected through a security mechanism called *permissions*. It provides protected APIs for the sensitive resources, including telephony, GPS, location, camera, Bluetooth, SMS/MMS and network access. To make use of the protected APIs, an Android app must declare the permissions associated to those APIs' in its own manifest file, and the permissions are agreed upon at installation time by the user.

2.2.3.2 Application sandboxing

All installed apps on Android platform run in an application sandbox. The Android assigns a unique user identifier (UID) to each app and runs it as an individual user in a separate process. Furthermore, each app runs in its own instance of the DVM under the assigned UID. This sandboxing mechanism also applies to native code contained in apps. However, apps from the same vendor can use a shared UID, hence basically sharing the sandbox. Basically, the kernel enforces security between applications via standard Linux features, such as UID-based permissions and process isolation.

2.2.3.3 Application signing

Application signing allows to identify the author of an app. Application signing is the first step to ensure the application sandbox mechanism; certificates are signed to ensure which UID is associated to which app and different apps run under different UIDs. As an app is installed on Android OS, the system verifies that the app has been properly certified.

2.2.3.4 Secure inter-process communication

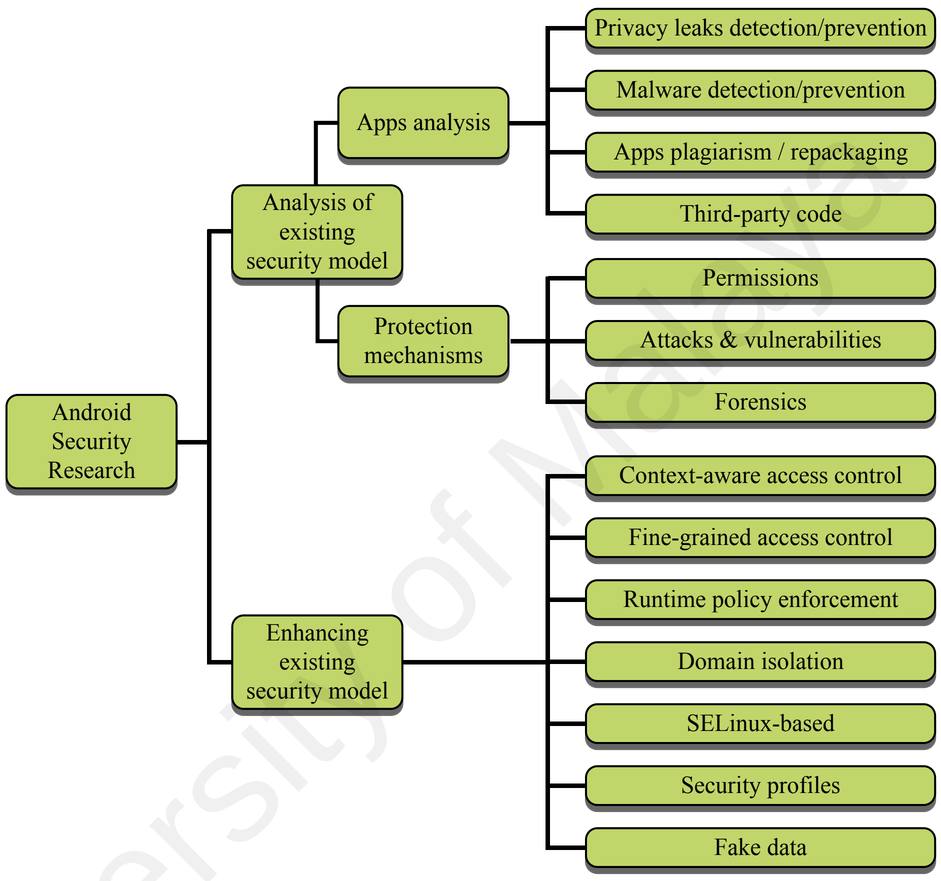
In Android platform, processes can communicate by using any of the standard UNIX-style mechanisms. Android platform also provides new Inter-Process Communication (IPC) mechanisms that includes *binder, intents, services and content providers*. *Binder* is a lightweight remote procedure call mechanism that is designed for high-performance in-process and cross-process calls. *Services* can provide interfaces directly accessible using *Binder*.

2.2.3.5 SELinux

Android employs Security-Enhanced Linux (SELinux) (Shabtai, Fledel, & Elovici, 2010) to apply mandatory access control. SELinux is the primary Mandatory Access Control (MAC) mechanism built into a number of GNU/Linux distributions.

2.2.4 Android Security Research Trends

Recently, a number of security extensions for Android have been proposed. Figure 2.7 is classifying the diverse and rich literature on Android security into a comprehensive taxonomy of research directions.



Android Security Research
Analysis of existing security model
Apps analysis
Privacy leaks detection/prevention
Malware detection/prevention
Apps plagiarism / repackaging
Third-party code
Protection mechanisms
Permissions
Attacks & vulnerabilities
Forensics
Enhancing existing security model
Context-aware access control
Fine-grained access control
Runtime policy enforcement
Domain isolation
SELinux-based
Security profiles
Fake data

Figure 2.7: Taxonomy of Literature on Android Security

Two major directions can be recognized in existing studies on the current security model of Android (Enck, 2011). Some researchers focus on the protection mechanisms while others analyse the apps themselves. When it comes to the Android security model, one of the most obvious and common targets for research studies is the permissions system. Researchers have covered several aspects related to permissions. Few works have explained the permission system (Felt, Chin, Hanna, Song, & Wagner, 2011), while others have analysed them (Au, Zhou, Huang, & Lie, 2012; Barrera, Kayacik, van Oorschot, & Somayaji, 2010; Sarma et al., 2012; Shin, Kiyomoto, Fukushima, & Tanaka, 2010; R. Stevens, Ganz, Filkov, Devanbu, & Chen, 2013). Other studies have questioned their effectiveness (Felt, Greenwood, & Wagner, 2011; Orthacker et al., 2011), and also their usability (Felt, Ha, et al., 2012; Kelley et al., 2012). Yet, further research proposed improvements to the permission system itself (Do, Martini, & Choo, 2014; Fragkaki, Bauer, Jia, & Swasey, 2012; Jeon et al., 2012; Shen et al., 2014). In addition, research on permissions includes attempts to guide developers on how to request the right permissions (Vidas, Christin, & Cranor, 2011) as well as guidelines for platform designers (A. M. A. Al-Haiqi, 2015; Felt, Egelman, Finifter, Akhawe, & Wagner, 2012).

One of the most infamous attacks on the Android platform is the *privilege escalation attack*. In this attack, an app that lacks enough permissions can delegate the performance of a task that needs missing privileges to another app with the necessary permissions. This attack has been analysed in many studies (Davi et al., 2011; Felt, Wang, Moshchuk, Hanna, & Chin, 2011; Marforio, Ritzdorf, Francillon, & Capkun, 2012). Another type of studied attacks include the exploitation of various vulnerabilities, such as those at the level of system design (Lee, Lu, Wang, Kim, & Lee, 2014) and the level of managing package updates (Xing, Pan, Wang, Yuan, & Wang, 2014).

Examples of other miscellaneous attacks include the exploitation of external device mis-bonding (Naveed et al., 2014), the *usr-interface* state inference attack (Chen, Qian, & Mao, 2014), the denial-of-app attack (Arzt, Huber, Rasthofer, & Bodden, 2014), and the attack on the *WebView* component (Luo, Hao, Du, Wang, & Yin, 2011). Furthermore, Android forensics received a lot of attention as well (Hoog, 2011; Lessard & Kessler, 2010; Spreitzenbarth, 2011), examples include forensic methods of collection and acquisition (Simão, Sícoli, Melo, Deus, & Sousa Júnior, 2011; Vidas, Zhang, & Christin, 2011), methods for analysing the file system (Quick & Alzaabi, 2011; Schmitt, 2011), and techniques to counteract the forensic methods (Albano, Castiglione, Cattaneo, & De Santis, 2011; Distefano, Me, & Pace, 2010; Karlsson & Glisson, 2014).

Malicious apps (malware) are the main vehicle to implement threats on the Android platform. Therefore, many research studies can be classified under the category of malware analysis (Felt, Finifter, Chin, Hanna, & Wagner, 2011; Yan & Yin, 2012; Y. Zhou & Jiang, 2012). Other studies focused on malware detection (Aafer, Du, & Yin, 2013; Arp, Spreitzenbarth, Hubner, Gascon, & Rieck, 2014; Bläsing, Batyuk, Schmidt, Camtepe, & Albayrak, 2010; Burguera, Zurutuza, & Nadjm-Tehrani, 2011; Enck et al., 2009; Gorla, Tavecchia, Gross, & Zeller, 2014; M. Grace, Zhou, Zhang, Zou, & Jiang, 2012; Sanz et al., 2013; Shabtai, Kanonov, Elovici, Glezer, & Weiss, 2012; Weichselbaum et al., 2014; D.-J. Wu, Mao, Wei, Lee, & Wu, 2012; W. Zhou, Zhou, Grace, Jiang, & Zou, 2013; Y. Zhou, Wang, Zhou, & Jiang, 2012). Few researchers even evaluated malware detectors (Maggi, Valdi, & Zanero, 2013; Rastogi, Chen, & Jiang, 2013; Zheng, Lee, & Lui, 2012).

The analysis of Android apps revealed new types of threats, such as apps repackaging (Crussell, Gibler, & Chen, 2012, 2013; Gibler et al., 2013; Hanna et al., 2012; Linares-Vásquez, Holtzhauer, Bernal-Cárdenas, & Poshyvanyk, 2014; W. Zhou, Zhang, & Jiang, 2013; W. Zhou, Zhou, Jiang, & Ning, 2012). Another example of new threats is the problem of embedded third-party code (Sun & Tan, 2014), especially threats from advertisement libraries (M. C. Grace, Zhou, Jiang, & Sadeghi, 2012; Pearce, Felt, Nunez, & Wagner, 2012; Shekhar, Dietz, & Wallach, 2012; Zhang, Ahlawat, & Du, 2013).

Malware is not the only threat that apps can impose on users. Apps might leak users' sensitive data unintentionally. This type of apps is sometimes referred to as grayware. Techniques to protect privacy were addressed by many research works. One of the most common techniques is the concept of information-flow tracking. In this technique, private data are tainted and then traced throughout its flow in the app. The flow is logged and possibly blocked whenever a labelled object moves from a private domain to a public domain (e.g., transferred out through the network). A famous implementation of this type of techniques is TaintDroid (Enck et al., 2014). Other examples of dynamic taint tracking can be found in (Gilbert, Chun, Cox, & Jung, 2011; Hornyack, Han, Jung, Schechter, & Wetherall, 2011; Mollus, Westhoff, & Markmann, 2014; C. Zheng et al., 2012).

Another technique to detect leakage of private data is the static analysis of the source code of Android apps after decompilation (Arzt, Rasthofer, et al., 2014; Chan, Hui, & Yiu, 2012; Chin, Felt, Greenwood, & Wagner, 2011; Fuchs, Chaudhuri, & Foster, 2009; M. C. Grace, Zhou, Wang, & Jiang, 2012; Z. Yang & Yang, 2012). Additional techniques include symbolic execution (Z. Yang et al., 2013), instrumentation of byteode (Bartel, Klein, Monperrus, Allix, & Le Traon, 2012; Karami, Elsabagh, Najafiborazjani, & Stavrou, 2013) and repackaging apps (Berthome, Fecherolle, Guilloteau, & Lalande, 2012).

Aside from examining protection mechanisms and analyzing available apps, significant portion of the research on Android security proposes various enhancements to its security model. Several studies proposed enhancements to extend the current platform in the form of new frameworks. Those enhancements aimed to improve several aspects of the system's security and the user's privacy. For example, Saint (Ongtang, McLaughlin, Enck, & McDaniel, 2012) proposes a framework to control grants of install-time permissions and the use of those granted permission during runtime, all according to a policy dictated by the app developer. Other frameworks that enhance runtime policy include Apex (Nauman, Khan, & Zhang, 2010), Aurasium (R. Xu, Saïdi, & Anderson, 2012) and AppGuard (Backes, Gerling, Hammer, Maffei, & von Styp-Rekowsky, 2014).

Another way to improve Android security is to provide context-aware privacy. Among the works that proposed the latter are (Bai, Gu, Feng, Guo, & Chen, 2010; Chakraborty et al., 2014; Conti, Nguyen, & Crispo, 2011). Other works extended the security model with fine-grained access control (Bugiel, Heuser, & Sadeghi, 2013; Russello, Crispo, Fernandes, & Zhauniarovich, 2011; Y. Zhou et al., 2011). One solution enables the users to reply to apps' requests with empty or unavailable resources, based on certain conditions (Beresford, Rice, Skehin, & Sohan, 2011). Yet another way to improve the security model is by implementing isolation between several security domains (Bugiel, Davi, Dmitrienko, Heuser, et al., 2011), by providing different security profiles (Zhauniarovich, Russello, Conti, Crispo, & Fernandes, 2014), or by enabling differentiated user access control (Ni, Yang, Bai, Champion, & Xuan, 2009). SELinux-based mandatory access control was also added to the Android architecture (Bugiel, Heuser, & Sadeghi, 2012; Shabtai et al., 2010).

It might be worthy to note, however, that comprehensive surveys on that rich Android security literature are very limited and many times outdated (Becher et al., 2011; Enck, 2011; La Polla, Martinelli, & Sgandurra, 2013). Table 2.1 summarizes existing security solutions on Android.

Table 2.1: A Summary of Existing Android Security Proposals in the Literature

System	Reference	Technique(s) Used	Brief Description
Kirin	(Enck et al., 2009)	Rule-based system	The Kirin security service is proposed to perform lightweight certification of apps to mitigate malware at install time. Kirin security service uses a set of security rules on apps' requested permissions to detect matched malicious permission requests and characteristics. Kirin extracts the permissions from an app's manifest file at install time, and checks whether these permissions are breaking certain security rules. Kirin uses a variant of security requirements engineering techniques to perform in-depth security analysis of Android platform to develop a set of security rules that match malware characteristics.
APEX	(Nauman et al., 2010)	Rule-based policy	Apex is a comprehensive policy enforcement framework for Android platform, which allows a user to selectively grant permissions to different apps, and impose constraints on the usage of resources. Users can define their constraints through a simple interface of the extended Android installer called Poly.

Table 2.1, Continued

System	Reference	Technique(s) Used	Brief Description
Paranoid Android	(Portokalidis, Homburg, Anagnostakis, & Bos, 2010)	Misuse detection	This system is cloud-based detection framework that performs security checks on a remote server. This server hosts exact replicas of user smartphones in separate virtual machines. The novelty of Paranoid Android is to move the process of security checks from the mobile device to a cloud server. The reason behind this is the lack of adequate computational power and battery energy on mobile platforms.
Porscha	(Ongtang, Butler, & McDaniel, 2010)	Rule-based policy, secure delivery	Porscha focuses on the protection of contents based on Digital Rights Management (DRM), such as MP3-based MMS or email. This system places reference monitors and content proxies within the middleware to enforce DRM policies embedded in the received content. The primary goal of the system is to improve the enforcement of DRM policy and ensure that protected content is only accessed by authorized parties and is only accessible by apps that are endorsed by the provider. In addition, Porscha ensures the ability to access contents under policy-defined contextual constraints (e.g., time limitation, a maximum number of viewings, etc.).
AppFence	(Hornyack et al., 2011)	Dynamic taint analysis, resource-access mocking	AppFence provides a mechanism to impose privacy controls on existing (unmodified) Android apps. AppFence provides a data shadowing mechanism to prevent apps from accessing sensitive information that is not required to perform user-desired functionalities, and it also provides an exfiltration blocking mechanism to block outgoing communications tainted by sensitive data. Both of these privacy controls are used to limit an app's misuse of user sensitive data.
QUIRE	(Dietz, Shekhar, Pisetsky, Shu, & Wallach, 2011)	Call-chain propagation	QUIRE deals with attacks based on inter-component communication. This system employs a call-chain tracking technique that provides important contexts in the form of provenance and OS managed data security to local and remote apps communicating by IPC and RPC respectively.
CRePE	(Conti et al., 2011)	Rule-based policy	CRePE is a system to enforce fine-grained policies in the Android platform based on the smartphone's context (i.e., the status of some variables such as location, time, noise, light, and temperature). Users or trusted third parties are allowed to define fine-grained context-related policies in the CRePE system.
TISSA	(Y. Zhou et al., 2011)	Resource-access mocking	TISSA implements a privacy mode on Android, which allows a user to flexibly control, what types of private information will be accessible to an app. Further, users can dynamically (re)adjust the granted access at runtime in a fine-grained manner to achieve their specific desired functionalities.

Table 2.1, Continued

System	Reference	Technique(s) Used	Brief Description
TrustDroid	(Bugiel, Davi, Dmitrienko, Heuser, et al., 2011)	Domain isolation	This is a security framework that provides a lightweight domain isolation on each layer (i.e., middleware layer, kernel layer, and network layer) of the Android software stack, in order to mitigate unauthorized data access and communication among apps. TrustDroid isolates data as well as apps of different trust levels in a lightweight fashion. Basically, it provides app and data isolation by controlling the main communication channels in Android: IPC, databases, files and socket connections.
MockDroid	(Beresford et al., 2011)	Resource-access mocking	MockDroid grants fake permissions to protect private data and allows users to provide fake or 'mock' data to an app interactively, while the app is being used. Users are allowed to revoke access to particular resources at execution time.
Crowdroid	(Burguera et al., 2011)	Anomaly detection	This is a behaviour based malware detection system. It detects anomalously behaving apps through a crowdsourcing framework. Crowdroid analyses the behaviour of Android apps to differentiate between the apps that have identical names and versions, but behave differently.
ComDroid	(Chin et al., 2011)	Static analysis	ComDroid is a mechanism to discover vulnerabilities related to communication between apps. Because many of such vulnerabilities caused by the ability of *intents* to implement both intra and inter-app communication, this system examines interactions between apps and detects risks that might arise in app components. Types of possible vulnerabilities include phishing, data loss, and other unexpected behaviors.
XManDroid	(Bugiel, Davi, Dmitrienko, Fischer, & Sadeghi, 2011)	ICC & channel control	XManDroid is a security framework that mitigates privilege escalation attacks. It extends the monitoring mechanism of Android OS to detect and prevent app-level privilege escalation attacks at run-time based on system-centric policy. XManDroid monitors all interactions between apps and dynamically analyses the apps' transitive permission usage. The communication links that are being monitored should pass verification process against a set of policy rules. In the end, depending on pre-defined policies, the system allows for an effective detection of (covert) channels established through the Android system services and content providers, while simultaneously optimizing the rate of false positives.
RiskRanker	(M. Grace et al., 2012)	Static analysis	RiskRanker is an automated system to analyse whether a particular app exhibits dangerous behaviour. It is a proactive scheme to spot zero-day Android malware. RiskRanker tries to measure potential security risks caused by untrusted apps.

Table 2.1, Continued

System	Reference	Technique(s) Used	Brief Description
Saint	(Ongtang et al., 2012)	Rule-based policy	Saint is another work that allows app developers to provide security policies to be enforced in order to regulate installation-time permission assignment and their run-time use. Saint introduces a fine-grained, context-aware access control model to enable developers to install policies that protect the interfaces of their apps. Though Saint could, with a corresponding system centric policy, provide the isolation of apps on direct and broadcast ICC, it cannot prevent indirect communication via system components.
FlaskDroid	(Bugiel, Heuser, et al., 2012)	MAC	FlaskDroid is a general architecture for ensuring Mandatory Access Control (MAC) on both Android's middleware and its kernel layer simultaneously. This architecture provides an effective and flexible security mechanism to setup various security solutions and fine-grained policies. The authors of the architecture designed a policy language that was inspired by SELinux (Loscocco & Smalley, 2001) in order to extract customized operational semantics at the Android middleware.
DroidScope	(Yan & Yin, 2012)	Dynamic analysis	DroidScope is an Android analysis platform that inspects apps and builds semantic views at both the operating system and Java levels. This platform is a dynamic *virtual machine introspection* tool that is based on QEMU emulator (Bellard, 2005) with custom analysis plugins in the form of defined APIs. To collect apps' activities and trace executions, this system exports APIs of three different types: APIs at the application framework layer, APIs at the hardware layer, and APIs at the Dalvik Virtual Machine layer.
DroidRanger	(Y. Zhou et al., 2012)	Static & dynamic analyses	DroidRanger is an analysis system to detect existing and known malware as well as unknown malware, usually named as zero-day threats. DroidRanger detect known malware using behavioral foot-printing scheme based on permissions. For detecting zero-day malware, DroidRanger applies a filtering scheme based on heuristics in order to find specific inherent behaviors of unknown malware families.
DroidMOSS	(W. Zhou et al., 2012)	Fuzzy hashing	DroidMOSS is an app similarity measurement system. It applies a fuzzy hashing technique (French & Casey, 2012; Server, 2007) to localize and detect the changes from app-repackaging behaviour. Basically, the main function of DroidMOSS is to detect repackaged apps on third-party Android marketplaces. They measure the similarity of apps that are collected from third-party Android marketplace with the apps from the official Android markets.

Table 2.1, Continued

System	Reference	Technique(s) Used	Brief Description
AppGuard	(Backes, Gerling, Hammer, Maffei, & von Styp-Rekowsky, 2013)	Binary rewriting	AppGuard enforces security-related policies on untrusted apps, which are customized by the users. This system employs *inline reference monitoring* to allow users of enforcing security policies defined by them on third party apps. AppGuard controls both third-party apps and the operating system itself.
AppInk	(W. Zhou, X. Zhang, et al., 2013)	Dynamic watermarking	AppInk is mechanism for dynamic watermarking based on graphs. The purpose of this mechanism is to mitigate app repackaging. This tool generates a new app from the source code of one app with a watermark value. The generated app has a transparent embedded watermark and an associated manifest file.
TaintDroid	(Enck et al., 2014)	Dynamic taint analysis	TaintDroid is a system that tracks data flow and allows users to analyze flows of sensitive data. This system enables expert users to detect misbehaving apps, by automatically tainting sensitive data in the smartphone in order to trace them and find out whether the labeled data leave the device. In the latter case, the label of the leaked data is recorded along with the app which sent the data and the destination address.
NativeGuard	(Sun & Tan, 2014)	Native code isolation	NativeGuard is a framework that uses the system-provided isolation between Android processes to sandbox native libraries of an app from other components of the app.

2.2.5 Issues in Android Security

A key component of the Android security model is the permission system that controls the access to sensitive device resources by third-party apps. However, Android's permission control mechanism has been proven ineffective to protect user's privacy and resource from malicious apps (Rashidi & Fung, 2015). Further, it has been shown that the majority of smartphone apps attempt to collect data that are not required for the main function of the app (Gunasekera, 2012; Rashidi & Fung, 2015). Reasons for the drawbacks of the existing permission system include users inexperience with realizing irrelevant resource requests and their urge to use the app even at the expense of compromising their privacy (Felt, Chin, et al., 2011; Felt, Ha, et al., 2012; Rashidi & Fung, 2015).

Further, vulnerabilities in Android kernel can be exploited to obtain access to resources or services that are by default protected from an app or a user. This type of attack is *called privilege escalation* attack and it enables unauthorized apps to perform actions with more privileges than they have been granted. This, in turn, leads to unauthorized access to user data and many sensitive information leakages. It is also possible to exploit Android exported (i.e. public) components an obtain access to critical permissions, and hence, to sensitive resources and information (Davi et al., 2011; Rashidi & Fung, 2015).

There is also the threat of colluding apps, where several apps are developed by the same developer, and therefore released under the same certificate. Users install theses apps, some of which are granted sensitive permissions and others are granted normal ones. Afterwards, each of these apps gets access to the combined pool of their permissions and resources because they are all assigned the same UID (Marforio, Francillon, & Capkun, 2011; Rashidi & Fung, 2015).

Android platform lacks a configurable, runtime ICC control. This was a design decision to fulfil several purposes. The first purpose is to prevent an app from accessing any open interfaces of another app, even if the former had obtained the required permissions at its install time (Chin et al., 2011; Felt, Wang, et al., 2011; Tan, Chua, & Thing, 2015). The second objective is to prevent an app from intercepting an intent broadcast, and possibly stopping its propagation afterward (Chin et al., 2011; Tan et al., 2015). By intercepting system-event broadcasts, a malicious app is able to intercept important system events stealthily, which contain sensitive information, such as an incoming call or SMS. A third purpose is to isolate apps and prevent them from communicating via ICC and other shared channels (Bugiel, Davi, Dmitrienko, Fischer, et al., 2011; Bugiel, Davi, et al., 2012; Tan et al., 2015). However, this lack of runtime inter-app access control can lead to data leakage and confused deputy problems. The presently uncontrolled ICC among apps in Android can be exploited by colluding apps.

Moreover, an Android device has several identifiers that can be used as a unique device ID, such as IMEI, Android system ID, or hardware serial number (Tan et al., 2015). As Android devices are prone to information leakage, if this device ID is also leaked, external parties can track the user easily.

Even outside the Android middleware, there exist potential security weaknesses that could compromise the security of an Android device. In particular, potential security weaknesses or vulnerabilities can be located at the Linux kernel and its native libraries (Loscocco & Smalley, 2001; X. Zhou et al., 2013). Also, weaknesses and vulnerabilities can be associated with device manufacturers' customization and preinstalled apps (M. C. Grace, Y. Zhou, et al., 2012; Tan et al., 2015; L. Wu, Grace, Zhou, Wu, & Jiang, 2013; X. Zhou, Lee, Zhang, Naveed, & Wang, 2014).

2.3 Security and Privacy of mHealth Apps

This section provides the most relevant and related work in the literature to the problem of mHealth-apps' security and privacy, starting with a review of the main security and privacy threats to mHealth apps, then presenting the results of an experimental assessment of sample mHealth apps, and then ending with a critical analysis of existing solutions.

2.3.1 Threats to mHealth Apps

Although mHealth apps provide a lot of benefits and easy access to healthcare services, they are loaded with new security and privacy risks to mHealth app user (Avancha et al., 2012; Kotz, Avancha, & Baxi, 2009; Poon, Zhang, & Bao, 2006). Smartphone apps for healthcare are rapidly increasing. There are several types of mHealth apps, some are using external devices such as medical sensors, and some apps are using smartphone resources, such as the camera for the treatment of the patient. Recently, several studies showed that lack of standardization, guidelines, security and privacy of user data are the main barrier to the widespread use of mHealth apps, and these issues should be addressed in order to improve mHealth apps reliability and usability (Adhikari et al., 2014; Faudree & Ford, 2013; Kharrazi et al., 2012).

The market of mHealth apps is nevertheless growing rapidly; hence, health data are also increasing. Privacy and security of sensitive medical data could be significantly affected by this new trend of treatment of the patient. mHealth apps handle sensitive medical data for patients and healthcare professionals, and those data are as sensitive as those handled by HIPAA entities, but mHealth apps handle the data using lower assurance than HIPPA entities. There is a need to develop frameworks and guidelines for mHealth apps to ensure the security and privacy of health data (He et al., 2014).

Android mHealth apps use third party servers and unsecured Internet communication which have raised the security and privacy concerns. He et al (2014) revealed that several mHealth apps send information over the Internet without encryption and put sensitive information into logs. Numerous apps have component exposure threats, and some apps store unencrypted information on an external storage, e.g., SD Card, where a malicious app can read them. Several mHealth apps use Bluetooth in order to collect data from health or medical sensors. In fact, Bluetooth play a major role for communication in sensor-based health monitoring systems; mHealth apps collect numerous types of health information from Bluetooth, such as electrocardiogram (ECG), heart rate, pulse oximetry, respiration, blood pressure, body temperature, body weight, exercise activities and quality of sleep (He et al., 2014). mHealth apps store various type of information without encryption, including but not limited to, mobile users' name, date of birth, country, preferred language(s), culture preference(s), insurance carrier identifier, personal app identifier, medications, medical conditions, physician(s), and pharmacies (Mitchell et al., 2013).

mHealth apps are different from other health information systems from various perspectives. First, mHealth apps collect large amount of data from patient because mobile devices are always with the patient and can collect data for a long time. Second, mHealth apps collect much broader range of data, which include not only physiological data, but also include the patient activities, location, lifestyle, social interactions, diet details, eating habits and so on. Third, the nature of communication between the patient and healthcare professionals is different (He et al., 2014). These aspects imposed new security and privacy threats to mHealth information systems.

There are several potential attack surfaces in Android system that a malicious party can use to gain unauthorized access to sensitive medical data in mHealth apps. A recent study (He et al., 2014) stated seven attack surfaces that need protection: Third Party Services, Internet, Logging, Bluetooth, SD Card Storage, Side Channels and Exported Components. mHealth apps in Google Play usually send sensitive medical data in plain text and store them on third party servers whose confidentiality rules are not sufficient for this type of data (He et al., 2014). Table 2.2 lists the seven attack surfaces that a malicious party can use to access sensitive medical data.

Developers can view and collect debugging output of apps from Android logging system; therefore, a malicious app with READ_LOGS permission may access sensitive information from log messages. A malicious app with WRITE_EXTERNAL_STORAGE or READ_EXTERNAL_STORAGE permissions can write or read files from external storages, such as SD card (He et al., 2014). Android app developers can declare a component as public or exported; if a component is declared as exported, then a malicious app can send unwanted intents to the component. Furthermore, if a content provider is declared as exported or public, this enables malcious apps to write or read the exported content provider without any permission (He et al., 2014). A malicious party can also use side channels to get sensitive information from mHealth apps in the Android operating system (He et al., 2014).

Table 2.2: Description of Attack Surfaces (He et al., 2014)

Attack Surface	Description
Internet	Sensitive information is sent over the internet with insecure protocols, e.g. HTTP, misconfigured HTTPS, etc.
Third Party	Sensitive information is stored in third party servers
Bluetooth	Sensitive information collected by Bluetooth-enabled health devices can be sniffed or injected
Logging	Sensitive information is put into system logs where it is not secured
SD Card Storage	Sensitive information is stored as unencrypted files on SD card, publicly accessible by any other app
Exported Components	Android app components, intended to be private, are set as exported, making them accessible by other apps
Side Channel	Sensitive information can be inferred by a malicious app with side channels, e.g. network package size, sequence, timing, etc.

He et al. (2014) identified another problem. Developers usually put sensitive information into HTTPS URLs for secure transmission of data. However, even though this information is not visible during transmission, it is still visible in other places, such as server logs, mobile app logs, browser history and so on. It may be difficult to identify or control who is accessing the logs. Mitchell et al (2013) used forensic analysis and testing to prove that current mHealth apps lack adequate privacy and security controls. The authors revealed that several apps store unencrypted personal information on the smartphone itself. The log files show plain text instead of encrypted text from top-downloaded app.

These issues are more important for mHealth apps that store personal health information (PHI) or electronic health records (EHR) and exchange that data with health-related websites (Mitchell et al., 2013). Data security, access control and confidentiality are the main security issues in mHealth apps that must be addressed in order to use these apps in healthcare system (Adhikari et al., 2014; He et al., 2014; Mitchell et al., 2013).

Security standards have not yet been fully implemented by OS developers, app programmers, device manufacturers or the different levels of government agencies; therefore, it is very easy for malicious parties to get access to user data collected by mHealth apps. OS manufacturers commonly hide information relating to data security and data collection policies inside the tens of pages of legalese presented to end users and they usually click "Agree" without reading it (Mitchell et al., 2013).

McCarthy (2013) reported that most of the users' data are poorly protected in mHealth apps. The author also reported that in a technical analysis study of 43 health and fitness apps, only 60% of the paid apps and 74% of the free apps had a privacy policy, and it is accessible either on the developer's website or in the app. However, only 48% of the paid apps and 25% of the free apps informed users about the privacy policy. In addition, none of the free apps and just one of the paid apps encrypted all the communications sent from the device to the developer. The data without encryption in mHealth apps pose a serious threat to users' data privacy.

Adhikari et al. (2014) performed analysis on 20 most popular mHealth apps and highlighted the various security concerns. In brief, the serious risks to users' data are insufficient security measures, lack of users' authentication, sharing of information with third party, and lack of users' knowledge about the app.

Nowadays, users can easily enhance the functionalities of their smartphones by connecting them to external devices, such as medical devices, sensors and credit card readers, which allow them to use smartphones in various application domains such as healthcare information systems and retail stores. However, this new development is not accompanied by a corresponding levels of protection; indeed, if an app has permission to use communication channels like Near Field Communication (NFC) and Bluetooth, then it can easily access the devices that communicate with the smartphone on these channels (Naveed et al., 2014).

Android's permissions and sandbox security model mainly protect local resources, such as SD-card, GPS, etc. Each of these resources is protected through one or more permissions, and can only be accessed by Android apps that have appropriate permissions to use particular resources. On the other side, no permissions are allocated to an external device; however, android can only control channels that links smartphone to external device, such as NFC, Bluetooth, Audio port, etc. The main problem here is that many external devices share the same channel in order to communicate with smartphone and many apps also claim the permission to use that channel for different purposes. Consequently, it is very difficult to control unauthorized access on external devices in the presence of those insiders (unauthorized apps that has permission to use the device's communication channel). Naveed et al. (2014) revealed this new security issue for Android devices called as External Device Mis-Bonding (DMB).

A malicious app using DMB attack can surreptitiously collect patient's data from an Android device or spoof a device and inject fake data into the original device's medical app. In data-stealing attack, a malicious app with Bluetooth permission can surreptitiously downlaod a patient's sensitive data from external devices without being noticed, using side-channel infromation to find the right moment for download. In data-injection attack, a malicious app with Bluetooth permission can collect the pairing information of an authorized device and reset the link key, and it can pair with a malicious device to inject fake medical data into the original device's official app. In fact, Bluetooth secure communcation is designed for protecting device to device communication, not to protect communcation between a device and an app (Naveed et al., 2014). External devices may not develop their own authentication systems because these are simple sensors and usually do not have much resources to ensure authentication.

Some of the most common threats to mHealth apps are defined in (Kotz, 2011). Those threats include: (1) identity threats: misuse of patient identity information (PII); (2) Access threats: unauthorized access to personal health information (PHI) or personal health records (PHR); and (3) Disclosure threats: unauthorized disclosure of PII or PHI.

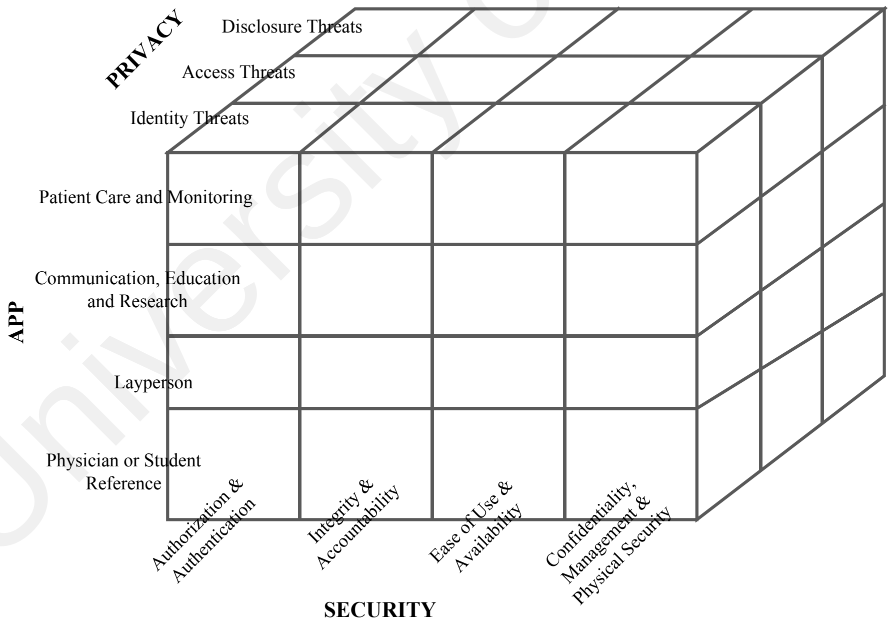

PRIVACY
Disclosure Threats
Access Threats
Identity Threats
APP
Patient Care and Monitoring
Communication, Education and Research
Layperson
Physician or Student Reference
SECURITY
Authorization & Authentication
Integrity & Accountability
Ease of Use & Availability
Confidentiality, Management & Physical Security

Figure 2.8: A Three-Dimensional Model for Classifying mHealth Apps in Terms of Security and Privacy Concerns (Plachkinova et al., 2015)

Plachkinova et al. (2015) report the various types of threats that mHealth app are posing to users. A taxonomy of mHealth apps with respect to security and privacy is proposed by (Plachkinova et al., 2015), and it has three dimensions, as shown in Figure

2.8. Figure 2.9 depicts the proposed taxonomy based on the model in Figure 2.8. Details can be found in (Plachkinova et al., 2015).

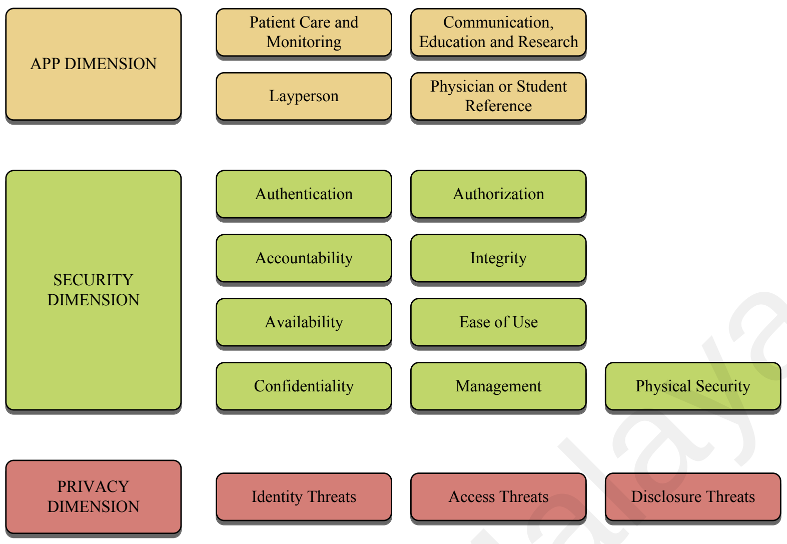


Figure 2.9: A Taxonomy of mHealth Apps - Security and Privacy Concerns (Plachkinova et al., 2015)

2.3.2 An Empirical Assessment of mHealth Apps' Security

To gain a first-hand experience with the security and privacy of mHealth apps, we conducted an assessment experiment on mHealth apps. A sample of 100 mHealth apps has been selected, downloaded, and inspected to identify possible security and privacy issues. Figure 2.10 presents the results of this experiment, summarising the issues that have been found on the sample mHealth apps. During the experiment, it was observed that a lot of apps were disclosing the user data (73 out of 100). Only 11 apps out of 100 were accessing the external devices, and when checked against DMB attacks (for both data injection and data stealing DMB attacks), not a single app was able to defend against DMB attacks. Furthermore, 53 apps were a source for a malicious app to perform privileges escalation attacks. Only 7 apps were using the encryption to secure user data. Some apps were accessing user information that was not necessary at all to fulfil their functionalities. Only, 45 percent of the apps were using authentication to secure user information.

Among these 100 apps, 73 were detected to be leaking the private information. Most of those apps request to access the location, contacts, call logs, phone identity, camera, account information and Bluetooth. Among the 73 apps, 48 apps leak the location information, 43 apps send the IMEI number, and 24 apps leak both IMEI number and location information. Although 32 apps are accessing the contacts, most of them do not need contacts to perform their functions. Further, 21 apps have permission to access Bluetooth, which can cause DMB attack.

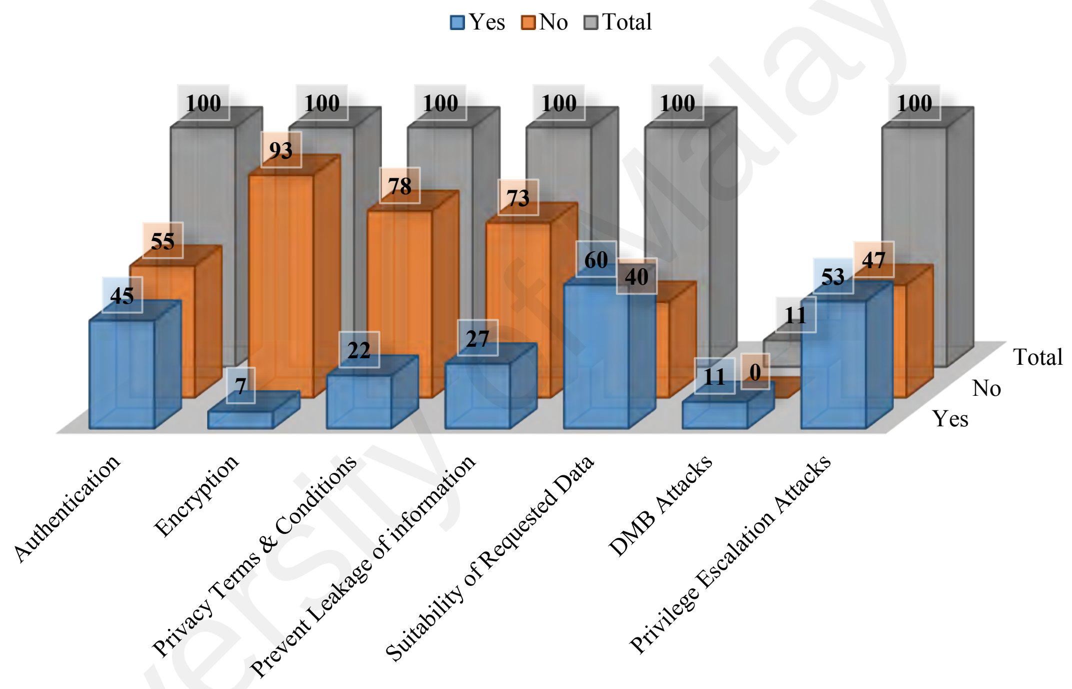


Figure 2.10: Security and Privacy Analysis of Sample mHealth Apps

2.3.3 Existing Solutions for mHealth Apps' Security

Attempting to resolve security issues in mHealth apps is the focus of this thesis. This section presents the most related proposals in the literature, which attempt to address the problems of security and privacy for mHealth apps. It is notable that the number of existing proposals is small, despite the importance of the topic. Below is a listing of those works.

A policy framework is proposed in (Mitchell et al., 2013), which includes the following guidelines to improve the security and privacy for mHealth apps:

- CONTEXT: Provide details about the applications, its capabilities and limitations, and its use of patient information.
- MINIMIZATION: Minimize the amount of information that is collected from/about the patient.
- INFORMED CONSENT/AWARE-PATIENT: The patient should be made aware of how the information will be used and has been used by the application. A more aware patient is likely to make better decisions about trade-offs involving information privacy-security and healthcare benefits.
- OWNERSHIP: The ownership of the information should be well defined meaning who owns the user data even in anonymous form.
- DELETION-AFTER-DEACTIVATION: If a patient has deactivated an application, all information about/from the patient should be deleted.
- SECURE STORAGE: The information should be kept securely at device, server or cloud. To reduce transmission over wireless networks, information that is subject to change can be stored locally, while more static information can be kept at a server.
- END-TO-END: Various weak points in the end-to-end security should be identified and efforts be made to correct these weaknesses in applications, devices, operating systems, networks, servers, among others.

This policy framework is a set of guidelines to follow for the developers and users. These recommendations were developed based on the results of conducting a physical forensics analysis of several widely used mHealth applications.

Another security and privacy solution for mHealth apps was proposed by (Adhikari et al., 2014). Again, the authors proposed a set of recommendations to consumers and mHealth app developers that can be found in Table 2.3. The proposed guidelines are based on a comparative analysis of the 20 most popular mHealth apps at the time of publication. The aim of the analysis was to identify risk and safe features that can help consumers select safe mHealth apps and aid developers in building mHealth apps with appropriate security and privacy measures.

Table 2.3: Recommendations to Consumers and Application Developers (Adhikari et al., 2014)

Consumers	Application developers
Research the app before downloading it	Sensitive consumers' information should always be stored encrypted so that attackers cannot simply retrieve this data off of the file system.
Try to use apps without entering personal information if permitted	Apps should be designed to help patients through the evolution of a disease and provide recommendations
Look for user reviews and the privacy policy of an app, either through the app store or online.	Include user authentication. Provide options so user can safely retrieve their login details if forgotten. Only 10% or 2 apps out of 20 apps ask for user authentication prior to log-in.
Remove data when usage stopped. This may prevent unauthorised use of stored data when consumers no longer use the apps.	Minimise sharing information with third parties or advertisers and ask users to confirm agreement before sharing. 65% or 13 apps shared consumers' information to third parties or advertisers.
Give feedback on product: Users' feedback on the features, privacy and policy, and functions of an app will help the developers to restructure the app appropriately.	Apps should allow consumers to delete their personal information completely. According to the analysis only 5% or 1 apps mentioned in its privacy policy that consumers can delete information completely therefore this criterion need to be improved appropriately.
	Provide user with information about the implementation of security measures and authentication and what how/where their data is stored.

On a more technical level, a static taint-analysis framework was proposed by (He, 2014a), which is shown in Figure 2.11. Static taint analysis works by analysing tainted data flows through Android apps and sending outputs to human analysts or to automated tools which can make security decisions. Again, in this research, the author also proposed some recommendations, listed below:

- Encryption is essential to secure personal data stored on mobile devices.
- When accessing web-based services for syncing users' sensitive data, TLS/SSL is necessary to be deployed throughout the Internet transmission session.
- Even though the network transmission session is protected and encrypted, using third party services to store users' sensitive data must be closely reviewed and users should be informed when it is happening.
- Developer guidelines or training can be helpful in avoiding many of the common mistakes that are rooted from development with poor secure practices.
- Risk assessment provided by authorities can further minimize the security risks that may harm users.

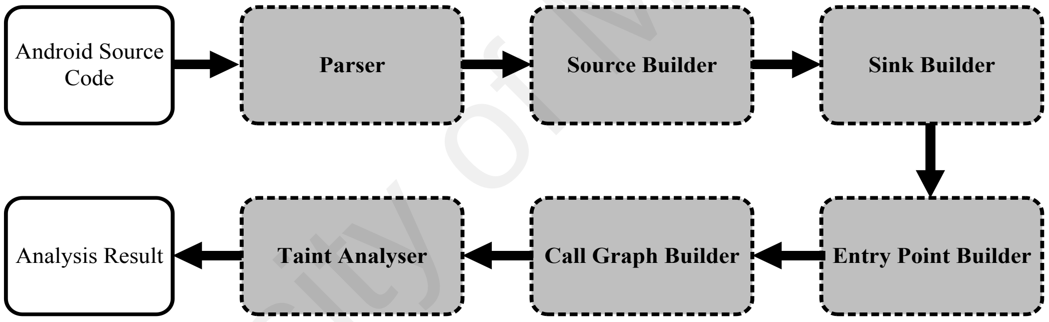


Figure 2.11: Static Analysis System Design Framework

In conclusion, there is no comprehensive solution for mHealth apps to address their security and privacy issues. Most of the research work in the literature just proposed some recommendations and guidelines for the developers and users of mHealth apps. This thesis, on the other hand, is proposing a practical framework for mHealth apps that protects the mHealth apps efficiently and effectively. The research in this thesis does not ignore previous works, however, as building an effective solution involves a range of functions and techniques that can be beyond the capabilities of a single project. For example, the taint-analysis system in Figure 2.11 is employed in this thesis as part of the proposed framework to provide static taint analysis of target mHealth apps.

2.4 Chapter Summary

This chapter provided a broad overview of the necessary background to appreciate the research in this thesis, as well as the most related works in the literature. The main focus of the research is to produce a security framework for Android platform to address the security and privacy issues on mHealth apps. As such, three major sections were covered in the chapter: first, a comprehensive literature-based review of mHealth apps and its related landscape, benefits and challenges was provided. Second, the chapter presented a brief background on the Android platform architecture and its security model, as the main target platform in the thesis. Finally, the crossing of the previous two sections, mHealth apps and Android security issues, was discussed in detail, in terms of specific threats to mHealth apps, an empirical assessment of current mHealth apps, and previous proposals to address the security and privacy of mHealth apps.

CHAPTER 3: RESEARCH METHODOLOGY

This chapter outlines the general methodology adopted in this research study. This research work was divided into four distinct and successive phases, each of which are described in a separate section, starting from the initial preliminary study that suggested the need for the intended security framework (Section 3.1), followed by the design process of the framework (Section 3.2), and then through the implementation of a prototypic proof-of-concept version of the framework (Section 3.3), and finally ending with the evaluation process (3.4). The overall research methodology is illustrated in Figure 3.1.

3.1 Phase I: Preliminary Study

Initially, the literature in three specific fields has been investigated to find and identify the research problem. The problem of this thesis includes the component of mHealth apps from medical informatics, the research on Android security from the field of mobile security, and the interdisciplinary research on the security and privacy of mHealth apps. mHealth apps were explored for the Android platform specifically. First, a detailed survey has been performed on mHealth apps in the literature as well as in available online app stores, and the impact of apps on the healthcare system has been identified.

Second, the existing security model of the Android platform was critically analysed to identify the security issues that need to be addressed. Android security architecture has received tremendous attention, mostly attempting to enhance system protection as a whole for all types of apps hat could be run on the system, and all kinds of data that could be stored on its internal storage. The analysis within this phase focused on a niche class of apps specialized in mobile healthcare. Although most apps share common concerns of security and privacy threats, different classes of apps might require specific requirements that needs to be addressed separately within the system overall security architecture.

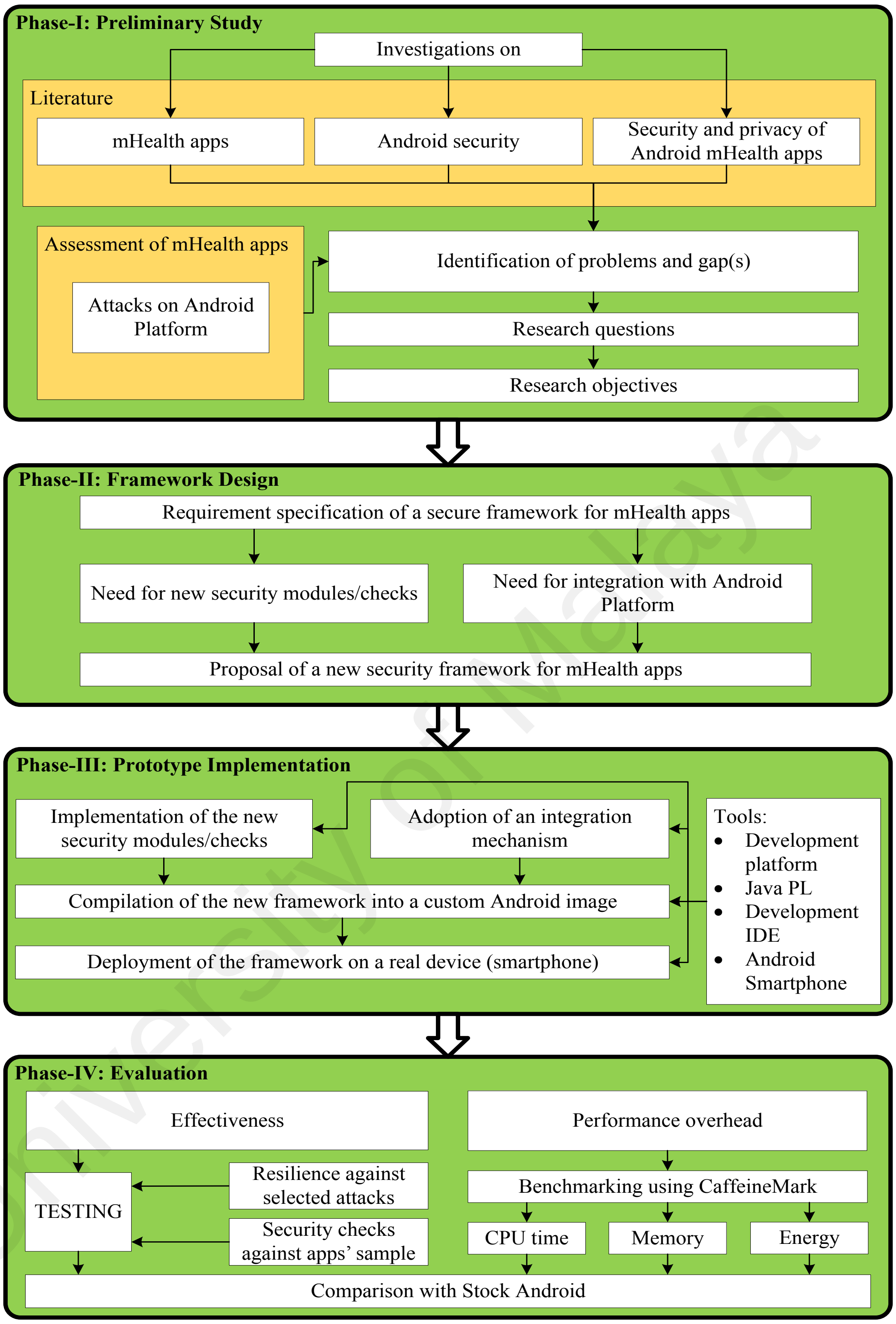

Phase-I: Preliminary Study
Investigations on
Literature
mHealth apps
Android security
Security and privacy of Android mHealth apps
Assessment of mHealth apps
Attacks on Android Platform
Identification of problems and gap(s)
Research questions
Research objectives
Phase-II: Framework Design
Requirement specification of a secure framework for mHealth apps
Need for new security modules/checks
Need for integration with Android Platform
Proposal of a new security framework for mHealth apps
Phase-III: Prototype Implementation
Implementation of the new security modules/checks
Adoption of an integration mechanism
Compilation of the new framework into a custom Android image
Deployment of the framework on a real device (smartphone)
Tools:
Development platform
Java PL
Development IDE
Android Smartphone
Phase-IV: Evaluation
Effectiveness
Performance overhead
TESTING
Resilience against selected attacks
Security checks against apps' sample
Benchmarking using CaffeineMark
CPU time
Memory
Energy
Comparison with Stock Android

Figure 3.1: Conceptual Framework of the Research

Third, previously proposed security frameworks in the literature for Android platform were categorized. As of the security and privacy of mHealth apps, issues related to mHealth apps have been identified, similarly, medical sensors-based threats to mHealth apps have also been explored. Subsequently, the existing solutions to secure mHealth apps were critically analysed to identify potential foundations for the current work as well as issues that need to be addressed, improved or extended.

To study the security and privacy issues of mHealth apps, a practical security and privacy assessment has been conducted on a sample of mHealth apps by actual installations on Android smartphones. This experiment almost revealed the same threats to Android mHealth apps as found in the literature. On the basis of both the literature review and the experimental assessment, the research questions and objectives have been set for this research study.

As discussed in the first two chapters, the result of this phase led to direct the focus of this research study into the development of a security framework for mHealth apps.

3.2 Phase II: Framework Design

The main goal of this research is to develop a security framework for mHealth apps to ensure the security and privacy of sensitive medical data. Based on the finding of the preliminary study in the previous phase, several specifications have shaped out to constitute the desired framework, considering the security issues of mHealth apps and the limited resources of mobile devices. First of all, the requirements of the security framework have been identified, which led to design a number of components that are necessary to implement the required functions, including the performance of security checks against installed apps, information leakage, malware, device connections, and similar cases. To orchestrate all those functions, there is a need for a core component to administer the checks, and take proper decisions probably based on a repository of security policies. The said components form the core of the intended framework, and were grouped in a single layer named as the Security Module Layer (SML).

Crucial to the operation of this layer is to have a low-level access as an entry point into the internals of the underpinning Android platform, in terms of reference monitors and hooks onto the various levels of the kernel, middleware as well as the application level. To keep the design modular, the task of interfacing the SML to the mobile operating system was delegated to another layer of the framework called the System Interface Layer (SIL). As explained in the next chapter, this layer was adopted from a previous research work that was leveraged to provide the necessary foundation for SML. According to this reasoning during the design process, the SML is required to do all the compulsory security functions and SIL is required to work as an integration layer between SML and the Android OS in order to make SML functional.

The resulting overall design was named as MHealth Apps Security Framework (MASF). MASF mitigates various security and privacy threats facing by mHealth apps, such as data leakage, privileges escalation attacks, DMB attacks, and misuse of granted permissions to apps. Furthermore, to protect sensitive medical data and mHealth apps from different attacks, MASF provides several mechanisms that includes fine-grained access control, context-aware access control, and it further provides data shadowing mechanism to protect user private information by providing *fake* versions of the requested data when deemed necessary by the framework. MASF also enables the users to define their own policies according to their requirements.

3.3 Phase III: Prototype Implementation

For validation purposes, the framework has been implemented on a real environment. The implementation of MASF encompasses the two layers designed in the previous phase; i.e. SML and SIL. SML comprises a number of software modules that correspond to the various required security functions, each of which can run in the user mode. Besides, it also implements a database utility to store security policies, implemented in

Android as a SQLite local database. In addition of implementing their own logic, several such modules need to call special API functions to access the internals of the Android platform in order to interfere with the normal operation of the system. In other words, those security modules need to be integrated into Android.

An integration framework was adopted from literature to work as the SIL; this helped in interfacing SML to the Android OS. SIL provides a platform that makes possible the monitoring of various references and actions made by mHealth apps, performing different security checks and also the enforcement of the security policies. Afterwards, the new framework based on both SML and SIL was compiled together within the Android source code into a custom Android copy, which was subsequently deployed on a real phone. During this process the following tools were used: Java programming language, a development station, a development IDE and an Android smartphone. The following steps describe the implementation of the framework prototype:

1. A distribution of the Linux operating system (Ubuntu 14.04) was installed to establish an environment to download and install the stock Android source code.
2. The source code of the stock Android operating system, version 4.3, was downloaded from http://source.android.c om/source/downloading.html.
3. To enforce security checks and hooks, MASF needs an integration layer into the Android source, which is provided by the System Interface Layer. SIL is adopted from another framework in the literature called ASF, and SIL patches were installed into to source code of Android 4.3.
4. MASF is programmed according to the design explained in the next chapter, and then installed onto the modified source code of Android.

5. Thereafter, this customize source code was compiled and installed into a real Android device.

3.4 Phase IV: Evaluation

In this phase of the research, MASF is evaluated and analysed in terms of effectiveness and efficiency. Effectiveness of this framework is evaluated by demonstrating that the framework can successfully protect the security and privacy of mHealth apps and its users. To check the effectiveness, a sample of mHealth apps has been tested against the various attacks. Furthermore, a number of experiments were conducted to evaluate functioning of MASF, such as to test effectiveness of MASF against different attacks, test against malwares, impact of data shadowing, impact of permission restrictions, impact of disabling intents, impact of enabling/disabling system peripherals. Subsequently, false positive and usability test were also performed on Android customised with MASF.

On the other hand, the efficiency of MASF is evaluated by examining the performance overhead on the underlying Android system during the operation of the framework. To check the performance overhead, the following metrics were measured: CPU utilization, memory usage, and energy consumption. Several experiments were conducted to evaluate the performance of MASF. Due to the lack of publicly available implementations of similar frameworks, the results of MASF's performance evaluation were compared to those obtained out of stock Android versions. The performance of mainstream Android copies is considered as a baseline against which the impact of the proposed framework's impact on the system is measured. Therefore, the CPU time, memory usage and energy consumption of Android apps on a stock version (without MASF) are considered as the reference of measurement for the apps performance after installing MASF.

3.5 Chapter Summary

This chapter outlines the general methodology that was adopted to carry out this research study. The research conceptual framework is presented in terms of four main phases. Each phase corresponds to a major distinct step in conducting the research in this thesis in producing the anticipated output. Beyond the phase of the preliminary study necessary to identify the research problem and main objectives, the chapter lists the major steps of designing the target security framework, implementing the proof-of-concept prototype of the framework, and then evaluating the resulting prototype against a set of performance criteria. The last three steps are yet to be elaborated in the next two chapters.

CHAPTER 4: THE DESIGN OF "MHEALTH APPS SECURITY FRAMEWORK"

This chapter presents the detailed design of the proposed MHealth Apps Security Framework, named as "MASF". This framework comprises several components that are divided into two major layers, each of which is discussed below. Section 4.1 gives an overall view of the framework design. Section 4.2 is the main part of this chapter and contains the detailed description of the framework components. Section 4.3 describes some use cases of MASF, while Section 4.4 concludes the chapter.

4.1 MASF Overall Architecture

As discussed earlier in Chapter 2, mHealth apps are facing several security and privacy challenges. To participate in addressing these issues, this thesis proposes a security framework (MASF) that can provide special measures to protect the users of medical apps and their data from malicious or otherwise incompetently written mHealth apps on the Android mobile. In order to achieve such a goal, several functions are expected from the framework.

The framework is expected to inspect apps before and during execution, starting from the point of installation. It is also required to provide security measures beyond the capability of the host system (Android in this case). For example, MASF is required to provide fine-grained access control to sensitive resources that is more effective than the coarse-grained permission system offered by Android. The concept of *context* should also be considered when controlling access to private resources or data. For example, collection of certain medical data is not expected at certain times or places, and the framework should be able to discern the allowed and disallowed actions based on the particular context at which the actions occur.

Furthermore, scenarios that are specific to mHealth apps should be paid special attention. For example, reading to or from medical sensors and devices should be monitored and checked against the current context, and against any known attacks as well. It is also important to enable the user of the framework to dictate a set of policies that is used to derive the decisions of the framework; for example, policies related to what actions are allowed in which context.

In the design of MASF, several components are assigned different tasks to ensure the above functionality. A main component is responsible for performing the security checks and taking decisions related to attempted actions by the various mHealth apps in the system. This main part is called the *manager*. To support its operation, the manager refers to a set of software tools called *checkers*. These tools perform specialized checks on the installation of new mHealth apps, context checks, malware checks, taint analysis and checks related to the communication of external devices. The manager refers to a *policy databas*e and makes use of those special checks to form a complete idea about the adherence of a specific app or actions of an app to the stated policies. It then delegates the enforcement of the policies to an *action-performer*. Another component called the *user-interactor* allows the user to provide policies to protect his/her security and/or privacy.

It is obvious that the functions performed by the various checkers and by the action-performer need special access to the underlining operating system, as they interfere with and control low-level operations of the host system. To modularize the design of MASF, the above described parts including the manager and its supportive components are grouped in one layer, named as the *Security Module Layer (SML)* since it is directly concerned with the security-related functions, and then this layer relies on another layer for interfacing to the Android internal parts. This latter layer is aptly called the *System*

Interface Layer (SIL). The general structure of the proposed MASF is illustrated in Figure 4.2. Further details on SML and SIL are given in the subsequent sections (4.2.1 and 4.2.2, respectively).

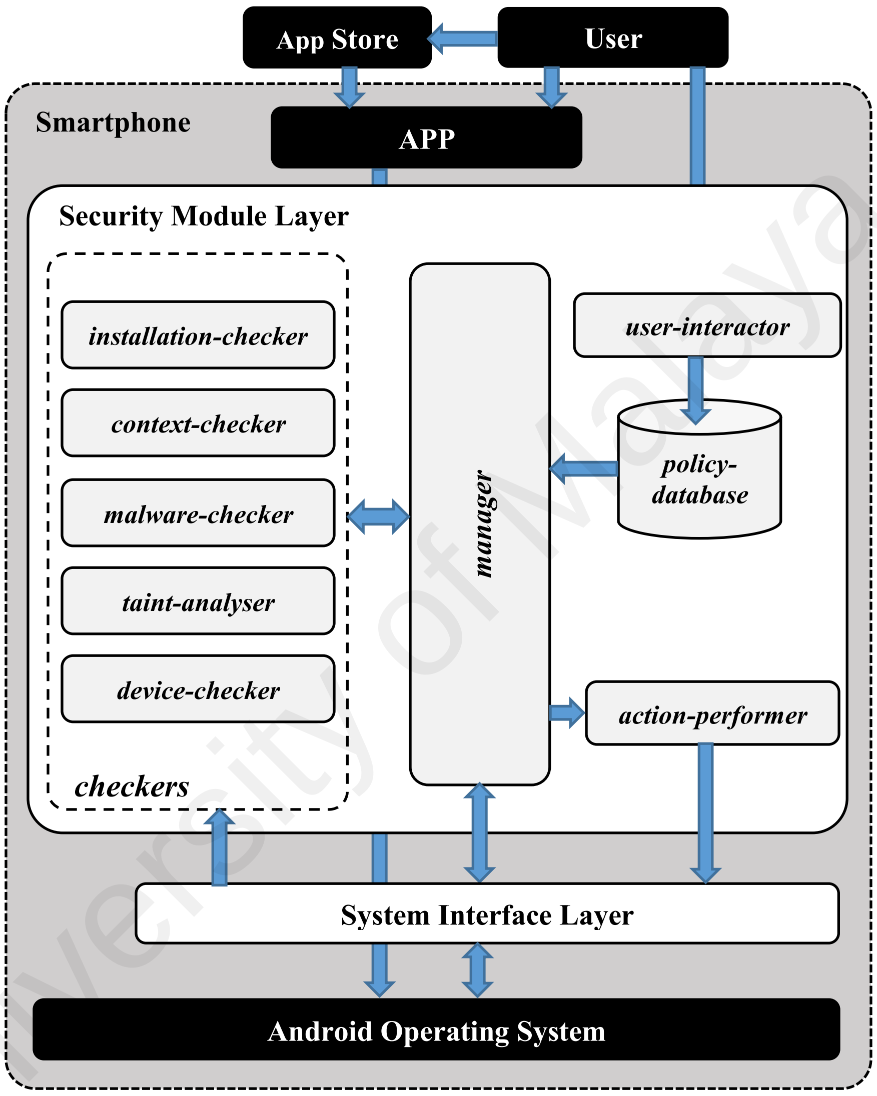


Figure 4.1: The Overall Architecture of the Proposed Framework MASF

4.2 MASF Layered Components

This section details the individual components of MASF. The main two layers of MASF are shown in white colour in Figure 4.1. Most contributions of this thesis lie within the first layer (SML), for which the building blocks are shown in light grey.

4.2.1 Security Module Layer

Security Module Layer (SML) is the main part of MASF, which encompasses the essential functions to protect the system's security and the user's privacy. In order to fulfil its purpose, this layer needs to be able to:

- Monitor references and actions made by mHealth apps
- Examine static and dynamic attributes of mHealth apps
- Receive policies defined by the user of the system
- Enforce user policies

One or more components are dedicated for each of these function categories. Several modules depicted in Figure 4.1 as *checkers* are responsible for providing the necessary security checks and examinations. A *user-interactor* component interfaces with the user to receive user policies (the form of which to be defined later in this Chapter), and a *policy-database* stores them for later use by the framework. The central component that monitors the apps' references and rationalise about their actions in line with the user's policies then make the required decision is the *manager*, which enforces the made decisions through the *action-performer*. The following subsections elaborate the design of each of those components.

4.2.1.1 Security checkers

Security checkers are used by the manager to essentially provide necessary information for making security decisions. In order to provide such information, a checker might perform operations such as simple collection of sensor data, up to sophisticated analysis of the apps code or behaviour. Five types of checks are deployed in SML. The corresponding checkers are explained below.

(a) *Context checker*

According to (Conti et al., 2011), a context could be defined as one of the following aspects: status of some variables (e.g., time, geographical location, temperature, light and noise), the presence of other devices and sensors, a particular type of interaction between the smartphone and user, or a combination of all these aspects. Some security rules depend on the context. In MASF, only time and geographical location are considered for *context-aware* access control.

The *context-checker* is responsible for detecting, reporting and thenceforth updating the context of the device and its apps. This job requires the *context-checker* to subscribe to Android system services, such as the *LocationManager*. Basically, this type of checks is based on the concept of context-aware access control. The *context-checker* collects the physical location parameters (GPS, Cell ID, Wi-Fi parameters) through the device sensors and reports those parameters to the manager upon request. These context checks enable the manager to allow the user of imposing run-time constraints on the usage of sensitive resources based on different contexts (e.g., location, time). The users can describe their constraints using a simple interface (i.e., app), implemented by the user-interactor. Possible rules and policies related to context-aware access control are defined in Section 4.2.4.

(b) *Malware checker*

A modern smartphone provides capabilities that are comparable to low-end computers. As such, it is also facing nearly the same security and privacy issues. Indeed, mobile malware is growing exponentially, and hence the process of malware detection is becoming an essential part of mobile security frameworks. The SML manager performs some checks against malware apps with the help of a *malware-checker* that is dedicated to identifying malicious apps.

Basically, the *malware-checker* invokes an anti-malware app that is installed on the smartphone, and scans the target apps. The results of the scan are reported to the *manager*. If any malware is detected, then the *manager* sends a notification to the user about that particular malware. The malware app is marked as an "untrusted" alongside its source, and this information is kept in the *policy-database* so that future installations of new apps from the same source can be prevented by the *installation-checker*. The untrusted app cannot access other mHealth apps, nor it can access any sensitive system resources.

(c) *External devices checker*

The use of medical sensors and other external devices is growing among smartphone users, and mHealth apps in particular are increasingly expected to support communication with external sensors and monitoring devices. Therefore, SML includes another special tool used by the *manager* specifically to check connections to external devices. For example, the *device-checker* tests against the Device Miss Bonding (DMB) attack that was exposed by (Naveed et al., 2014) and discussed in Chapter 2.

The *device-checker* monitors the apps that are using sensors or external devices (any device outside the smartphone), and checks the information that is being transferred between the smartphone and the external sensors or devices. It is invoked by the *manager* when an app accesses the external devices (e.g., heartbeat reader) through Bluetooth or Wi-Fi, the two most common connectivity options on smartphones. On the basis of security and privacy policies, the *manager* takes the decision whether or not the app is allowed to communicate (send and receive) that information with the external device or not. The *device-checker* also provides the *manager* with information on whether the communication with the external device is encrypted or not.

(d) *Installation checker*

Another tool in the arsenal of the SML *manager* is the *installation-checker*. This checker is unlike the above checkers works only at the time of installation of a particular app. Some security rules have been developed for the *installation-checker* regardless of the installed app or the user-provided policies. When users want to install a new app, *installation-checker* first checks its permission and requested resources, as well as other meta-data that the *manager* needs to decide whether to accept the new installation or raise an alarming notification for the user.

When users install an app, they grant the app all permissions it requests, and have to trust on the way the app uses the granted permissions and smartphone's resources. Although there is now an option available since Android 6.0 that enables users to revoke permissions, still users are not aware of how the app is using the device resources after granting the permission. Apps can easily misuse smartphone resources to compromise user privacy, and can reveal user sensitive medical information in case of mHealth apps. Presently, after granting the permissions to an app, there is no way to impose extra constraints on how, when and under what circumstances those granted permission can be used. The actual problem lies in the inability of users to really understand the implications of the granted permissions and correctly judge their approval.

installation-checker checks the newly requested permissions against few predefined combinations of permissions that can be dangerous for mHealth apps. It is really difficult for mHealth app users to understand the requested permissions and most importantly how a combination of some requested permissions can misuse medical information or can leak sensitive medical data. So, *installation-checker* refers to some dangerous combination of permissions and perform analysis at install time.

At the time of installation, *installation-checker* first extracts the security configuration of an app from the target package manifest, and reports the result to the *manager*, which evaluates the configuration against a collection of security and privacy policies. If an app's security configuration fails to pass all the policies, then the *manager* has two options. The more secure choice is to reject the installation of that particular app. Otherwise, installation can continue after giving the user a warning notification that the app could be harmful for the sensitive medical data. Obviously, this is a less secure option for the users who usually install apps ignoring any warnings.

For example, the *manager* might take the decision to either allow or deny the installation of a particular app based on the following dynamics: 1) don't accept any app from other than Google Play store, 2) don't install an app if the developer(s)' name(s) appear in a blacklist (i.e., users can add a developer's name in that list on the basis of their own experience with apps from that particular developer), 3) don't install an app if it has some strange features (i.e., strange features are a list of some features, such as an app wants to access camera, however, it does not need it to fully perform its functionality).

The policies related to *installation-checker* are defined in section 4.2.4.

(e) *Taint analyser*

Determining how an app uses and reveals privacy-sensitive information is achievable using fine-grained taint analysis, commonly called "taint tracking". A taint is simply a label on a data item or variable. The label assigns a semantic type (e.g., geographic location of device) to the data, and may simultaneously use multiple such types (commonly called a *taint tag*). It is the task of the taint tracking system to (1) assign taint labels at a *taint source*, (2) propagate taint labels to dependent data and variables, and finally (3) take some action based on the taint label of data at a *taint sink*.

The *tainting-analyser* traces the flow of information, and if there is disclosure of sensitive information, for example via transmission of information from an app's component and the Internet, then it notifies the *manager*. The *taint-analyser* can be invoked by the *manager* in several occasions and in tandem with other checkers for tracking sensitive information and comparing against the context, connected device, and security policies.

Static taint analysis technique can keep track of sensitive tainted information through the app by starting at any one from a list of sources and then following the data flow until it reaches any one from a list of sinks. So, it reveals which sensitive data is being leaked to which sink channel, as shown in Figure 4.2.

Static taint analysis has been used in many previous research works (Arzt, Rasthofer, et al., 2014; Lu, Li, Wu, Lee, & Jiang, 2012; Rasthofer, Arzt, & Bodden, 2014). Dynamic taint analysis has also been used in some works such as TaintDroid (Enck et al., 2014). Both dynamic and static analysis can be used to achieve taint tracking. But dynamic analysis may require many test runs to reach appropriate code coverage. In case of dynamic analysis, however, malicious app can be developed to be able to recognize the behaviour of dynamic analysis and pose itself as a benign app to bypass the detection. Dynamic analysis also entails a heavy overhead on the performance of the system if used in real time. For these reasons, static analysis is chosen over dynamic analysis for data leakage detection in MASF.

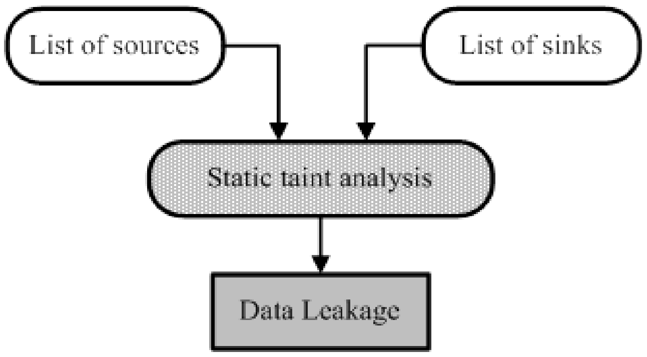


Figure 4.2: Data Leakage Detection with Static Analysis

Implementing static analysis on Android is very challenging due to the special design of Android OS. Existing data flow analysis techniques are not directly applicable to Android apps due to the Android programming paradigm's special multiple entry points. Unlike a Java program, Android app doesn't have a single entry point, and many entry points can be defined for an Android app. As Android apps have four main components (i.e., *activities*, *services*, *content providers* and *broadcast receivers*), Android framework can call the methods associated with these components to start and stop the components. To be able to effectively predict the data flow, static analysis need not only precisely model the life-cycle of components but also need to integrate callbacks for system-event handling, UI interaction and so on.

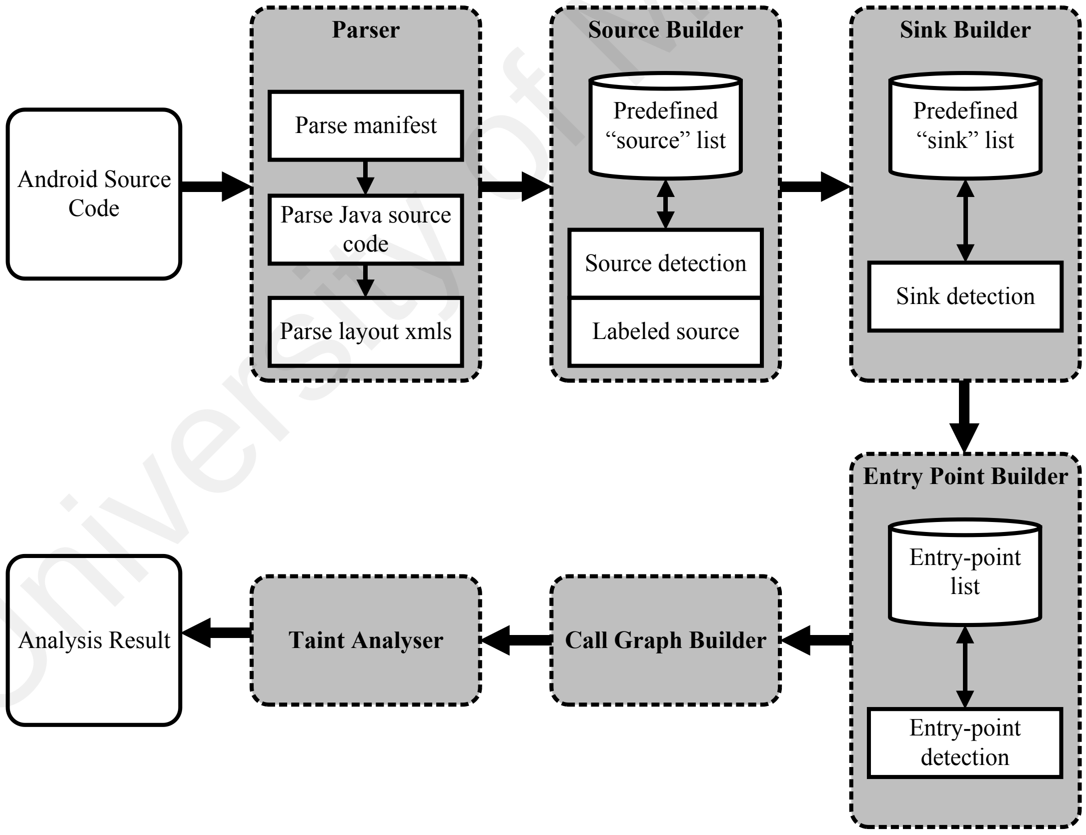


Figure 4.3: Static Taint Analysis System Design Framework

Figure 4.3 shows the design of the *taint-analyser.* This design is basically based on FlowDroid (Arzt, Rasthofer, et al., 2014) and extended by (He, 2014b). Tainting searches through the app for lifecycle and callback methods by parsing various Android-specific files, including the manifest file, layout XML files, Java source files and so on. Then, a list of sources and sinks is constructed from label defined by developers in source code. After that, tainting generates a main method as a single entry point for the Android program from the list of lifecycle and callback methods. This main method is used to generate a call graph for the taint analysis. The taint analysis reports any possible links between the sources and sinks as warnings of potential vulnerabilities to the *manager*.

4.2.1.2 SML manager

The *manager* component is the heart of the Security Module Layer. It performs the core function of monitoring references and actions taken by the mHealth apps and uses other components of SML to both examine those references and actions against the defined policies in the *policy-database* and to enforce the policies through the *action-performer* component. There are many types of actions that *action-performer* can take based on the *manager* direction, such as data shadowing, blocking access to data, granting access to data, revoking permissions, installation control, saving state, and disabling intents.

MASF extends Android's middleware through the System Interface Layer (SIL). For example, the *PackageManager* in Android is responsible for the installation of new apps. The *PackageManager* is extended by the SIL to interface with SML *manager*, so the *manager* can interact with the apps, and can enforce the necessary security and privacy rules and policies for mHealth apps.

Android uses the mechanism of Inter Component Communication (ICC) as the primary method of communication between apps. However, ICC is technically based on IPC at the kernel level, and it can be seen as a logical connection in the middleware. Access control on ICC is important for the enforcement of security and privacy policies in the middleware.

Different types of ICC can be used by apps for communication. First, the most common way for apps to communication through ICC is to establish direct communication links, known as *Direct ICC*. For example, an app can send an Intent to another app, query its content provider, or connect to its service. The *manager* detects this communication through *hooks* provided by the SIL and prevents it in case the sender and receiver apps of the ICC are not allowed communicate or exchange specific information according to the corresponding policies. However, system apps form an exception and direct ICC is not prohibited if either sender or receiver of the ICC is a system app.

Besides the direct ICC, apps can also send *Broadcast Intents*, which are delivered to all registered receivers. Similar to the approaches followed by (Bugiel, Davi, Dmitrienko, Fischer, et al., 2011) and (Ongtang et al., 2012), the *manager* filters out all the receivers of a broadcast intent who are not allowed according to the policies before the broadcast is delivered. Again, system apps have an exception and are not filtered from the receivers list.

A mechanism for apps from different domains to communicate indirectly is to share data in *System Content Providers*, such as the Contacts database, the Calendar, or the Clipboard; ICC reads in this case reads data from such providers. MASF extends the System Content Providers, and upon read access to a provider, all data are filtered by the *manager*.

Mandatory Access Control (MAC) mechanism is provided by the underlying Linux kernel. Such mechanism, implemented by SELinux or TOMOYO Linux, is already a default feature of the Linux kernel and provides mandatory access control on various aspects of the OS, including the file system and the Inter-Process Communication. Thus, by leveraging such mechanism, the *manager* can perform access control on the file system and IPC levels. The file system provides for a further communication channel between apps. Apps can share files system-wide, by writing to a system-wide readable location. Hence, a sending app is able to write such a file and a receiving app would simply read the same file. The mandatory access control mechanism enforces isolation on the file system. By using this, the *manager* makes sure that system-wide readable files can be read by other apps only if both the reader and writer are trusted apps.

To prevent any communication of apps via Linux IPC (e.g., sockets, pipes, shared memory, or messages). The *manager* leverages the same domains already established for the file system access control. Thus, other third party apps cannot establish IPC with mHealth apps. However, system apps form an exception, as denial of communication to system apps can be harmful for the system and other apps, or can affect the functionalities of apps.

A further channel that should be considered here is the Internet network, i.e., network sockets used for communication through Internet protocols (such as TCP/IP). By using these sockets apps can communicate with remote hosts, however they also can communicate with other apps on the same platform. Thus, the MASF has to take both local and remote communication into consideration. To enforce access control here, the *manager* component employs a firewall to modify or block the Internet socket based communication. To locally enforce isolation between mHealth apps and other third party apps, the *manager* prohibits any communication from a local network socket of other third party apps to another local network socket on the platform. This might look as over-restrictive, but it is a very reasonable enforcement, as apps exist on the same platform so they should use lightweight ICC to communicate instead of network channels.

4.2.1.3 Policy database

The *policy-database* component serves as the main repository to store all the policies and rules, which are discussed later in this section. All policies and rules are stored in this database, and the *manager* accesses this database during the various checks to ensure the implementation of all policies. Rules and policies stored by the *policy-database* can be updated and modified by the user of the smartphone.

4.2.1.4 User interactor

The *user-interactor* component is an app that interacts with the user. User can interact with MASF to create, update, and delete security and privacy policies. Furthermore, for context-related policies, user can also create, update and delete contexts. The *user-interactor* stores all the policies and related information in the *policy-database*.

4.2.1.5 Action performer

Another important component at the disposal of the *manager* is the *action-performer*. This component performs the decisions taken by the *manager* to enforce the various policies. There are a number of actions that the *action-performer* can take, including *blocking*, *data shadowing*, *granting*, *disabling intents*, *saving device state* and *revoking permission*. The functions of each action and its description is given below.

(a) *Blocking*

The *action-performer* might block outgoing communications tainted by sensitive data. This control is applied in MASF to limit the misuse of sensitive medical data. Blocking happens usually in the case of conflicts of information flow or other functionality with privacy policies. When a conflict occurs, the *manager* decides to block the information flow or functionality which created the conflict with the policies. However, the blocking option can sometime harm or crash the app, as it needs the data to continue. There are two types of blocking that MASF provides and implements through the *action-performer*.

In the first type of blocking, app cannot communicate with other apps, and cannot send data through the Internet. In the second type of blocking, apps can communicate with each other, but cannot send/receive data through the Internet. The latter option is necessary when it is compulsory to make data available for other apps in order to accomplish some essential task.

To block the data that is being exchanged through network sockets, first the *manager* intercepts calls to the network stack to associate domain names with open sockets and detects when tainted data have been written to a socket. When an output buffer comprises tainted data, the *action-performer* just drops the buffer and choose one of the following actions: dropping the offending message covertly, or misleading the app by indicating that the buffer has been sent.

(b) *Data shadowing*

As mentioned above, blocking sometimes cannot be possible, therefore it is mandatory to provide an app with data so that it can continue functioning. To solve this issue, the technique of *data shadowing* option is used. The *action-performer* implements data shadowing to prevent apps from accessing sensitive medical data. It conceals the actual user data stored on the device and provides a fake copy of sensitive data (e.g., medical history, etc.) instead of the original one.

Android apps use the file system to access the microphone, camera, logs, etc. When apps try to open these resources and the *manager* decides to deny their attempts, the *action-performer* shadows all sensitive data, such as the browser history and bookmarks, contacts, accounts information, subscribed feeds, SMS/MMS, and calendar entries, by returning a fake set of data. When apps request the device's location, the response is a set of fake location information. When apps request the device's phone state, the returned value is a fake phone number with a fake app-specific device ID (IMEI). The *action-*

performer can also return a fake version of the SIM serial number, voice mail number, and subscriber ID (IMSI); however, very rare apps request this type of data.

(c) *Granting*

Another action that can be taken by the *action-performer* is to provide the original requested data. The SML *manager* takes this decision when it thinks the requesting app can be trusted with private user's data.

(d) *Installation control*

This type of action is related to the checks at installation time of new apps. If the *manager* decided based on the information from the *installation-checker* that an app does not satisfy the policies, then the *action-performer* either warn the user and allow the installation process to proceed, or it might prevent the installation of the app altogether.

(e) *Revoking permission*

MASF supports also the action of revoking selective permission(s) from an app for a particular period of time, or a particular location, at runtime.

(f) *Disabling intent*

This action intercepts and drops the specified intent message. MASF can enforce a number of controls on different activities by intercepting intents, such as to launch an app (prevent certain apps from running on the device), app installation and uninstallation (prevent an app from sending an intent to install or uninstall an app), services (prevent apps from starting background services), lock and unlock the device (prevent requesting pin code to unlock the device), broadcasts (prevent apps from broadcasting Intents), and app multitasking (prevent running multiple user-app simultaneously).

(g) ***Saving device state***

This action disables toggling the state (ON/OFF) of the specified system peripheral. It is particularly relevant when mHealth apps exchange data with medical sensors or other kinds of external device, so that users can restrict the access of other apps to system peripherals (e.g., Bluetooth and NFC).

4.2.2 System Interface Layer

The System Interface Layer (SIL) is the second layer of MASF. SIL acts as an interfacing layer between the first SM layer described above and the underlying Android operating system, and provides the former with access to the latter. SIL is as essential part of the framework to deploy all the enforcements issued by the SML into the Android OS. SML implements a logic that receives necessary input and produce desired output. The required input initiates from the internal workings of other apps, which are well beyond the access of any normal application in the Android architecture. The desired output also interferes with the system-level functions and cannot be achieved using normal apps. There is a need for special modifications of the stock Android code base that offers entry points to the internals of the system, which can be used by the upper layer of the proposed framework.

SIL inserts some hooks on different layers of the Android OS, including the kernel, the middleware layer, and the application layer, to provide the required interface for the SML in order to receive the necessary input from the workings of other mHealth apps, and to enforce the suitable actions on the system level. Separating SML from SIL makes the design of MASF modular, so that the main logic (checks) of the framework can be deployed later on any platform by using a compatible interface with that platform. It also allowed the implementation of MASF to utilize available solutions in the literature that are policy-independent and provides exactly the required interfacing to the internals of

Android without imposing any specific security logic, which is to be left for the design of MASF.

The SIL that has been used to implement MASF is shown in Figure 4.4. This SIL is originally proposed by (Backes, Bugiel, Gerling, & von Styp-Rekowsky, 2014), and named as Android Security Framework (ASF). ASF is a general, extensible and policy-agnostic security infrastructure for Android. The basic idea behind ASF is to extend Android with a new security API. This API allows to easily author, integrate, and enforce generic security policies. ASF allows security experts to develop Android security extensions against a novel Android security API and to deploy their security models in the form of modules as part of Android's platform security. In essence, the Security Module Layer in MASF can be thought of as a special security extension to ASF in the latter terminology, which is specifically designed with mHealth apps in mind.

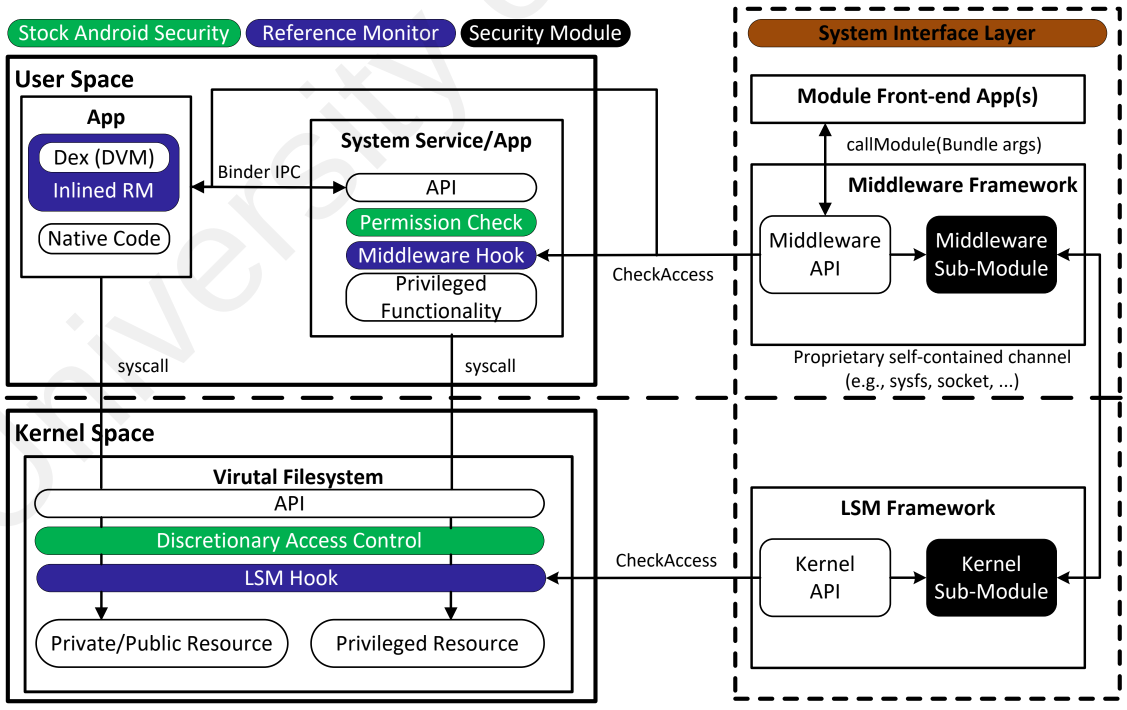


Figure 4.4: System Interface Layer

4.2.3 Other External Components

This subsection completes the picture of the layered components of MASF by listing the external components that are interacting with the main two layers of SML and SIL.

In a sense, a security framework to that controls mHealth smartphone apps and protect the security and privacy of users would naturally run on a mobile platform and be utilized by users; so those entities can be viewed as an integral part of the framework. These components are listed below.

4.2.3.1 Android operating system

Android OS was selected to deploy the proposed framework because it is the most popular and widely used smartphone operating system, and it is open source as well. Chapter 2 explains the Android OS and its working in some detail. The proposed framework cannot work on a stock Android copy and needs special modifications to allow for the intended functionality. These modifications are implemented in the System Interface Layer on which the Security Module Layer is built.

4.2.3.2 Apps

A smartphone application (*app* for short) is a piece of software designed to accomplish a particular purpose. There is a huge number of apps available on online app stores, classified into many categories and can perform a large number of functions. Android apps typically contain one or more of four software components: *activities*, *services*, *content providers* and *broadcast receivers*. These components can interact with each other within the same app or across other apps' using *intent* messages. In this research, the focus is on mHealth apps, which are growing exponentially alongside the security threats targeted at them.

4.2.3.3 App store

This is basically a collection of online accessible apps, and mobile users download apps from the app store. For Android apps, the official and most popular app store is Google Play by Google Inc.; however, there are number of app stores available from untrusted third parties, and they may provide malicious apps as well.

4.2.3.4 Users

MASF serves users in the first place, and it allows users to define policies according to their requirements. Users basically provide policies to secure medical data. Users work as input for the *policy-database* in the SM layer, through the *user-interactor* component, which is a custom Android app.

4.2.4 MASF Policies

In addition to the predefined security rules and policies in the framework, policies in MASF can also be provided by the users through the *user-interactor,* and they form an important component of the SML. Security and privacy policies feed the *manager* component and define the decisions made by the *manager* in response to the various actions and references requested by the apps. The *policy-database* stores all the policies, including user-defined rules.

These rules ensure the confidentiality and integrity of sensitive medical data. Most of the previous research is based on coarse-grained policies to enforce the security. In this framework, users are able to enforce fine-grained security and privacy policies, which allow users much more flexibility to control access to the sensitive medical data; for example, users can change security policies while an app is running, and they can control access to different resources based on the current context (e.g., location and time).

Currently, most of security controls on smartphones are based on policies per app, and normally policies are set at installation time. App developers declare all the required permissions in the compulsory *manifest* file, in order to be able to interact with other apps and to access protected parts of the system's API. Normally, users grant all those permissions at installation time. Granting all permissions together at installation time is a coarse-grained control: a user usually has no idea how the permissions are being exercised after the installation. Further, Android does not have a mechanism that allows policies to grant access to a particular resource only for a specified number of times, or only in specific contexts (e.g., location or time), or only under some special circumstances. To resolve the above issue, MASF provides a number of security and privacy policies that are enforced by the SML *manager* with the help of the various *checkers* introduced in the previous subsection as well as the *action-performer* component.

MASF supports several types of policies. One major class of policies is based on the current context while using the apps. Context-related policies resolve very complex problems that are usually faced by mHealth apps. Sometime there is a need to give a permission based on the context for a particular communication or connection, and later on the need calls for revoking that permission from the connection. In other words, control of the access depends on the context.

As defined earlier, a context could be one of the following aspects: status of some variables (e.g., time, location, temperature, light and noise), the presence of other devices and sensors, a particular type of interaction between the smartphone and user, or a combination of all these aspects. In MASF, users, developers and trusted third parties are allowed to define context-related policies, and the framework can enforce the policies at run-time when the smartphone is within a particular context. MASF uses context-related security policies to extend the control of the users and trusted third parties to secure the information. By definition, a security policy divides the system states into two sets: a set of authorized (secure) states, and a set of unauthorized (insecure) states. Therefore, a context-related security policy is a security policy that separates the authorized states from the unauthorized states of the system based on the context.

In MASF, the policies are defined as a set of restrictions and corresponding conditions. The restrictions are applied to apps whenever the conditions are encountered. Policy restrictions represent the constraints on accessing the device resources, services, system methods, functions, and user data. Policy conditions on the other hand define the situation in which those restrictions should be applied, such as at a certain context or when a suspicious combination of permissions is requested at installation.

A number of policies are defined in this subsection, some are applied at the time of installation of new app, while other policies are enforced on the basis of context (e.g., time and location). MASF policy rules are based on (Shebaro, Oluwatimi, & Bertino, 2015) and significantly extended to consider policies for various other security aspects of the framework. In the following, the two aspects of policies, restrictions and conditions, are described respectively.

4.2.4.1 Policy restrictions

Policies impose restrictions on the use of apps, subject to certain conditions. To define a policy restriction, the following sets are defined first:

- $\boldsymbol{APP}$: is the set of subjects representing the device apps
- $\boldsymbol{RES}$: is the set of protected objects representing the services, resources, user data, permissions, and functionalities available for the apps
- $\boldsymbol{ACT}$: is the set of restriction actions that can be applied through MASF policies

The set of subjects APP is composed of the $PackageNames$ of all apps installed on the device. Additionally, a special character * is added that represents all the installed apps. This special character is useful for policies that need to be enforced on all apps, instead of creating the same policy for every app. Further, it is assumed that each object from the set RES has an associated type from the set (*permission*, *intent*, *data*, *system peripheral*). Let r be an object from the set RES; notation $t(r)$ denotes the type of r. The set of actions ACT defined for MASF includes the following actions, as discussed above in section 4.2.1.5 in the context of *action-performer* role: preventing installation, revoking permissions, disabling intents, shadowing data and saving device state. Note that revoking permissions can be used to block specific data transfer through the network.

i. Definition 1 (*policy restriction*):

Let $p \in APP$, $r \in RES$, $a \in ACTION$, and notation $t(r)$ denotes the type of r. A policy restriction (PR) is defined as the tuple $[p, r,\ a]$ such that:

$$a = \begin{cases} revoke\ permission & if\ t(r) = permission \\ disable\ intent & if\ t(r) = intent \\ shadow\ data & if\ t(r) = data \\ save\ state & if\ t(r) = system\ peripheral \end{cases}$$

4.2.4.2 Policy conditions

Access control policies are enforced on the basis of context, and other conditions such as the set of permissions requested upon installation.

(a) *Context conditions*

In MASF, device location and a time interval are considered for context. The device location data is captured from GPS and users can assign logical location names in which the device is located. A policy time interval is introduced in MASF, which represents the specific time period within which a policy should be enforced. The date and time is represented in the following format $DD - MM - YYYY - hh{:}mm{:}ss$. In addition, MASF uses the R flag to define recurring events. The value of R is drawn from the set $[O, D, W, M, Y]$, which is defining the event frequency: $O \longrightarrow Once, D \longrightarrow Daily, W \longrightarrow Weekly, M \longrightarrow Monthly,\ and\ Y \longrightarrow Yearly$. Date and time define the policy time interval and an event is recurred based on the value of R (i.e., the value of R defines the frequency with which that particular policy condition should be checked). For example, to set an event that occurs every Friday for 8AM to 4PM for six months, R would be set

W and the time interval would be set to event date-time, such as starting on $01-01-2016-08:00:00$ and ending on $30-06-2016-16:00:00$.

ii. Definition 2 (*context condition*):

A context c could be defined as one of the following aspects: time or location. A context is a condition for enforcing an associated restriction. A context policy is the combination of the condition and restriction associated with it; one context can be associated with one policy and one policy can be associated with only one context (one-to-one relation).

Let L be a location name and representing a particular location, and let $[S, E, f]$ respectively be the starting time, ending time, and frequency, which define when a particular policy is going to be enforced. So, a context condition for the policies is defined as the tuple of $[L, (S, E, f)]$.

iii. Definition 3 (*active context (active policy)*):

A context c is called as an active context at a given time $\boldsymbol{t}$, if the required considerations that the context describes are verified. A policy $\boldsymbol{P}$ that is associated with active context $\boldsymbol{C}$ is called as active policy. There exists a possibility of more than one active context at the same time.

(b) *Installation conditions*

For installation polices, promising work has been already done in this area by (Enck et al., 2009). In MASF, an installation condition is represented by the result of a test to whether an installation rule is passed or failed. Table 4.1 describe some useful installation rules for MASF to be checked at the time of installation. This table is based on examples from (Enck et al., 2009) and extended with few conditions.

Table 4.1: Sample Installation-Time Policy Rules

#	Sample conditions
1	An app must not have any of the following combinations of permissions: 1) Phone state, internet, and record audio 2) Outgoing or incoming calls, record audio, and internet 3) Receive SMS and write SMS 4) Send SMS and Write SMS
2	An app should be downloaded from the user-listed app stores. For instance, if user listed only Google Play store for downloading of apps, then apps cannot be downloaded from any app stores except Google Play store.
3	User can block specific developers. An app cannot be downloaded from a list of developers who the user has blacklisted.

The rules defined above only require the knowledge of the permission labels that are requested by an app, as well as the action strings used in the intent filters. A rule indicates the combinations of permission labels and action strings that should not be used by third-party apps. Each rule is the conjunction of sets of permissions and action strings received.

A simple logic to represent a set of rules can be defined. Let IR be the set of all security and privacy rules for *installation-checker*, and let IP be the set of all possible permission labels used by the app and IA be the set of all possible action strings used by *activities*, *broadcast receivers*, and *services* to receive *intents*. Then, each rule ir_i is a tuple $(2^{IP}, 2^{IA})$[1], where $ir_i \in IR$. Each rule ir_i can be defined with the notation $ir_i = (IP_i, IA_i)$ to refer to a specific subset of permission labels and action strings for rule ir_i, where $IP_i \in 2^{IP}$ and $IA_i \in 2^{IA}$. Next, a configuration based on package manifest contents is defined. let IC be the set of all possible configurations extracted from a package manifest. Here, it is only required to get the set of permission labels used by the app and the set of action strings used by its *activities*, *broadcast receivers*, and *services*. Then, each configuration ic can be defined as a tuple $(2^{IP}, 2^{IA})$, where $ic \in IC$. Consequently, the

[1] A standard notation 2^X is used to represent the power set of a set X, which contains the set of all subsets including $\emptyset$.

notation $ic_t = (IP_t, IA_t)$ is used to refer to a specific subset of permission labels and action strings used by a target app t, where $IP_t \in 2^{IP}$ and $IA_t \in 2^{IA}$. It is possible now to define the semantics of a set of installation rules. A function

$$fail : IC \times IR \rightarrow \{true, false\}$$

is defined to test if an app configuration fails for a particular rule ir_i. Let ic_t be the configuration for target app t and ir_i be a rule. So we can define $fail(ic_t, ir_i)$ as:

$$(IP_t, IA_t) = ic_t, (IP_i, IA_i) = ir_i, IP_i \subseteq IP_t \wedge IA_i \subseteq IA_t$$

So the permissions and actions strings (i.e., the permissions and actions strings that are defined in the rule should not access any app, because it is a dangerous combination, and the app who is accessing this can act maliciously) must not be accessed by the target app, but according to the above equation it is in the target app configuration, so this app fails the defined rule.

iv. Definition 4 (*installation condition*):

Let $F_{IR}: IC \rightarrow IR$ be a function that returns the set of all rules in $IR \in 2^{IR}$ for which an app configuration fails:

$$F_{IR}(ic_t) = \{ir_i | ir_i \in IR, fail(ic_t, ir_i)\}$$

The configuration ic_t passes a given rule-set IR if $F_{IR}(ic_t) = \emptyset$. Hence, the result of this function $F_{IR}(ic_t)$ defines the installation condition for an app, which is tested upon installation and, if a non-empty set, will lead to the enforcement of the associated restrictions defined by the corresponding installation policy.

4.2.4.3 Policy definition

A policy in MASF is the combination of a policy condition and the policy restriction associated with it.

v. Definition 5 (*policy*)

Let PR be a policy restriction as defined in *definition 1* and c be a condition such as the one defined by *definition 2* or the one defined by *definition 4*. A policy P is defined as a tuple $[PR, c]$. The following example illustrates a *context* policy:

$$\boldsymbol{P} = [[*, \boldsymbol{android.permission.CAMERA}, \boldsymbol{Revoke\_Permission}],$$
$$[\boldsymbol{Hospital\_A}(\mathbf{01-05-2016-10{:}00{:}00}, \mathbf{30-06-2016-16{:}00{:}00}, \boldsymbol{W})]]$$

The example policy shown above disables all the apps from having the camera permission weekly between 10.00 AM to 4.00 PM for two months in Hospital-A.

4.3 Framework Operation

This section explains interactions between different components of MASF. Some of the use cases are presented to describe the functionalities of the framework and how it protects the user data from unauthorized access.

4.3.1 Use Case I: Installation of a New App

To explain how *installation-checker* works and how the corresponding installation policies behave during the installation of a new app, a use case (installation of a new app) is depicted in Figure 4.5. This use case shows how MASF and its corresponding policies behave when user tries to install a new app. Arrows 1 and 2 are showing the installation of a new app from the app store. For installation, Android app installer first handles the app, and before it completes its installation process the SML *manager* intercepts the installation process and asks the *installation-checker* to check the app manifest file and look at the permission labels requested by the app, as well as the action strings used in the intent filters (arrow 4). This checker then returns a result to the *manager* as per *definition 4* above (arrow 5). The *manager* reads the relevant installation policies from the *policy-database* (arrow 6), and follows the policy defined in *definition 5* to decide whether to prevent or proceed with the installation process. The decision is delegated to the *action-performer* (arrow 7), which enforces it upon Android installer (arrow 8).

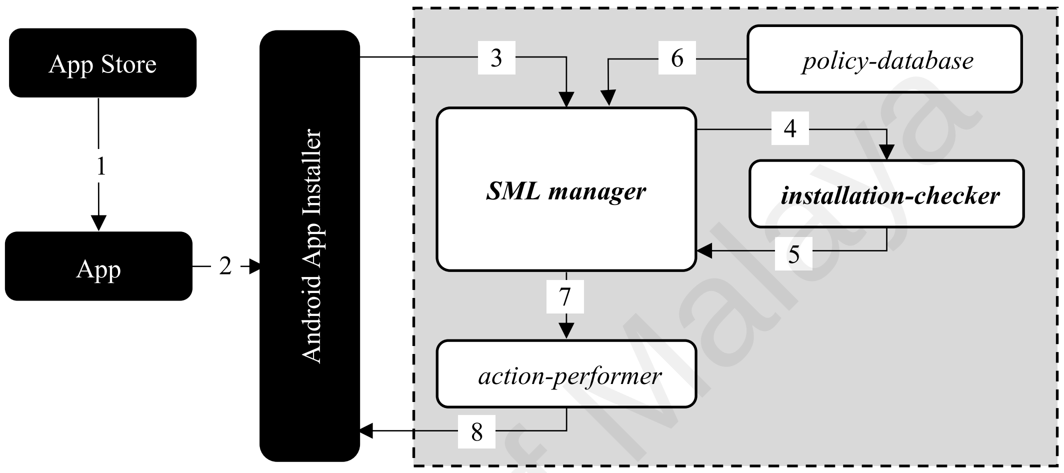


Figure 4.5: Use Case 1 - Installation of a New App

4.3.2 Use Case II: Privacy Enhanced Content Providers and System Services

System Services and *Content Providers* are an integral part of the Android application framework and implement the API exposed to third party apps. Prominent services are the *LocationManager* and the *Audio Services*, while prominent content providers are the *Contacts* app and *SMS/MMS* app. Android enforces permission checks on access to the interfaces of these services and providers.

However, the default permissions of Android are too coarse-grained and protect access only to the entire service/provider but not to specific functions or data, and once these permissions are granted then users are not aware how the permissions are being used. Thus, the user cannot control in a fine-grained fashion which sensitive data can be accessed, how, when and by whom. For example, the Facebook have access to the entire contacts database although only a subset of the data is required for their correct functioning.

MASF deploys some hooks into each service interface and function. The SIL inserts a hook into each of the *AutdioService*, *LocationManager*, and *SensorManager* to achieve fine-grained access control on these functions. To show how MASF protects the content providers and system services, an example of protecting a content provider (e.g. the *Contacts*) is shown in Figure 4.6.

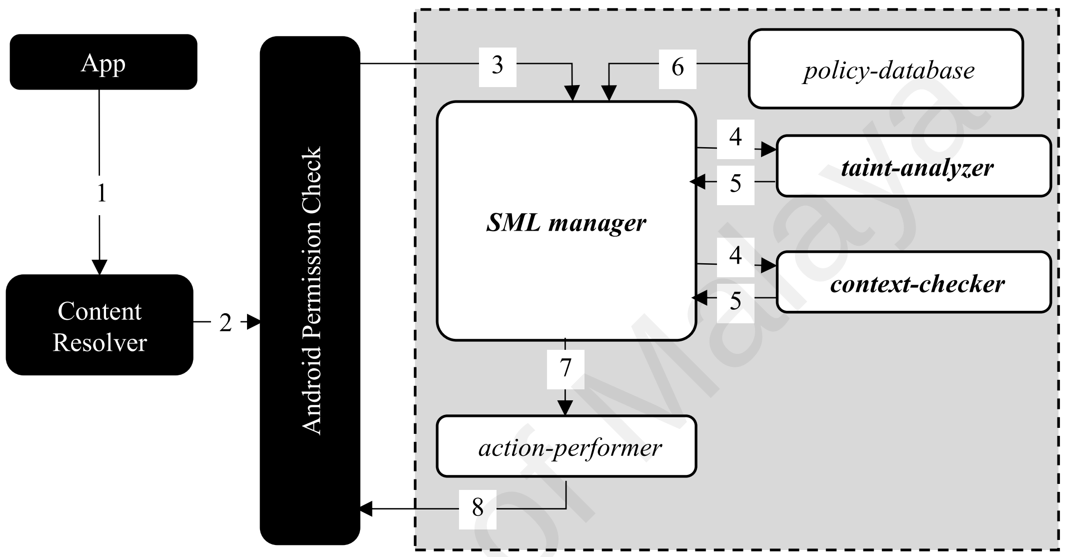


Figure 4.6: Use Case 2: Protecting Contents

When an app requests for some content such as the contacts, the request first goes to a content resolver (arrow 1). Afterwards it goes to Android permission check, a mechanism that is provided by Android OS to check the permissions granted for an app before allowing it to access the protected resources (arrow 2). The *manager* takes the control here (arrow 3), and decides whether it is safe to grant the app an access to the requested content. In order to reach a decision, the *manager* might resort to the *taint-analyser* to examine the app and track the path of the requested data through its code. This could reveal whether the requested data are leaked out to the untrusted domain or are just consumed locally (arrows 4 and 5). The *manager* can also make use of the *context-checker* to check the conformity of the access to context-related policies (arrows 4 and 5). Based on the results and the policies read from the *policy-database* (arrow 6), the *manager* either instructs the *action-performer* to grant the required access or to take one of the following actions: provide *shadowed* contents, or deny the access to contents.

4.3.3 Use Case III: Context-Aware Fine-Grained Access Control

As mentioned earlier, MASF can provide a context-aware and fine-grained access control mechanism, so that the user can control the access to sensitive resources according to the current context (e.g., location or time). Figure 4.7 shows an example of providing context-aware access control based on the current location.

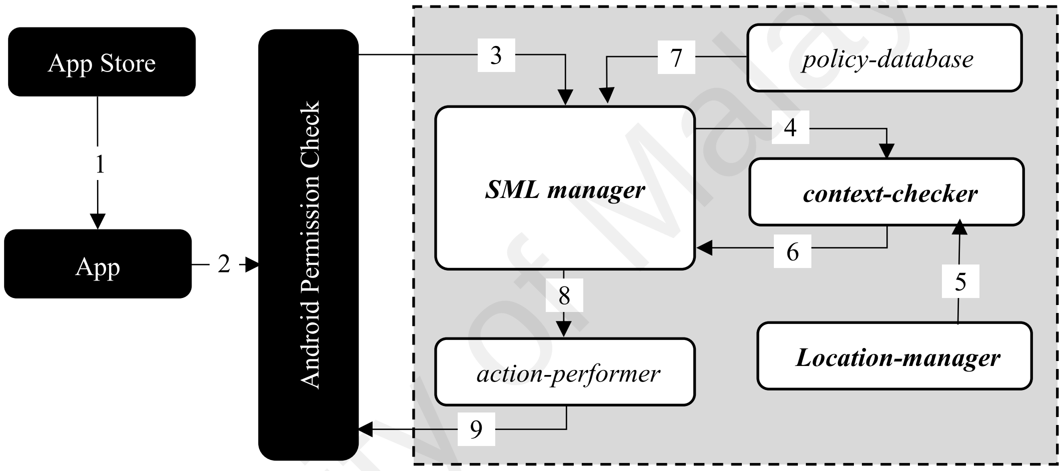


Figure 4.7: Use Case 3: Context-Aware Fine-Grained Access Control

A context-aware fine-grained access control mechanism is demonstrated in Figure 4.7. This figure is showing a use case for location-based context-aware fine-grained access control. As depicted in the figure, when an app sends a request to use a system service or to access resources (arrows 1 and 2), first the Android permission check system verifies the permissions for the corresponding activity, if it allows to access the service or resource, then the SML *manager* intercepts the request (arrow 3). From there, the *manager* refers to the *context-checker* (arrow 4), which in turn reads the location data from Android's *location-manager* (arrow 5), and returns a result to the *manager* in terms of a tuple that was defined in *definition 2* in section 4.2.4.2 (arrow 6). The *manager* then compares the returned context with the defined context policies in the *policy-database* access (arrow 7). To achieve its job, the *context-checker* refers to the system-provided *location-manager* (arrow 5) to learn about the current GPS and other location-related measurements. Finally, the enforcement of the decision made by the *manager* based on the context policies is delegated to the *action-performer* (arrow 8).

4.3.4 Use Case IV: Mitigating the DMB Attacks when Connecting to Devices

This use case defines how MASF mitigate the DMB attack (DMB attack definition is provided in Chapter 2). Considering this case is important in mHealth apps, as these apps usually communicate with medical sensors and other external medical devices. mHealth apps generally collect measurements from external devices (e.g., heartbeat reader) and sensors to accomplish their medical services. During this process, medical information might be stolen by an adversary through a malicious app that is installed on the same smartphone, or the adversary can inject fake data, which could be very dangerous for the user.

MASF deploys the *device-checker* to detect these attacks. When an app tries to access data from external resources. Figure 4.8 illustrates the architecture for Bluetooth socket communication on Android 4.3, and it also depicts how MASF works against DMB attacks. Android platform supports pairing a device programmatically, using the system calls $setPairingConfirmation$ and $setPin$ or $SetPasskey$ of the $BluetoothDevice$ class. To unpair a device, the app uses the API call $removeBond$. Otherwise, the app can invoke the built-in settings program to control the Bluetooth adapter. In both cases, an IPC request needs to be sent to $AdapterSevice$ to control the Bluetooth device. Once a bond is established between the app and the device, the app can make a socket connection to access the device. It first needs to talk to the $BluetoothAdapter$, to get a list of paired devices. From this list, the app identifies the target device (MAC) and further requests a socket through the object $BluetoothDevice$. This request is also delivered using an IPC, through the $IBluetooth$ interface, to $AdapterService$, which creates the socket for the connection.

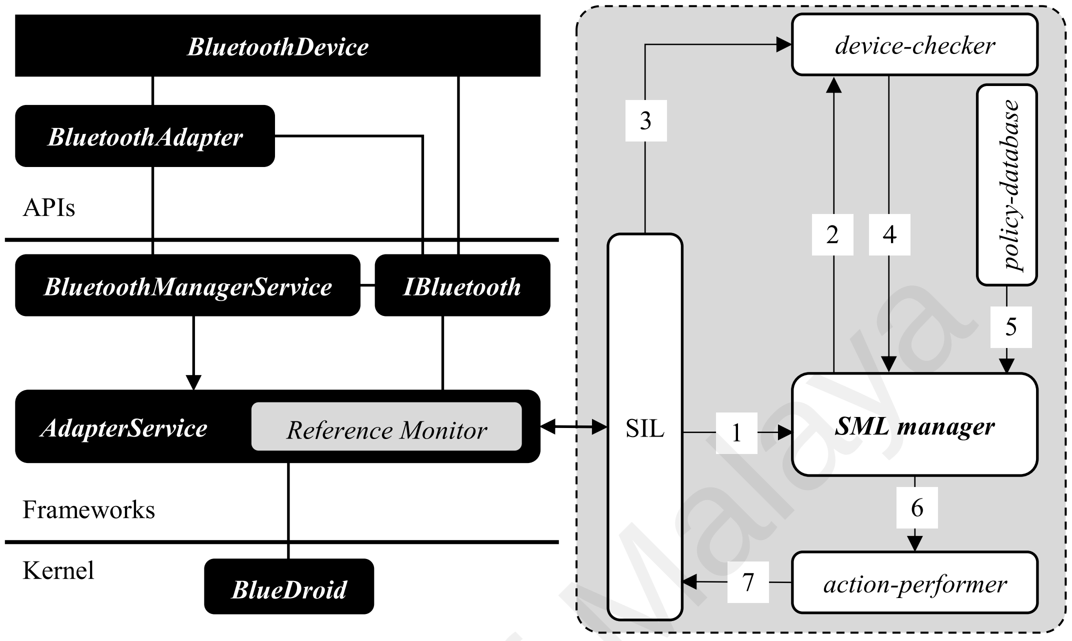


Figure 4.8: Use Case 4: Collect Data from External Devices/Sensor (DMB Attacks)

Once an external device is activated, it is paired with its authorized app by the user. MASF observes this pairing process and then generates a bonding policy that associates each device (name, MAC) to its official app (UID). In MASF, users can also define their own policy rule, such as the rule that a particular device can only be accessed by a specific app, as well as rules based on the context. Furthermore, users can also use *save state* policies (policies related to system peripherals) to control access of apps to the Bluetooth interface (i.e., they can use policies to stop toggling of Bluetooth).

Whenever Android receives a Bluetooth socket-connection request from an app, the manager component of MASF asks the *device-checker* to check whether the app is associated in the bonding policy to the device it is trying to talk to: if the app is not on the device's bonding policy, the request is denied; otherwise, it is allowed to proceed. In this way, MASF can mitigate data stealing attacks. The framework can also provide policies for unpairing, to dissolve a pairing relation between the smartphone and a device, so user can stop the unauthorized unpairing attempts to defeat the data injection attack, because this attack is contingent on resetting the link key for the phone-device communication, so it cannot work without unpairing the phone from the device.

Referring to Figure 4.8, $AdapterService$ controls socket establishment and manages the unpairing operation. MASF inserts a Reference Monitor into the $AdapterService$ through the SIL, in order to control the bonding (pairing) between apps and Bluetooth devices. First, any request by an app to pair/unpair to a Bluetooth device is delivered to the SML manager (arrow 1), which might perform several tests including the ones directly related to the communicating app/device pair and the parameters of the communication. For the latter, the manager calls upon the device-checker (arrow 2) to receive a direct input from the reference monitor via the System Interface Layer (arrow 3) and then report the obtained information about the app/device pair and the communication parameters to the manager (arrow 4). As per all other checks, the manager refers to the policy-database to read the policies related to external-device connections (arrow 5), and then to make a decision on the current connection, and to inform the action-performer of the decision (arrow 6), in order to enforce the required actions (arrow 8).

4.4 Chapter Summary

In this chapter, the detailed design of the proposed MHealth Apps Security Framework (MASF) is presented. MASF is meant to be a practical and lightweight framework to secure sensitive medical data that is handling by mHealth apps. The framework consists of two major layers: the Security Module Layer and the System Interface Layer. The main contribution of this thesis lie in the SM layer, which comprises several components to fulfil the framework functionality. This chapter explains how the core component of SML, which is called the *manager*, relies on a set of special *checker* tools and a repository of policies to make decision on the various requests and actions made by mHealth apps to access system resources. With special monitors for the context, installation, malware, external devices, as well as a special taint analyser, the *manager* can acquire a fair idea on the events on the smartphone, and can delegate the enforcement of security decision to a dedicated component called the *action-performer*. SML cannot function at all without a *window* into the internals of the underlying Android system, which is provided by the SI layer. The final part of the chapter discussed four different scenarios in which MASF can operate to further clarify the expected operation of the framework.

CHAPTER 5: IMPLEMENTATION AND EVALUATION

This chapter explains the process of validating and evaluating the proposed framework. It commences with a brief overview of the implementation of MASF in Section 5.1, where the implementations of different components of the framework are briefly explained for the purpose of validating the design presented on the previous chapter.

MASF achieves all its security and privacy objectives with a minimal trade-off between security and performance. The implemented solution induces a small overhead both in terms of time and energy consumption. Further, all security and privacy checks and extensions are incorporated in the Android system with minimal changes to the codebase of the Android stock version, and to the user interface of the existing security architecture. The framework is also backward compatible with the current security mechanism for better acceptability in both the Android ecosystem and its healthcare community. Section 5.2 explains the process and results of evaluation of this framework in terms of effectiveness and performance overhead (for example induced computational time and energy consumption), compared to the stock version of an Android system. Section 5.3 discusses the overall performance of MASF, while Section 5.4 provides a summary of the chapter.

5.1 Implementation Details

This section presents the technical details of MASF's implementation. As discussed earlier in Chapter 4, the design of MASF encompasses two main layers, one of which provides the main functionality of security checks and in turn consists of several components and is written in Java; the second layer is providing the necessary integration with the Android platform, and as such, entails modifications to the Android OS. This section starts with the latter layer, the System Interface Layer (SIL), then proceeds to present the Security Module Layer (SML).

5.1.1 Implementation of the System Interface Layer

Starting from the bottom of MASF architecture, this layer is essential for integrating the functionality of the whole framework, mainly provided by the upper SM layer, into the underlying Android platform internals. SML is implemented as a set of user-space modules; therefore, it must be provided with some interface to access the Android system. This interface should include adequate function calls and event triggers to enable the functionality of MASF. Providing this kind of Application Programming Interface (API) is the purpose of SIL.

Because the stock version of Android does not allow that kind of *hooking* into its internal workings, SIL must modify the Android stock code to fulfil its intended purpose. To this end, there are two options for SIL: either to be built from scratch, or to be adopted from any available API system that does provide the required hooks into the Android OS. Following the first option entails a significant amount of work on an essential part of MASF but nevertheless not the main focus of the framework, which is the set of security checks against mHealth apps, not how to integrate them into Android specifically.

The problem with the second option is that the needed API should be specialized on the one hand, providing hooks into specific Android internals that relate to its security model in particular, and should be generic on the other hand, allowing other modules to implement their own logic; i.e. it should not enforce a specific security model or policy on the calling modules. Fortunately, such an API system do exist, and one particular generic framework is selected to provide the basic interfacing needs of MASF to the Android platform. This solution is adjusted for the purpose of MASF and wrapped in the bottom layer, SIL.

SIL implementation relies on Android Security Framework (ASF) (Backes, Bugiel, et al., 2014). ASF is an extensible and model-agnostic security infrastructure. The basic idea behind ASF is to extend Android with a new security API that allows to write and integrate general security policies. This function perfectly fits the need of MASF. In that way, SIL allows for the development of security extensions against a well-defined security API in the form of security modules that integrate nicely in the Android's platform security. Those modules are located in the upper SM layer.

Technically, ASF provides necessary hooks into different layers of Android: the applicatios, the middleware as well as the kernel. Those hooks take the form of reference monitors that are planted at various points in the Android security architecture and are being called each time a protected resource in invoked. For each reference monitor, there is an associated API function that should be implemented by some security module written as a normal Android app. The API function is provided by ASF (and hence by SIL in MASF), while the security modules reside in the SM layer of MASF and are implemented in this thesis to enforce security-related policy on direct ICC, broadcast intents, and other channels through system content providers and system services.

ASF infrastructure has been prototypically implemented for Android v4.3 and is available in source code. It currently comprises 4606 lines of code. As mentioned earlier, SIL uses the implementation of ASF, which occurs at three different levels, each of which are briefly presented below.

5.1.1.1 Kernel space

To provide access to Android kernel and enable enforcing policies at that level, SIL adopts the mature Linux Security Module (LSM) (Wright, Cowan, Smalley, Morris, & Kroah-Hartman, 2002) framework. LSM itself implements mandatory access control at the kernel level and enables other modules to register for enforcement hooks to kernel components such as process management and the virtual file system. A submodule of SIL is no more than a standard LSM module that registers for LSM hooks using the LSM API. Then, this submodule provides its own *kernel API* to other modules in the upper layer of MASF (i.e. Security Modules Layer), to implement mandatory access control at the kernel level through the SIL API, which is ultimately using the LSM API. Several other known solutions depend on LSM, including SELinux (Smalley, Vance, & Salamon, 2001), and TOMOYO (Harada, Horie, & Tanaka, 2004).

5.1.1.2 Middleware layer

Android architecture contains a set of system services (e.g. activity manager service, location manager service and network manager service) and system apps (e.g. Dial app, Calendar app and Camera app) that offer the Android API available to developers of other apps, including mHealth apps. SIL extends this middleware of the Android security model using a large number of reference monitors embedded inside Android system services and system apps. These reference monitors act as hooks that are linked to any module in the upper SM layer through SIL *middleware API*, in contrast to SIL *kernel API* mentioned in the previous subsection. SML modules are expected to implement access control and security policy decisions in the functions of this middleware API, through which the hooks would enforce them upon the control flow within Android middleware.

SIL middleware API contains 168 functions (full listing of these functions is provided in Appendix B). These functions define the bulk of functionality available to the upper components of MASF. This API can be broken down into several categories, the most important of which is the category of *enforcement functions*, which is composed of 136 methods and are called by SIL whenever the enforcement hooks in system apps and services are triggered. Each hook has a corresponding function in the API that implements the policy decision logic for this hook. Enforcement functions receive the same parameters as their hooks, and they can modify the program flow at run time by passing arguments by reference or returning objects as return values.

Another category of middleware API functions provided by SIL is *life-cycle* management. SML modules implement functions to manage the system life-cycle, such as initialization or shutdown. For example, modules can use these functions to initiate their policy engines or to save internal states to persistent storage before the device turns off. Furthermore, SIL middleware API include event notification interfaces used to propagate important system events to SML modules. For instance, modules should be immediately informed when an app was successfully installed, replaced, or removed. SML modules can also use callback interfaces for communicating in a more direct manner with system services, such as the *PackageManagerService*, and avoids the need to go through the Android API. For example, this category of SIL API includes functions that allow modules to efficiently resolve PIDs to application package names.

SIL also provides a special *callModule* function that allows SML modules to implement communication with front-end apps (e.g. the *user-interactor* module in SML that enables users to provide custom security policies). When using *callModule()*, this communication is based on *Bundles*, which are key-value mappings used to send arbitrary data from one activity to another by way of intents.

5.1.1.3 Application layer

At the application layer, SIL allows modules of SML to hook inside mHealth apps themselves, using the technique of *Inlined Reference Monitors (IRM)*, introduced by Erlingsson and Schneider (Erlingsson & Schneider, 2000). This mechanism rewrites an mHealth app such that the reference monitor is embedded right into the app itself. The security module at SML can use an instrumentation API provided by SIL to hook any

Java function within a selected app, then function calls are redirected to the embedded inlined reference monitor, which in turn enforces policy decisions made by the module.

Hooks injected via the instrumentation API are local to the app process that the API is called from. The main advantage of policy enforcement in the caller's process context is that the hook and subsequently the security module has full access to the internal state of the app and can thus provide rich contextual information about the caller. In contrast to the hooks placed in the Android middleware, application layer hooks are dynamic, which means that hooks are injected by directly modifying the target app's memory when a new app process is started.

5.1.2 Implementation of the Security Modules Layer

The detailed design of the Security Modules Layer (SML) is presented in Chapter 2. As explained in that chapter, SML consists of a set of modules including the *manager*, the *policy-database*, the *user-interactor*, the *action-performer* and a set of security *checkers*. This section describes the structure of such *security modules* within SML. Each of these modules is no more than a user-space Android app, created using the Android SDK. What differentiates most of these modules from normal Android apps is the use of the security API provided by SIL. Modules within SML are installed in a protected location on the file system, and are loaded during the system boot. Similar to any Android app, the package of an SML module is a *jar* file that contains program code in terms of Java classes, resources, and the *manifest* file (Figure 5.1).

The *manifest* file is an XML-based file that declares the main properties of a module, such as the module author or code version, and, more importantly, the name of the main Java class that forms the entry point for the module. As in regular apps, the *classes.dex* file contains the Java code compiled to Dalvik executable bytecode (DEX). It contains all Java classes that implement the security module's logic. During the load process of the security module, the SIL middleware uses the Java reflection API to load the module's main class (as specified in the manifest file) from *classes.dex*. To ensure that the reflection works error-free, the main class must implement the SIL API discussed in the previous subsections.

Security modules also contain a native library *liblsm.so* written in C, and a corresponding Java class *LSM.java*, which exposes the native library via the Java Native Interface. The purpose of this code is to implement a proprietary interface between the user-space processes of security modules that uses SIL middleware API and those that uses SIL kernel API, which are implemented as Linux security module in the kernel (Section 5.1.1.1). *LSM.java* has to implement the generic interface for the communication with the kernel. The generic kernel module interface of SIL loads *LSM.java* through the Java reflection API into Android's application framework. This allows apps and services to communicate via SIL (and reflectively through *LSM.java*) with the kernel module and avoids a policy-specific interface.

Finally, each module can ship with proprietary resources, such as initial configuration files or required binaries. During module initialization, Android informs the module about the location of its Jar file, enabling the module to extract these resources on-demand from its file.

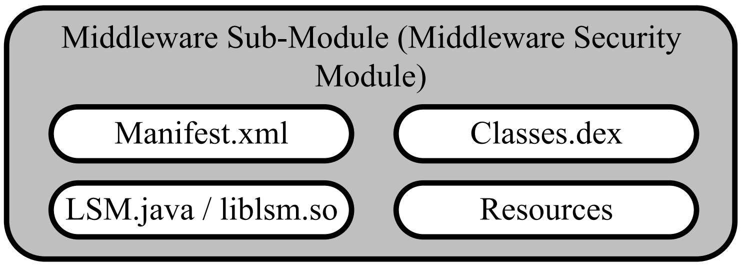


Figure 5.1: Middleware Security Module Structure

SML modules aim to enforce access control policies as explained in the previous chapter, the implementation of these access control and policy enforcement mechanisms are presented in the following sub sections.

5.1.3 Enforcement of Fine-Grained Access Control Policies

Enforcing fine-grained access control policies on apps is implemented by restricting third party apps' ability to access critical system resources. To realize this goal, SML modules mostly use two mechanisms provided by SIL: the middleware-level enforcement functions, and the application-level IRM. In general, the *manager* component would implement several reference monitors or *hooks* embedded within Android's system services and apps to intercept important events and also respond with decided actions. Each reference monitor protects one specific privileged resource and is placed such that it is always invoked by the control flow between the Android API and access to the resource. In total, all monitors enable powerful and semantically rich security policies. For example, fine-grained filtering of requests for data from *content providers* can be achieved using the pre-query hooks on Android *ContentProvider* system service to modify selection arguments and retrieve only contacts that are allowed for the current caller specified via a policy entry.

Using the IRM instrumentation API, SML modules dynamically hook selected Java functions within the mHealth app process. Function calls are redirected to an inlined reference monitor that enforces fine-grained policy decisions made by the module, which are difficult to enforce using the native Android security model. For example, functions that setup the registration with sensors can be hooked to enforce low data resolutions, and functions that access the web can be enforced to use encryption through *https* rather than the use of *http*.

In SML, most policy decision logic is implemented by the *manager* referring to policies in the *policy-database* persistent storage. The *manager* responds to reference monitors at various levels in the Android architecture, selects the apps into which IRMs are injected, and when necessary makes use of other *checkers* to make security logic decisions. The response of the *manager* includes taking a decision about the requested access or intended action and then invoking the *action-performer* to execute one of pre-determined set of actions that have been defined in Section 4.2.1.5.

5.1.3.1 Context-aware access control

A pertinent example of fine-grained access control is context-aware access control. A context is based on the geolocation of the device as well as the time. Depending on this context, MASF either allows or denies apps access to security and privacy sensitive information. To accomplish that, the *manager* invokes the *context-checker* component upon the start of an mHealth app. The *context-checker* in its turn registers as a listener for location updates to detect context changes. When the app requests a resource or service, the *manager* checks if there is any policy that is associated with the app request. If such a policy exists, the *manager* asks the *context-checker* to report the context of the device in terms of location and current time. The *manager* then matches the reported context with the context defined in the policy. In case of a match, the *manager* enforces the corresponding policy restrictions by invoking the *action-performer* to apply those restrictions on the app request.

5.1.3.2 One-time checks

As an example of the implementation of other security checks, event functions provided by SIL are used to trigger malware checks and taint analysis upon the event of installing a new app. These checks are necessary only once per installation of an app or its updates, and the event of installation is registered with SIL by the *manager*. There is no reason for waiting until the app actually runs and makes access requests to check for malware behaviour or information leakage. To guard against these threats, the *manager* is notified whenever a new app is installed, upon which it asks the *malware-checker* to exam the app against malware behaviour. In the current implementation, this is achieved by simply invoking an installed anti-malware app, specifying the app as the target for the scan process. The results of the scan are delivered back to the *manager.* In addition, the *taint-analyser* is also invoked by the *manager* to perform taint analysis on the newly installed app. As explained earlier in Chapter 4, the implementation of the *taint-analyser* is based on FlowDroid (Arzt et al., 2014) and extended by (He, 2014). Taint analysis could be very efficient in revealing special threats such as the *privilege escalation attack*.

5.1.3.3 Permission management

In Android, all the resources that require explicit permissions to access are protected by the *ActivityManagerService* class through the *checkComponentPermission* method, which is called to verify that calling app has the right permission(s) to access a resource. Among the hooked system services, the SIL layer of MASF provides 10 hooks for the *ActivityManagerService.* The SML *manager* implements the enforcement functions corresponding to the *checkComponentPermission* hook, and thereby intercepting the permission checks before they are performed by the Android system and then enforcing more fine-grained control permissions that better reflect the app function and narrow down its accessibility to system resources. For example, the *READ_PHONE_STATE* permission provides apps with access to a large set of information on the phone number, the IMEI/MEID identifier, subscriber identification, phone state (busy/available), SIM serial number…etc., while only a subset of this information might be adequate.

5.1.3.4 Data shadowing

Data shadowing means that an app that wants to retrieve sensitive information (e.g., contacts information, location data, or IMEI number) only gets empty, fake, or filtered data. This is one of the possible actions that could be enforced by the *action-performer* component based on the *manager* decision, though its implementation might not be obvious like other straightforward actions such as blocking or revoking permissions. Data shadowing is implemented using *edit automaton* hooks in the *ContectProvider.Transport* class, the *ContactsProvider*-specific hooks, the *Telephony* service and *Location* service. For *ContentProvider* and *ContactsProvider*, the SIL pre-query and post-query hooks allow a fine-grained filtering or replacing (faking) of the returned data as well as returning an empty data set. However, the current coverage of enforcement hooks does not include some of the data shadowing points, such as microphone, logs, or camera, and are left for the future work.

5.1.3.5 Installation checks

MASF also extends Android's app installation process with policy-based checks and denies the installation of a new app when it violates the relevant policies. The manager component makes use of a special module called the installation-checker. This module performs a set of investigations based on the permissions requested by an app and the interfaces (e.g., Broadcast receivers) it wants to register in the system. The result of that examination is then returned to the manager and the installation of the app is rejected if the relevant policy dictates so. To implement this security service, the manager uses the scanPackage hook in the PackageManagerService, checks the new app against the policy and aborts its installation in case the policy rejects the app.

5.1.3.6 Intent management

MASF can enforce a number of controls/restrictions on different activities by intercepting intents for purposes such as launching an app (to prevent certain apps from running on the device), app installation and uninstallation (to prevent an app from sending an intent to install or uninstall an app), services (to prevent apps from starting background services), locking and unlocking the device (to prevent requesting pin code to unlock the device), broadcasts (to prevent apps from broadcasting intents), or app multitasking (to prevent running multiple user-apps simultaneously). The implementation of intent interception is achieved by the *manager* using SIL-provided enforcement functions. In particular, the *manager* implements the function associated with the hook of the intent broadcasting subsystem of the ActivityManagerService.

5.1.3.7 Managing system peripheral state

MASF also enables users of configuring policies to restrict access to peripheral devices such as the Bluetooth, based, for example, on a certain context. Users can configure their devices to prevent apps from modifying the state of a device (whether the device is enabled or disabled). The *manager* uses enforcement functions into *BluetoothAdapter* and *WifiManager* classes to implement the control over peripherals state rather than permission management. In that way, MASF prevent apps from crashing if they lack the necessary code to handle exceptions that may result from just revoking permissions.

5.2 Experimental Evaluation

This section presents the results of evaluating MASF from two distinct perspectives: effectiveness and practicality. Through a series of experiments, the aim of the evaluation is to verify that MASF can actually protect private information, and can control the access to sensitive data in accordance with users' policies. Users are expected to be able to (re)adjust and enforce security and privacy policies after installation even at runtime without affecting app functionality. This section presents the experimental results of various checks that control the privacy and security of user in the context of mHealth apps. Further, the impact of MASF on user experience is also evaluated and discussed. A number of mHealth apps are tested to check the effectiveness and performance impact of the framework, most of which are explained in Section 5.2.1.

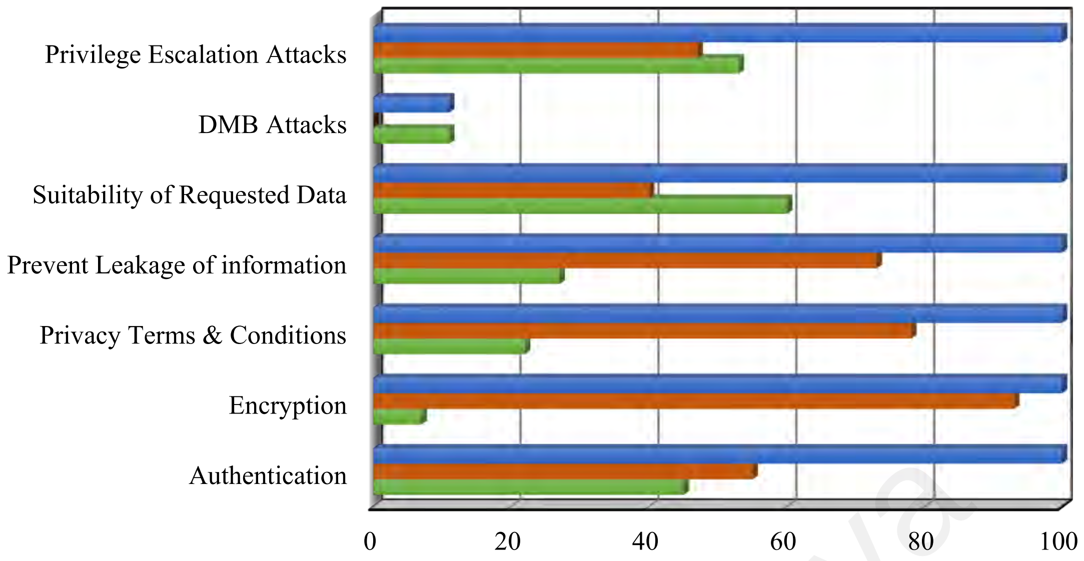
In addition to evaluate the implementation MASF to understand its effectiveness in protecting sensitive data, the conducted experiments also aimed to evaluate the framework's performance impact on the phone's normal operations. All the experiments were conducted on a Samsung Galaxy S3 device with a Quad-core 1.4 GHz Cortex-A9 and 1GB memory, running the Android 4.3 operating system (API level v. 18). At the time of evaluation, the top 100 apps from the Google Play market were run for testing and evaluating the modifications to the stock Android.

A number of experiments were performed to evaluate the effectiveness and efficiency of MASF against different attacks and in various scenarios. The results of these experiments are presented in the following subsections.

5.2.1 Experiment 1: Effectiveness

This section presents a number of experiments that were conducted to show how effectively can MASF protect user data against security threats. A sample of 100 mHealth apps was used to check how the framework works against leakage of private information, and what steps it takes to prevent this leakage. As a first step, the sample apps were executed on a stock Android version to establish an insight into the scene of mHealth apps security. The complete list of the sample apps is given in Appendix A.

Figure 5.2 presents the results of examining the set of apps, showing a number of threats faced by mHealth apps on the stock Android operating system. During the experiment, it was observed that a lot of apps are disclosing users' data (73 out of 100) without the knowledge of users. Only 11 apps out of 100 were accessing the external devices. Those apps were checked against DMB attacks (for both data injection and data stealing variations of the DMB attacks), with no single app being able to defend against these attacks. Furthermore, 53 apps had the potential to perform privileges escalation attacks. Only 7 apps used encryption to secure user data. Some apps accessed user information that was not necessary at all to fulfil their functionalities. Only 45 percent of the apps employed authentication to secure user information.



	Authentication	Encryption	Privacy Terms & Conditions	Prevent Leakage of information	Suitability of Requested Data	DMB Attacks	Privilege Escalation Attacks
🟦 Total	100	100	100	100	100	11	100
🟧 No	55	93	78	73	40	0	47
🟩 Yes	45	7	22	27	60	11	53

🟦 Total 🟧 No 🟩 Yes

Figure 5.2: Security Assessment of a 100 mHealth Apps on Stock Android OS

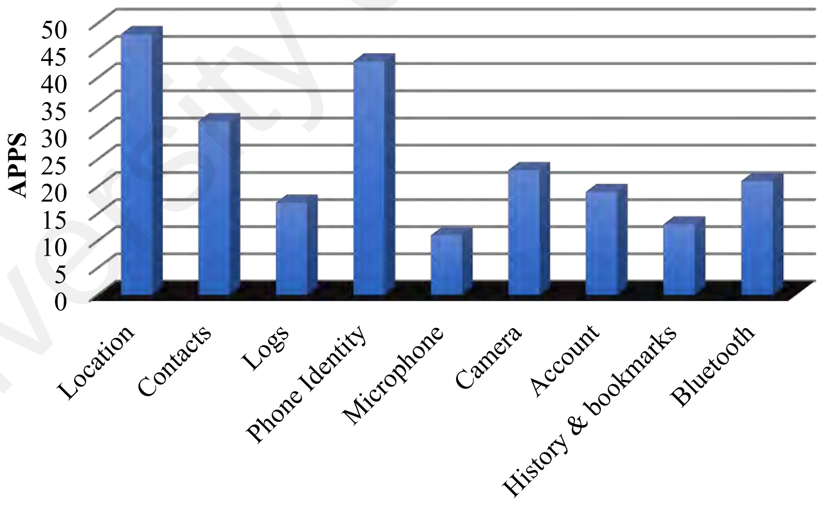


Figure 5.3: Number of Apps Attempting to Access Various Resources Containing Sensitive Data

Most of the examined apps attempted to access the device location, list of contacts, call logs, phone identity, camera, account information and Bluetooth, as can be seen in Figure 5.3. Figure 5.3 shows how many out of the 100 target apps accessed these resources and how frequently, which also reflects the relative importance of the resources. Among 73 apps that found to be leaking information surreptitiously, 48 apps leaked location information, 43 apps sent the IMEI number, and 24 apps leaked both the IMEI number and location information. Although most of them do not need to do, 32 apps did access the contacts information, 21 apps had permission to access Bluetooth, which can cause DMB attacks in the case of mHealth apps. Other apps attempted to access the logs, microphone and other sensitive resources, which leads to disclosure of sensitive user data.

In later experiments, the same set of 100 mHealth apps were installed on the device containing Android 4.3 enhanced with MASF, which provides a set of privacy and security policies to effectively prevent leakage of private information, DMB attacks and privilege escalation attacks. MASF can provide apps with shadow data, and deny permissions to access Bluetooth by untrusted apps. MASF indeed prevented all the 73 apps from leaking information that was detected in the previous experiment. MASF also successfully defended against DMB and privilege escalation attacks. MASF solved the encryption problem by providing shadow or empty data so that apps cannot expose actual sensitive data in plain text.

A number of experiments was performed to test the effectiveness of MASF against different attacks, and experimental results show that the framework is very effective to control unauthorized access of user data and other system services and resources. Experimental results of three scenarios are presented below.

Scenario 1. In this example, an app named *Heart Rate Monitor* was used, which was known to request the permission to access the location, even though this app does not need to access the location to perform its operations. When this app was installed and run on the stock Android, after granted the permission to access location, the app leaked the phone location information, and sent that information to a remote server. However, when MASF was installed on the smartphone, and then the same app was installed and run, the app provided empty/fake location to the remote server. That was the effect of checking the rules and policies related to privacy of location by the framework. Furthermore, the experiment showed that no app was affected by enabling MASF on the smartphone; they were running smoothly and working normally.

Scenario 2. Another two experiments were performed on the app named *Diabetes*, one without, and the other with, the protection of MASF. At the time of installation, this app declared the permission to access the phone identity, but this information was leaked to a remote server. In the first experiment, this app sent the IMEI number of the smartphone to a remote server. The leaked IMEI number (376855633798032) was in the query string of HTTP GET request to the server. After the confirmation of leakage on stock Android, the second experiment started on Android enhanced with MASF. The same app was run again to check the effectiveness of MASF. Although the app was again sending an IMEI number, but this time the IMEI number was not the original one, the framework sent a fake IMEI number. The reason here to send the fake value instead of blocking the access is to make sure that the app continues its operation without crashing.

Scenario 3. To check the effectiveness of MASF against DMB attacks, several attack scenarios are attempted against a set of apps as described in Naveed et al. (2014), including DMB data-injection and data-stealing attacks. All these attack attempts were thwarted. In particular, for all data-stealing attacks, MASF stopped the malicious app from making socket connections to the target device, as these connections violated the policies. MASF also did not allow access to system peripherals (i.e. Bluetooth devices) for apps that were not explicitly allowed by the phone user through the policies. For the data-injection attacks, MASF blocked all the attempts to unpair the phone from the devices and therefore defeated the data-injection attacks.

5.2.2 Experiment 2: Malware Test Suite

This set of experiments adopts the malware test suite that was presented by (Bugiel, Davi, Dmitrienko, Fischer, et al., 2011). MASF was evaluated by applying that malware test suite, which constitutes a set of recent privilege escalation attacks (Davi et al., 2011; Enck, Ongtang, & McDaniel, 2008; Lineberry, Richardson, & Wyatt, 2010; Schlegel et al., 2011). The test suite exploits transitive permission usage to perform attacks against user privacy or to gain unauthorized access to protected system interfaces. A group of 6 example attacks were chosen for the experiment, which are defined in Table 5.1. Each row in the table shows the granted set of permissions for two colluding malicious apps, in a given scenario attack. Attack scenarios 2-4 are proof-of-concept examples of malware, while scenarios 1, 5 and 6 emulate the attacks in (Davi et al., 2011; Enck et al., 2008; Lineberry et al., 2010; Schlegel et al., 2011).

Table 5.1: Malware Test Suite

#	1st App	2nd App
1	Malicious voice recorder RECORD_AUDIO and PHONE_STATE or PROCESS_OUTGOING_CALLS	Malicious wallpaper (Schlegel et al., 2011) INTERNET
2	Malicious step counter ACCESS_FINE_LOCATION	Malicious wallpaper INTERNET
3	Malicious contacts manager READ_CONTACTS	Malicious wallpaper INTERNET
4	Malicious SMS widget READ_SMS	Malicious wallpaper INTERNET
5	Malicious app no INTERTNET	Vulnerable browser (Lineberry et al., 2010) INTERNET
6	Malicious app no SEND_SMS	Vulnerable SMS widget (Davi et al., 2011) SEND_SMS

Attacks 1 to 4 involve two colluding malicious apps, where one app gets the Internet access, and another one can gain access to sensitive user data, such as contact, user location, SMS database and recorded audio. In the attack scenario 1, the malicious voice recorder also requires the PROCESS_OUTGOING_CALLS or PHONE_STATE permission, because this permission is required to be notified at what time the incoming or outgoing call starts. Apps collude to send private user information to the remote adversary. In attack scenarios 2 to 4, apps establish the ICC communication link between them, whereas in scenario 1 they communicate through a covert channel.

In scenarios 5 and 6 a malicious app misuses a vulnerable app that has the permission to access Internet, voice call or SMS services to get unauthorized access to these system interfaces. Scenario 5 emulates attacks reported in (Egele, Kruegel, Kirda, & Vigna, 2011; Lineberry et al., 2010), which exploits an unprotected interface of the Android web browser to do unauthorized download of malicious files. In scenario 6 the malicious app sends unauthorized text messages, similar to the attack shown in (Davi et al., 2011).

MASF is tested to evaluate its effectiveness in detecting the malware presented in the above description. All tests were performed on the device running Android 4.3 and MASF. After enforcing MASF access control, the malicious apps were installed from the test suite and performed the corresponding attacks. All attacks were successfully detected and prevented by MASF.

5.2.3 Experiment 3: Impact of Permission Restrictions

This experiment was conducted to find the impact of permission-related policy restrictions on apps. In particular, the experiment aimed to check whether an app crashes or not when a permission that was granted at the time of installation is denied. A stress test was performed on each app, and the impact of revoking the permissions of each app was observed when it was requesting for a service or resource. The experiment was also performed on a set of 100 apps and used the ADB logging utility to view the permission being revoked when the *checkComponentPermission* hook is invoked.

Figure 5.4 shows both the number of apps that crashed and those that did not, upon performing the test on each permission. An app was considered as crashed if it failed during the execution of any part of its functionality, whether major or minor. The primary cause of crashes of an app is the lack of developer's skills to handle the denial of previously granted permissions. App crashes can be prevented or reduced if error-handling was added whenever an app tries to access resources or request for a service.

5.2.4 Experiment 4: Impact of Data Shadowing

The aim of this experiment was to observe the impact of data shadowing or blocking access to user data. In data shadowing, when an app tries to access sensitive information, MASF returns a fake copy of data. For example, when an app tries to access device geographical location, MASF sends fake location information if the defined policies restrict such access. The data shadowing effect was tested on the same sample of 100 apps used in the previous experiments. The result of this experiment is shown in Figure 5.5. The effect of blocking access to user data was also checked in this experiment.

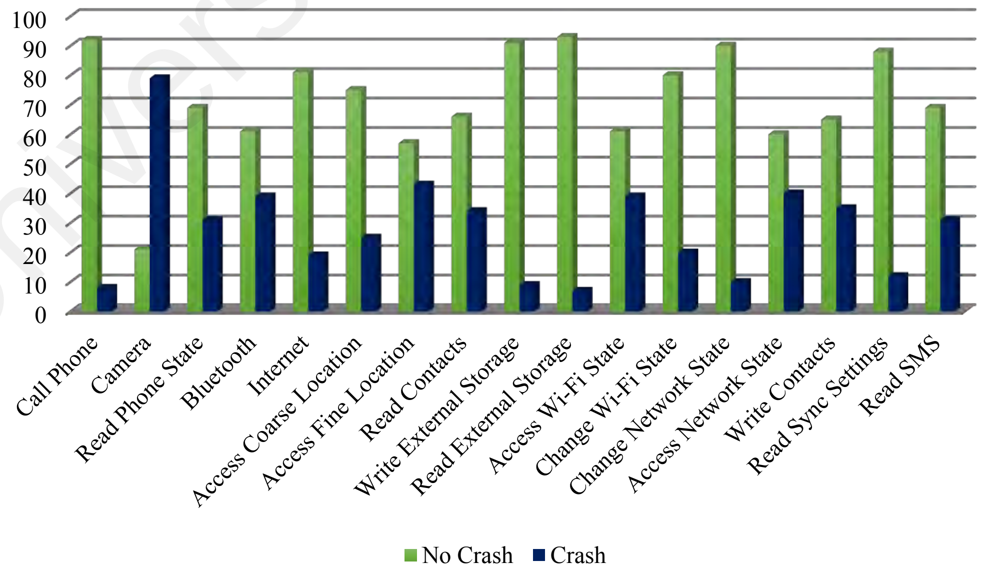


Figure 5.4: Impact of Permission Revoking on Applications

The experiment results show that data shadowing is very effective and it can successfully protect the user information without giving access to original data; as can be seen in Figure 5.5, only 9 out of the 100 apps were crashed during the experiment (as previously, an app is counted as crashed even if it is failing to perform a minor functionality). Actually, data shadowing works very successfully where revoking permissions does not work, because sometimes the user has to give the permission for particular data, service or resource in order to make sure that apps behave normally, so revoking permission does not work all the time. MASF can also deny access to sensitive data when a particular app attempts to access user data. Hence, the result of blocking access to sensitive data is also demonstrated in Figure 5.5. Blocking access to sensitive data causes crashes for more apps; as evident in Figure 5.5, 23 out of 100 apps crashed during the experiment when using the blocking access option. Thus, data shadowing is more effective and robust compared to data blocking. However, both options are successfully protecting the user sensitive data.

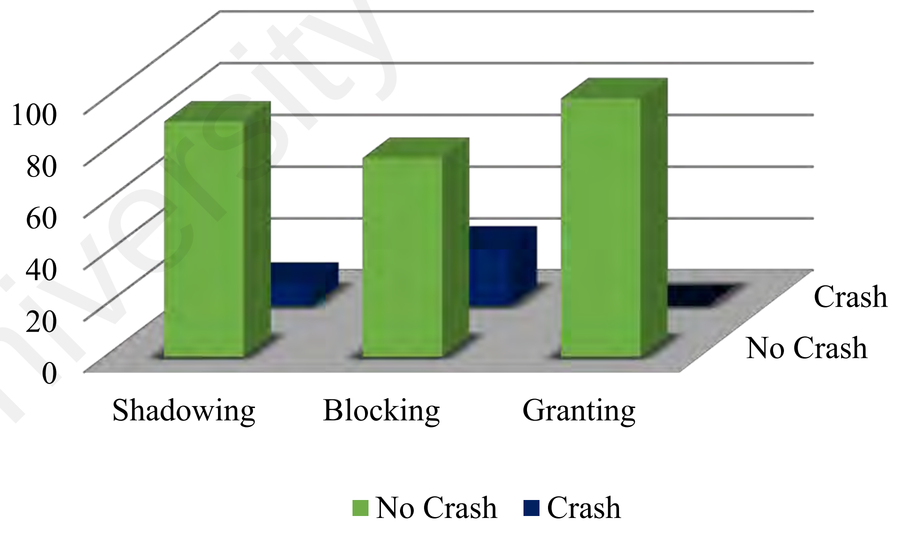


Figure 5.5: Impact of Data Shadowing on mHealth Apps

5.2.5 Experiment 5: Impact of Disabling/Blocking Intents

This experiment was conducted to observe the impact of disabling the intents that can be helpful for an adversary to steal user data or can cause to reveal sensitive user data. This experiment was also conducted on the same sample of 100 apps. This experiment tested how intent disabling can affect apps functionality. The results of this experiment are shown in Figure 5.6.

Disabling intents is effective against certain types of attacks (e.g., to mitigate privilege escalation attacks and data leakage). As can be seen in Figure 5.6, only 17 out of the 100 apps crashed during the experiment. Intent disabling is working very fine with 83 out of 100 apps, and it does not affect the app’s main functionality. Disabling some intents may affect the functionality that is related to the disabled intent; however, this is done intentionally to protect sensitive data, resources and services. Generally, disabling Intent is very useful to protect the sensitive information.

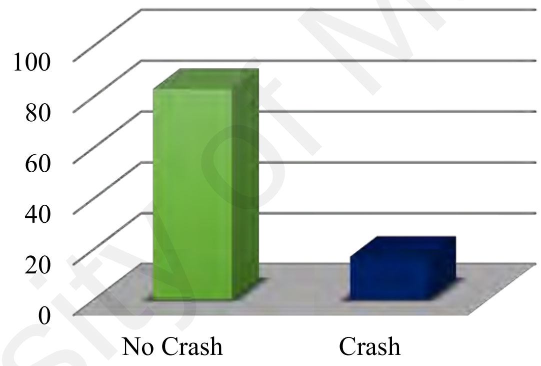


Figure 5.6: Impact of Intent Disabling on Apps

5.2.6 Experiment 6: Impact of Enabling/Disabling System Peripherals

The main objective of this experiment is to find the impact of denying the access to system peripherals (e.g., Bluetooth). In this experiment, the behaviour of apps was checked when apps try to access the Bluetooth, and MASF policy restrictions do not allow that. Similar to previous experiments, this experiment tested how many apps crash or cannot achieve their main objective when restricting access to peripherals. The experiment was again performed on the sample of 100 apps; however, only 43 out of the 100 apps were accessing the system peripherals (i.e., Bluetooth). The results of this experiment are illustrated in Figure 5.7.

The results of the experiment show that only 11 out of 43 apps crashed during the experiment. If denying the access to system peripherals affect the main functionality of the app, then the case was considered as a crash. The experiment proved that MASF can successfully revoke access to system peripherals in order to protect sensitive user data.

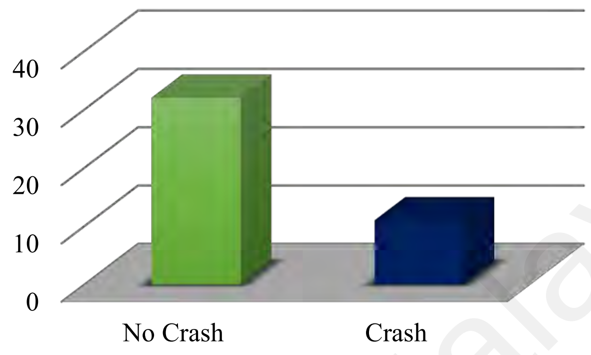


Figure 5.7: Impact of Enabling/Disabling System Peripherals

5.2.7 Experiment 7: ICC False Positives

Although MASF was shown to be effective in enforcing security restrictions, it is possible that some of the framework decisions may be too restrictive and can be considered as *false positive*s. For example, denying an access request or revoking a permission when there is no real reason to do that is a wrong decision (false positive). To evaluate this possibility and study how MASF affects third party apps in this regard, the sample of 100 apps was employed again. During the experiment, all apps were installed and thoroughly used in a test set in an arbitrary order, with interleaving installation, uninstallation, and usage of the apps. To quantify the evaluation, the collected measurements focused on the Inter-Component Communication (ICC) issued and denied requests, since ICC is the primary mechanism of communications between apps and access control on ICC is important for the enforcement of security and privacy policies in the middleware.

The apps were tested and their behaviour was checked with and without MASF. Figure 5.8 shows that MASF performed a number of policy checks during the test of 100 apps, and in response to these checks, some ICC attempts were denied during the experiment that are also shown in the figure. The number of denied ICC is very little as compared to a static system like Kirin (Enck et al., 2009). If MASF policies restrictions would be enforced with Kirin, then each of 100 test apps would in average conflict with 54 other apps from the set.

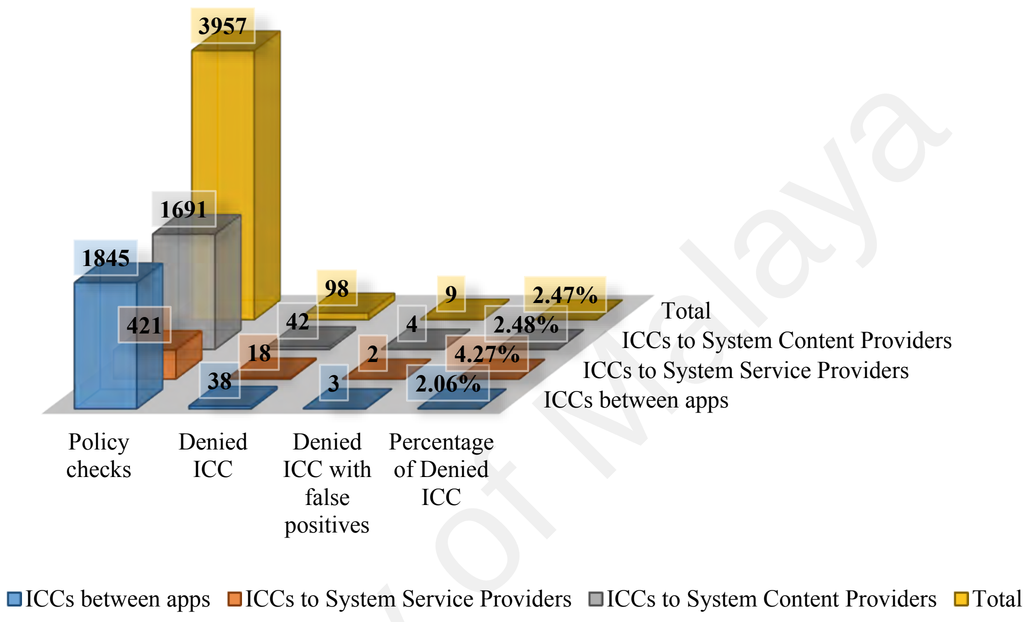


Figure 5.8: Denied ICCs by Different Policy Checks

Manual inspection of each message along with the network packet trace confirmed that there were very few false positives. The denied ICCs were evaluated, which revealed that very few of the denied ICCs were false positives, as shown in the Figure 5.8. In particular, the following were the main sources for false positives: (1) for direct ICC, 5 apps were the main source for the false positives, because they held a high number of permissions. Those apps caused 3 false positives out of the 38 denied ICCs; (2) the power system service provider, caused only 2 false positives out of the 18 denied ICCs with system service providers; and (3) the system settings content provider, caused just 4 false positives out of the 42 denied ICCs in this case. False positives for apps can be prevented by adjusting and refining the policies.

5.2.8 Analysis of the Impact on Android Security

This section presents results of some experiments and observations that were conducted to analyse the impact of MASF on Android security itself. This section also presents a security analysis of possible threats from malicious users or apps that can bypass the framework's policy restrictions.

First of all, it is noted that MASF does not reduce the Android OS security itself. For each requested access to an app or system service, MASF only introduces further checks, and these checks depend on the security and privacy policies. However, each access that is not denied by MASF is still passed on to the Android Permission Check system and not influenced by MASF anymore. As a result, MASF can only reduce the number of accesses allowed, not reducing the security.

In Android platform, each app is assigned a unique UID that the system uses to refer to an app. However, if two apps are created and signed by the same developer, then the system gives both apps the same UID, so these apps can share the same processes if needed (Bugiel, Davi, et al., 2012). The Android OS enforces its security policies not based on the app label or its package name, but rather on the process UID. MASF, on the other hand, obtains the name of the package (app) which is performing an action by calling the *PackageManager*'s *getPackagesForUid (int uid)*. In this way, MASF policy restrictions are not dependent on the UID but are transformed to refer to the package name. For instance, if two apps, app X and app Y, are sharing the same UID, and the user has blocked the access to GPS from app X, this app may still be able to get information about the device's geographical location because App Y was still able to access GPS according to the policies. MASF prevents such threat through blocking all package names associated with a UID using the *getPackagesForUid* method.

In MASF, user can define security policies to limit the access to resources in some necessary situations. For example, the user can define a policy for a particular set of apps allowing to use Bluetooth only at home. As mentioned earlier, users can configure policy restrictions based on the context (i.e., time and location), and these policy restrictions are either enforced system-wide or per app. If apps can modify these policies, then any malicious app can execute specific attacks based on policy configurations. In order to protect the policies, MASF does not allow write privileges to be granted on policies directly for any other app, thus preventing policies from being modified. All writes are performed by MASF through the *user-interactor* component.

Further, a malicious app that is aware of MASF policies may attempt to drop a policy or modify the device's identified context so that the wrong policy is applied. Nevertheless, in MASF implementation, the context information is directly retrieved from the system protected APIs that cannot be modified by apps. Context information is managed by the *context-checker* that collects such information regardless of which apps are running on the device or which services are requested by apps, based on requests from the *manager*. This independency of the *context-checker* gives robustness in gathering context data that is forwarded to the *manager*.

Once an app requests access to a resource, the Android OS verifies whether this app has permission(s) for the requested resource only at the time of the request. However, some processes associated with certain resources may continuously run even if the device is later moved to a different context for which the user has not allowed access to this resource. The reason behind this is that permission granting is not checked continuously while the process is running, instead it is only checked once the request is issued. Malicious apps may take advantage of this, e.g., by continuously recording audio in one context while transitioning to another context. The Android OS does not continuously verify whether an app has audio recording permission during recording. It verifies the permission each time a request is made, after that when the permission is approved the app can continue using the permission for that particular session. MASF implementation thwarts this type of attacks. When a registered location is associated with a policy restriction on audio or video access, MASF forces the apps with the associated permissions in their *AndroidManifest.xml* to close.

Finally, it is important to make sure that an adversary cannot skip MASF enforcement. As mentioned, MASF is designed as an extension of Android platform, and it is deployed to a protected location on the file system, from where it is loaded during boot. To avoid the adversary modifying the operating system of the phone itself (drivers and MASF included), *Trusted Computing* mechanisms leveraging *Trusted Platform Module (TPM)* can be used. However, the discussion of these mechanisms is outside the scope of this thesis.

5.3 Performance Evaluation

This section presents the experimental results conducted to evaluate the performance of MASF. Particularly, time and energy consumption are evaluated by running different features of MASF because energy consumption and time performance are two main limitations of smartphone. All performance evaluation experiments were conducted on Android v4.3. As mentioned earlier, the custom system image of Android that includes MASF was installed on this smartphone. Details of the experiments and how MASF induces overhead are given in the following subsections.

5.3.1 SML Performance Overhead

5.3.1.1 Performance overhead of permission checks

First, the overhead induced by MASF permission check system is examined. This is the induced overhead caused by each request that leads MASF to check the permissions of an app. Experiments were conducted to find the amount of overhead induced by MASF permission check with respect to both time and energy consumption. To find time overhead, an experiment is conducted to measure the time induced by MASF's checks. MASF permission check works through a hook before the Android permission check mechanism is taking control. The time interval is measured between the request of a resource (app or system service) and the moment that request is fulfilled and MASF is finished with the permission checking. MASF hooks into both access requests by apps and system services, but not much differences were noticed between these two cases. In fact, both requests are treated in the same way by MASF. Further, for some resources, MASF does not influence the results with any overhead.

The results of this experiment are shown in Figure 5.9. In the graph, Y-axis shows the time overhead measured in milliseconds, and X-axis shows the number of rules examined in response to permission checks. This graph is plotted by obtaining the average of 200 measurements. As evident from the figure, the time overhead induced by MASF for its permission checks is negligible, with all measured delays even corresponding to larger number of rules being under 0.5 ms. As expected, the time overhead increases as the number of active policies are increasing.

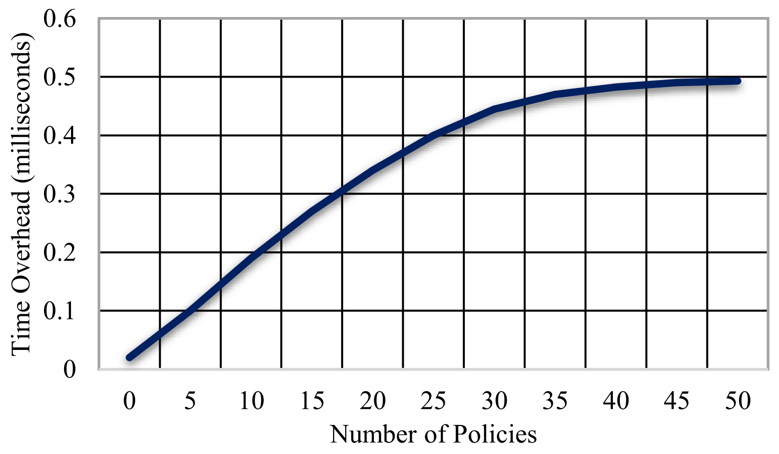


Figure 5.9: Time Overhead of MASF Permission Check System

As mentioned earlier, energy consumption is an important issue in smartphones, so energy overhead induced by MASF permission check system was also investigated to ensure that this framework is a practical solution. To investigate energy overhead, all MASF functions and policies were enabled during the experiment to estimate the maximum energy demand of MASF. The experiment was started with fully charged battery, and then a number of different functions was performed on the smartphone every ten minutes during the experiment, such as sending of SMSs, making phone calls, running different medical and non-medical apps. The experiment was repeated 30 times for each of two cases when the smartphone was running Android as well as when it was running Android with MASF.

The results of the experiment are shown in Figure 5.10. Android with MASF consume higher energy as compared to the original Android. This is due to the energy consumption of different checkers in the MASF. In particular, 4571 permission checks were called during the experiment for 24 different resources. Further, as expected, consumed energy by MASF increases as the number of active rules increases. As shown in Figure 5.9, the energy consumption of MASF is almost 5 % of the battery when 15 rules are active, whereas it turns into almost 9% of the battery when 45 rules are active. It can also be seen from the figure that the energy consumption of MASF does not increase linearly with the increase in number of active rules. This is because some basic actions of MASF remain the same for a number of rules.

In conclusion, according to the experiment results, the energy consumption of the MASF permission check system is reasonable. However, most probably it would be very rare cases in which the framework would need 45 active rules or more, which makes MASF a quite reasonable solution because very few rules would be active for a particular event and for a specified period of time. It should also be noted that this solution, to the best of the author knowledge, is the first of its nature, and hence no particular attention is paid towards any possible optimizations.

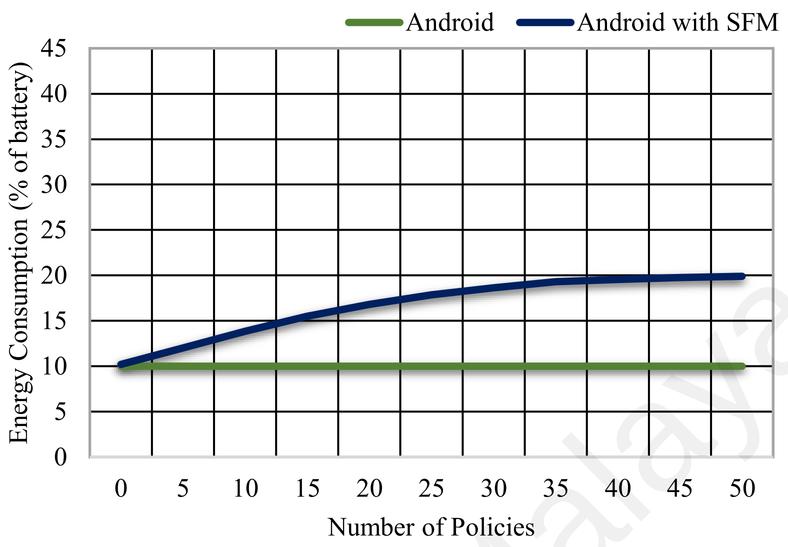


Figure 5.10: Energy Overhead of MASF Permission Check System

5.3.1.2 Performance overhead of context checks

The purpose of this experiment is to measure the Android device's energy consumption change when MASF context-related policies are enforced compared to when they are not. For this purpose, the device's battery percentage was monitored when running both the stock Android and Android with MASF, separately. In each case, it was ensured that the device's screen never turns off and that Wi-Fi and GPS are both enabled.

Finding the context through context-checker, the overhead depends mostly on how much the system is desired to be responsive to context changes. Actually, if system wants to detect context changes sooner, then the context checking frequency will be higher, hence, it would lead to increasing the overhead. To study the effect of various update frequencies, the performed experiment was run over three different frequencies of location-data updates: 5, 10, and 15 minutes, in addition to the case where no context checks were involved. Further, the experiment was started with a fully charged battery.

According to the measurement, if the *context-checker* registers for context information updates every 5 minutes, then it consumes 15% of the battery. However, if the period of checking the context extends to every 10 minutes, then it consumes only 7% of the battery. If the context update period further extends to every 15 minutes, then it consumes just 3.25% of the battery energy. This can be seen in Figure 5.11 by noticing the battery percentage displayed on the device with and without enforcing MASF policies.

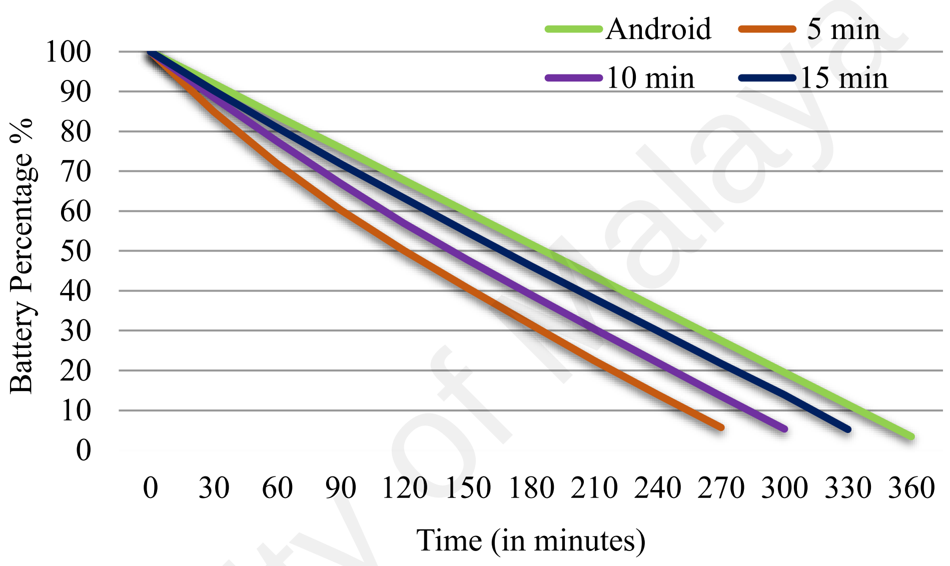


Figure 5.11: Comparison of Device Battery Consumption while Checking for Context Updates

In the first case, the overhead is not negligible, while in the third case, the energy consumption for checking every 15 minutes is quite promising. Moreover, optimizations are possible to the current implementation, for example, the context checking frequency might be reduced while the value of a variable of interest (e.g., location) is far from the current value of the context. The experiment results also demonstrate the importance of the issue of energy consumption in the context of smartphones. In conclusion, MASF has a reasonable amount of context-check overhead in terms of energy consumption.

5.3.1.3 Java microbenchmark

The Java CaffeineMark 3.0 benchmark tests were employed to measure the overall performance overhead of MASF. This benchmark contains a set of programs to evaluate the app runtime overhead. For evaluating MASF, an Android port of the standard CaffeineMark 3.0 [Pendragon Software Corporation 1997] was used.

The CaffeineMark 3.0 is a series of tests that measure the speed of Java programs running in various hardware and software configurations. CaffeineMark scores roughly correlate with the number of Java instructions executed per second, and do not depend significantly on the amount of memory in the system or on the speed of a computer's disk drives or Internet connection. CaffeineMark uses several tests to measure various aspects of Java virtual machine performance. Each test runs for approximately the same length of time period. The score for each test is proportional to the number of times the test was executed divided by the time taken to execute the test.

The following is a brief description of each test, originally presented in (Pendragon Software Corporation, 1997):

- *Sieve*: the classic sieve of Eratosthenes finds prime numbers.
- *Loop*: the loop test uses sorting and sequence generation as to measure compiler optimization of loops.
- *Logic*: tests the speed with which the virtual machine executes decision-making instructions.
- *Method*: the Method test executes recursive function calls to see how well the VM handles method calls.
- *Float*: simulates a 3D rotation of objects around a point.

The overall CaffeineMark score is the geometric mean of the individual scores.

The CaffeineMark 3.0 benchmark was run on both stock Android and the Android with MASF. The results for both experiments are shown in Figure 5.12. This figure shows the score for all the tests, where a higher score is better in terms of performance. During the experiment, the *Sieve* score was 9830 for stock Android and 9339 for Android with MASF. CaffeineMark gave *Loop* scores of 21833 and 19651 for stock Android and Android with MASF respectively. *Logic* test obtained 18880 for stock Android while 17370 for MASF. Stock Android acquired a score of 15184 for the *String* test while MASF Android attained a score of 13362. *Float* score is 9617 for stock Android and 9040 for Android with MASF. Further, MASF achieved a score of 7009 for the *Method* test as compared to 7875 for the same test with stock Android. The unmodified Android system had an average score of 12924, while the average score of MASF was 11799, which shows that MASF has an 8.7% overhead as compared to the stock Android according to the CaffeineMark benchmark tests.

In conclusion, the results of performance overhead indicate that MASF is a lightweight framework that is capable enough to effectively protect user private and sensitive medical information.

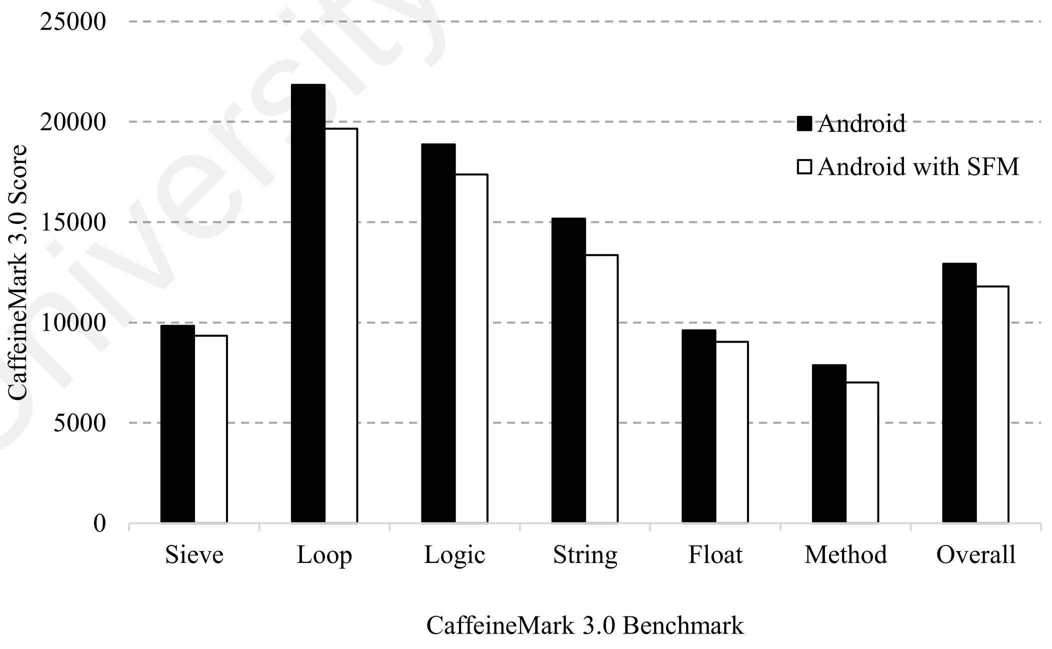


Figure 5.12: The Result of CaffeineMark 3.0 Benchmark / Microbenchmark of Java Overhead.

5.3.1.4 Macrobenchmarks

To measure the overhead of MASF on a higher-level, several macrobenchmarks were conducted for high-level smartphone operations. The experiments were performed on the device running Android with and without MASF. Each experiment was run at least 50 times. Average results with 95% confidence interval are shown in Table 5.2. During the study, only limited performance overhead was observed as detailed below.

Table 5.2: Macrobenchmark Results of Time Overhead for Modified Core Android Methods

Method / Benchmark	Android	MASF	Overhead
App Load Time	109 ms	113 ms	3.66 %
Check Component Permission	101 ms	108 ms	6.93 %
App Filter	140 ms	142 ms	1.42 %
Intents to Start Activity	116 ms	125 ms	7.75 %
Network Access	90 ms	91 ms	1.1 %
Policy Change/Alteration	-	2 ms	-
Intent to Start Service	73 ms	76 ms	3.94 %
Intent to Send Broadcast	69 ms	71 ms	2.89 %
Phone Call	159 ms	170 ms	6.9 %
User Data Content Resolver	107 ms	115 ms	7.47 %
Device Peripherals Set Enable/Disable	83 ms	88 ms	6.02 %

To evaluate the timing overhead introduced by MASF modifications to Android, the amount of time that takes MASF modified methods to fully execute was calculated during the experiment. The execution times of these modified methods before the modifications were also calculated (i.e., the time of execution of these methods on stock Android). The two sets of times were compared to estimate the overhead of MASF modifications. Specifically, this experiment measured the overhead time caused by "*intercepting app permissions*", "*intent messages to start activity, service or send broadcast*", "*app load time*", "*network access check*", "*user data accesses*" (e.g., contacts), "*phone call*", "*policy change*", and "*access to system peripherals*" (e.g., Bluetooth). Detailed descriptions of these parameters are given in next paragraph. Table 5.2 reports in milliseconds the time imposed on these methods. As the results show, the overall delay introduced by enforcing MASF policies is not perceivable by the end-user.

- In "*app load time*", the time to start an app in MASF was compared to a baseline app load time in Android. The app load time measures starting from when *ActivityManager* receives a command in order to start an activity component to the time the activity thread is displayed. This time includes app resolution by the *ActivityManager*, *IPC*, and graphical display. The average overhead for loading apps is 4 ms, which is negligible. It means MASF adds only 3.66% overhead, as the operation is dominated by native graphics libraries.
- In "*check component permission*", the time overheads for both Android and MASF were measured to find how much additional time MASF consumes to check component permission for the enforcement of policies. In this case, MASF only takes 7 ms more as compared to stock Android, which shows that MASF adds 6.93% overhead.
- In "*app filtering*", MASF filters the potential target apps when Android uses an implicit intent to start an activity component. The time between sending an intent message and the resolution of the final list of apps presented to the user was measured during the experiment. MASF only causes a negligible delay of 2 ms, which means it introduces a mere 1.42% overhead.
- The time delay caused by "*intents to start activity*" was also measure during the experiment. On stock Android, it takes 116 ms, however, for MASF it takes 125 ms. Hence, MASF adds 9 ms, which translates to 7.75% overhead.
- In "*network access check*", a hook is placed in the kernel by MASF, which is called every time a process attempts to access the network. This experiment was conducted by using an app which attempts to access the network repeatedly. Since Android already performs similar check to enforce its INTERNET

permission, MASF's additional checks have negligible impact (i.e., it causes a negligible delay of 1 ms, which entails only 1.1% overhead).

- In "*policy change*", the time to change the policies and contexts, and the time for reassignment of services and resource to all apps is also measured during the experiment. This policy re-enforcement only takes 2 ms.
- The time delay caused by "*intent to start service*" was also measured during the experiment. On stock Android it takes 73 ms, however, for MASF it takes 76 ms. Hence, MASF brings 3 ms, which means it adds only 3.94% overhead.
- The time delay for "*intent to send broadcast*" was also measured, which takes 69 ms for stock Android and 71 ms for MASF. It causes a negligible delay of 2 ms, which means MASF adds only 2.89% overhead.
- The "*phone call*" benchmark measured the time from pressing "dial" to the point at which the audio hardware was reconfigured to "in call" mode. MASF adds less than 11 ms per phone call setup, which means 6.9% overhead, which is significantly less than call setup in the network that takes time on the order of seconds.
- The time overhead for "*user data content resolver*" was 107 ms and 115 ms for stock Android and MASF respectively. This means a delay of 8 ms, which translates to a 7.47% overhead.
- MASF also needs to enable or disable some of the system peripherals to enforce policies, the time delay to "*enable or disable a device peripheral*" was further measured during the experiment. The result shows that it causes only a delay of 5 ms, which means MASF adds a negligible 6.02% overhead.

5.3.1.5 System memory overhead

An experiment was conducted to measure the amount of memory overhead placed by MASF on the system. Mainly, the purpose of this experiment was to observe the changes in memory usage caused by MASF restrictions and by the context detection mechanism (i.e., *LocationService*) that continuously run in the background for context updates.

Figure 5.13 depicts that the memory usage when enforcing MASF policies closely matches the memory usage when these policies are not enforced. As the figure shows, memory usage due to permission checks in Android is 9.25% of the memory while the same is 10.36% of the memory during the experiment on Android with MASF policy enforcement. Further, usage ratio due to intents is 6.22% of the memory in case of Android whereas it is 8.11% of the memory in the case of MASF. Likewise, user data cause a memory usage of 18.53% and 21.37% in the cases of Android and Android with MASF, respectively. Finally, system peripherals are taking 9.93% of memory space during the operation on stock Android and 12.43% of memory when using MASF.

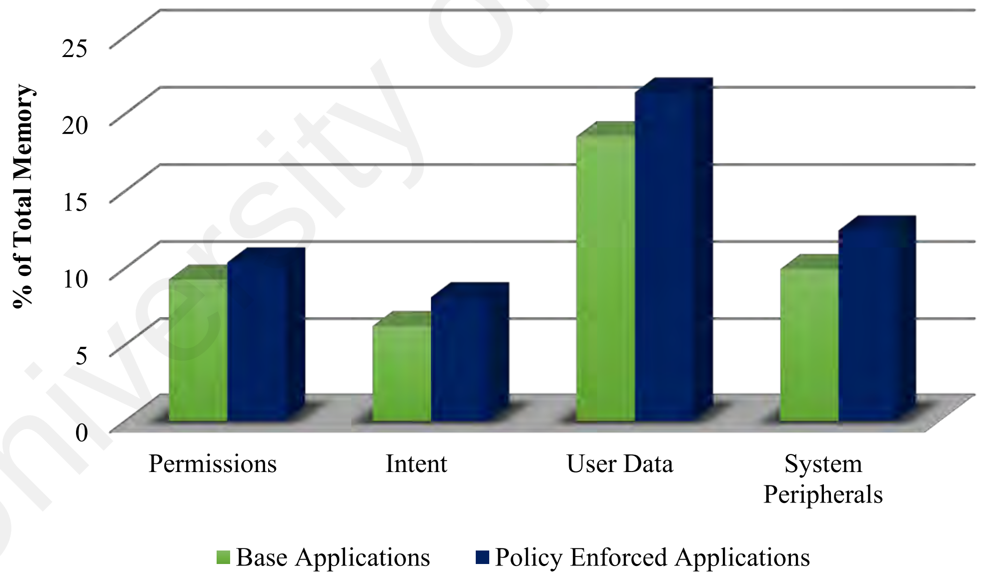


Figure 5.13: Total Memory Overhead Comparison with and without MASF Policy Restrictions

The maximum memory overhead caused by MASF during these tests was about 2.84% of the memory, thus it is within an acceptable range. The average memory footprint of MASF is 1.5 MB with a standard deviation of 389.4 KB.

5.3.2 SIL performance overhead

Most of the actual performance overhead comes from the Security Model layer in MASF; nonetheless, it is also interesting to check the impact of SIL on the system performance. SIL is based on LSM at the kernel level, and the performance of the latter was already evaluated separately, e.g., for SE Android (Smalley & Craig, 2013). The focus here is to check the effect of SIL middleware security framework on the performance of instrumented middleware system services and apps.

SIL is implemented as a modification to the Android OS code base in version 4.3_r3.1 (“Jelly Bean”) and used the Android Linux kernel in branch android-omap-tuna-3.0-jb-mr1.1. Microbenchmarks were performed for all execution paths on which a hook diverts the control flow to SIL’s middleware framework. First, the execution time of each hooked function was measured without loading SML models and allowing by default all access. Afterwards this test was repeated with hooks disabled to measure the default performance of the same functions and thus operating like a stock Android. All those microbenchmarks were performed on Android version 4.3, which was booted and then used according to a test plan for different daily tasks such as sending SMS and emails, browsing the Internet, contacts management, and installing and uninstalling third party apps.

Table 5.3 reports the number of measurements for each test case and their mean values. To exclude extreme outliers, the highest decile in both measurement series was excluded from the measurements. For SIL, the mean is the weighted mean value with consideration of the frequency of each single hook. During the experiment, the mean time of stock Android was 106.556 μs, whereas SIL without loading SML models imposed 118.924 μs, which is approximately only 11.61% overhead compared to stock Android. Figure 5.14 presents the relative cumulative frequency distribution of the measurements series and further illustrates this low performance overhead.

Table 5.3: Weighted Average Performance Overhead of Executing Hooked Functions in Stock Android and in SIL. The Margin of Error is given for the 95% Confidence Interval.

Type (System)	Frequency	Mean (µs)
Stock Android	6743	106.556
SIL	5535	118.924

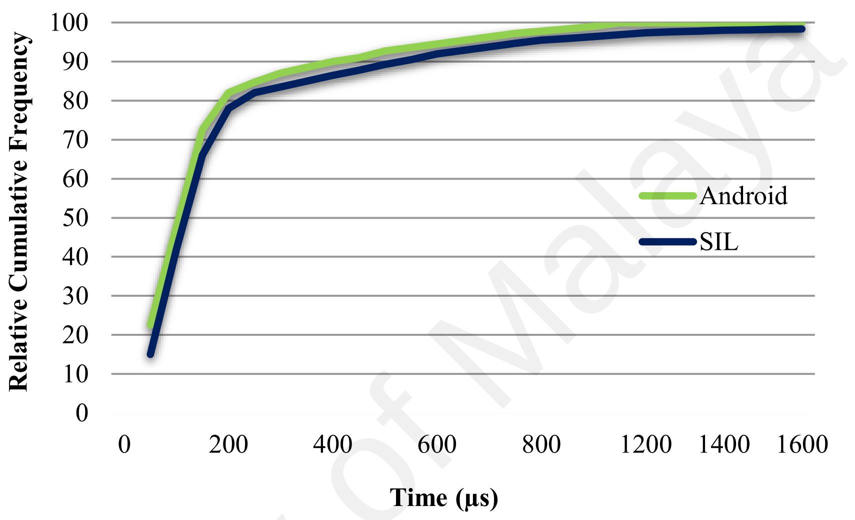


Figure 5.14: Relative Cumulative Frequency Distribution of Microbenchmarks in Stock Android vs SIL

5.4 Chapter Summary

This chapter presents the technical details of the implementation of the proposed framework, MASF. The implementations of the two main layers, SML and SIL, along with their components are discussed. The resulting prototype is meant to validate the design of the framework in previous chapters via a proof-of-concept implementation. This chapter also describes the evaluation of MASF through a number of experiments that show its effectiveness in thwarting attacks on mHealth apps, and preventing intentional as well as unintentional disclosure of user privacy. Furthermore, experimental results of performance evaluation showed that MASF' overhead is not human perceivable, suggesting that this framework is a practical and lightweight solution viable for deployment in smartphone platforms.

CHAPTER 6: CONCLUSIONS AND FUTURE WORK

This chapter concludes the presentation of the work on developing a security framework for mHealth apps on the Android platform. Besides summarizing the main points of the research work, this chapter re-evaluates the research objectives and puts forward the accomplishments of this study. Furthermore, it summarizes contributions of the research and discusses its limitations. Finally, it presents a few potential future directions to improve this work.

The complete organization of this chapter is as follows. Section 6.1 discusses the reappraisal of the objectives of this research work. The contributions of this research are highlighted in Section 6.2. Moreover, Section 6.3 discusses the limitations of this research work, while Section 6.4 proposes several directions for future work.

6.1 Research Summary and Objectives Achievement

Smartphones and their apps have dramatically changed the way of communication, computation, and the model of many traditional and new services. One active area of smartphone apps that has witnessed an astonishing growth is the mHealth apps. mHealth apps are defined in this thesis as software programs that provide health related services through smartphones and tablets. Using mHealth apps in the delivery of healthcare is rapidly proliferating. mHealth apps have several potentials that drive this popularity, including the ability to increase patient satisfaction, improve doctor efficiency, reduce the cost of healthcare, and to improve the availability, affordability and effectiveness of healthcare services.

The problem of this thesis starts with the observation that despite all those benefits, mHealth apps bring about new risks to the security of user’s sensitive medical data. Existing smartphone operating systems, particularly Android, are not sufficient to ensure privacy and security of users' data, particularly in the case of mHealth apps. The lack of that privacy protection is one of the major barriers to the widespread use of mHealth apps. This observation leads to the need for better solutions to secure mHealth apps, and ensure the confidentiality, integrity and availability of medical data, and consequently facilitate the adoption of these apps by the healthcare system.

Motivated by the above argument, this thesis aimed to improve the security of medical data associated with Android mHealth apps, as well as to protect the privacy of users from threats that might be imposed by those apps. Because the required functions to achieve such a goal cannot be implemented using a single app that is running in the user space, there is a need to develop a complete framework of multiple components that roots deep in the internals of the system and effectively protects the target data. The overall goal of this research was to design and implement such a framework that is both practical and effective, directed specifically to the Android platform. To achieve this goal, this research had set five objectives, listed in Section 1.4. These objectives are revisited and commented in the following paragraphs.

To identify the outstanding issues related to the security and privacy of mHealth apps, and find out research gaps in this area, the literature was comprehensively reviewed as a first objective of this research. This review also covered the Android security architecture and the corresponding issues in the Android platform. A thorough investigation and analysis were performed on the field of mHealth apps, producing a thematic taxonomy of research works on mHealth apps (Section 2.1). This taxonomy classified the research field on mHealth apps into four main classes (with further sub-classes). Then, a brief introduction of Android operating system and its security mechanisms was presented (Section 2.2). Subsequently, the existing solutions in the literature to protect Android platform were discussed, and specific security weaknesses of the Android platform were explained, mainly reporting on privacy threats exposed by works in the literature. After that, a comprehensive survey on security and privacy threats to mHealth apps was provided (Section 2.3). An analysis on a set of mHealth apps was performed to further investigate the security issues of mHealth apps. Hence, the challenges mHealth apps are facing were discussed thoroughly. Finally, existing security solutions that are specifically designed for mHealth apps were examined, highlighting their weaknesses.

As the main objective of the thesis, a MHealth Apps Security Framework (MASF), was proposed to secure the execution of mHealth apps and their users' data (Section 4.1). MASF addresses various security and privacy threats of mHealth apps, such as data leakage, DMB attacks, privileges escalation attacks, and misuse of permissions. This framework provides mechanisms for fine-grained access control, context-aware access control, and protection of private information through taint analysis and data shadowing. Moreover, MASF provides the users the ability to define their own policies according to their requirements to control apps' access to system resources based on several criteria, including the current context of the device such as the time and location. The framework can revoke certain permissions, revoke access for system peripherals, disable intents, and provide shadow data.

The proposed design was then implemented in a real environment to meet the objective of evaluating the framework. According to the design of MASF, its implementation consists of two main parts: the Security Module Layer (SML) and the System Interface Layer (SIL). SML performs all the necessary security checks and tests and defines all security policies to be enforced on the apps in runtime, while SIL provides SML with the necessary interface into Android internals. As such, SML is the main part, and user can only interact with this layer. Users can define their own policies and store them within SML databases. SML comprises several components and works on top of the SIL. As a result of the implementation, a stock Android version was extended with several hooks and new components to implement MASF.

The proposed framework was evaluated and analysed in terms of effectiveness and efficiency. Effectiveness was shown by demonstrating that the framework can successfully protect the system from a particular set of attacks, while efficiency was evaluated by examining the performance overhead in terms of energy consumption, memory and CPU utilization. Experiments conducted to evaluate efficiency included testing MASF against different attacks and malware, as well as testing the impact of data shadowing, permission restrictions, disabling intents, and the impact of enabling/disabling system peripherals. The results of those experiments showed that MASF is very effective against all the listed attacks, and it successfully protected the mHealth apps and their users' data in the tested scenarios. Subsequently, false positive and usability tests were also performed, which showed a very small number of false positives.

Another set of experiments tested the performance of MASF in terms of processing time, energy consumption and memory. These experiments demonstrated clearly that MASF induce negligible overhead in the process of deploying all the checks and enforcing all the policies. Hence, MASF proved itself as a lightweight and a practical solution. From the above summary, it can be concluded that all the objectives of this research have been successfully achieved.

6.2 Contribution of the Research

As discussed in the previous section, it is possible to refer to the targeted objectives and list the successful output in meeting them to derive the list of contributions. For a one-to-one mapping, please refer to Table 1.1 in Chapter 1. The aim of this section, however, is to highlight the contributions of the thesis in a more perceptive manner, to help appreciate the real value of the work.

(i) This work provides a comprehensive survey on mHealth apps and on their security issues. It summarizes some serious security and privacy issues associated with the use of Android mHealth apps, including DMB attacks, side channel threats, and usage of third party storage and services without encryption. Considering the relative recency of the topic, such survey has its own value to research community.

(ii) This work assessed the security of mHealth apps by testing a sample of top 100 free mHealth apps to find state-of-the-art security threats, which revealed many threats to Android mHealth apps, such as lack of authentication, authorization, confidentiality, and the leak of sensitive medical information by many apps.

(iii) This thesis also specifies the basic requirements that are needed to secure users' data in mHealth apps and to propose security framework on Android platform. As an emerging field with no or little previous guidelines, this thesis contributes a set of basic parameters needed to secure information in mHealth apps.

(iv) This thesis proposes a security framework to protect mHealth apps' user data. To the best of the author's knowledge, this is the first comprehensive framework for mHealth apps. A policy framework has been proposed previously in the literature (Mitchell et al., 2013), which only provides some *guidelines* to mitigate mHealth apps threats. This thesis, on the other hand, proposes a practical framework that could be installed and work on Android platform. This framework comprises two software layers, a Security Module Layer (SML), and a System Interface Layer (SIL). These are to be installed on Android OS to protect user information against security and privacy threats.

(v) The proposed framework is demonstrated and evaluated using proof-of-concept implementation on real Android devices. For evaluation, the effectiveness and performance overhead of the framework were examined. For effectiveness, a sample of mHealth apps is checked against information leakage, confidentiality and other attacks that are stated in section 1.3. For performance overhead, and due to lack of similar framework implementations, the framework was benchmarked with the stock version of Android 4.3, in terms of memory, processing and battery consumption. The unmodified Android system was considered a baseline against which to compare the performance impact of MASF.

6.3 Research Limitations

The scope of this research is limited to investigating and analysing the security and privacy threats of *mHealth apps* specifically, and then to propose a security framework to protect that class of apps. Although the analysis and evaluation did not explicitly include other classes of smartphone apps (e.g. banking, educational, and communication apps), there is no reason that the proposed solution cannot be applied to other apps.

This solution was also implemented on Android platform exclusively, and no other mobile platforms was considered for the implementation of the framework. Porting the solution to other platform is feasible in theory, though the amount of changes may be significant in practice due to the intimate relation between the design of the framework and the internals of Android operating system. Nevertheless, Android is by far the most widespread used smartphone platform, and choosing it for the development of the target framework seems a reasonable decision.

This research also suffers from the rapid evolution of the technology in the field of smartphone computing. Google is working hard to enhance the user experience as well as the security of its Android platform with each new release of the operating system. By the time this research has finished, there will probably be few enhancements to the platform that were not considered by the produced solution. This is a common issue in information technology-related long-term research. However, the value of the research is not diminished because of such advancements, and the produced solutions can usually be deployed in newer versions with no or little modifications.

Because of the nature of the addressed problem, the proposed solution had to go through a modification to the stock-based Android version, which might limit the wide acceptance of the solution. This limitation is common with all other similar solutions at the same level of effectiveness. In fact, the ability to access and modify the open source Android platform is a main motivation and enabler for such line of research, which increases innovation and contributes to the overall good of the technology. The ideal scenario for such proposals is to be adopted by Google in their main code base for next Android releases.

Finally, it might be worthy to note that during the design of MASF, the underlying Linux kernel and the Android middleware were considered as trusted base. Therefore, MASF does not attempt to prevent an adversary from compromising this base itself.

6.4 Future Work

This work is simply the first step in a longer journey towards realizing practical mHealth apps security. To increase the deployment opportunity of the proposed framework, further research and development work is needed, in part to cover for the current limitations, and also to introduce more features and improvements. Few suggestions for potential future work are listed below.

- One possible way to enhance the applicability of the proposed framework is to evaluate its performance against other classes of apps besides mHealth apps. For example, banking apps might require few special considerations to be taken, which might lead to the introduction of more security *checkers*, such as for key certificates and specific phishing attacks.
- It is also tempting to consider porting the framework to other smartphone platforms, e.g. Apple iOS.
- It is important to note that the successful deployment of the proposed framework would probably rely on the regular update of its implementation to cater for new developments introduced by Google the incremental releases of Android. For example, the introduction of dynamic run-time permissions might render few of the security checks performed by MASF redundant. The framework should also adopt newer capabilities of the underlying platform to provide better results and user-experience.
- The *context-checker* of MASF considers only the time and location as criteria for context-aware access control. It is possible to consider other types of context, such environmental variables, the presence of other devices and sensors, or a particular type of interaction between the smartphone and user.
- The current version of MASF uses static tainting to track sensitive data flows. Though the choice of static tainting has been justified in Chapter 3, there may be situation in which dynamic tainting can provide better results.

REFERENCES

Aafer, Y., Du, W., & Yin, H. (2013). *DroidAPIMiner: Mining API-level features for robust malware detection in android.* Paper presented at the International Conference on Security and Privacy in Communication Systems.

Abroms, L. C., Lee Westmaas, J., Bontemps-Jones, J., Ramani, R., & Mellerson, J. (2013). A content analysis of popular smartphone apps for smoking cessation. *American journal of preventive medicine, 45*(6), 732-736.

Adhikari, R., Richards, D., & Scott, K. (2014, 8th - 10th December 2014). *Security and Privacy Issues Related to the Use of Mobile Health Apps.* Paper presented at the Australasian Conference on Information Systems (ACIS), Auckland, New Zealand.

Aguinaga, S., & Poellabauer, C. (2013). Stealthy Health Sensing to Objectively Characterize Motor Movement Disorders. *Procedia Computer Science, 19*, 1182-1189.

Al-Hadithy, N., & Ghosh, S. (2013). Smartphones and the plastic surgeon. *Journal of Plastic, Reconstructive & Aesthetic Surgery, 66*(6), e155-e161.

Al-Haiqi, A., Ismail, M., & Nordin, R. (2014). A New Sensors-Based Covert Channel on Android. *The Scientific World Journal, 2014*(Article ID 969628), 1-14.

Al-Haiqi, A. M. A. (2015). *Sensors-based side channel security threats on android platform.* (Ph.D.), National University of Malaysia.

Albano, P., Castiglione, A., Cattaneo, G., & De Santis, A. (2011). *A novel anti-forensics technique for the android OS.* Paper presented at the Broadband and Wireless Computing, Communication and Applications (BWCCA), 2011 International Conference on.

Albrecht, U.-V., von Jan, U., Jungnickel, T., & Pramann, O. (2012). App-synopsis-standard reporting for medical apps. *Studies in health technology and informatics, 192*, 1154-1154.

Alexander, S., Hoy, H., Maskey, M., Conover, H., Gamble, J., & Fraley, A. (2013). Initiating Collaboration among Organ Transplant Professionals through Web Portals and Mobile Applications. *OJIN: The Online Journal of Issues in Nursing, 18*(2).

Android. (2016a). Android Ndk. Retrieved from https://developer.android.com/ndk/index.html

Android. (2016b). Security. Retrieved from https://source.android.com/security/

Android. (2016c). Sign Your App. Retrieved from https://developer.android.com/studio/publish/app-signing.html

Anokwa, Y., Ribeka, N., Parikh, T., Borriello, G., & Were, M. C. (2012). *Design of a phone-based clinical decision support system for resource-limited settings.* Paper presented at the Proceedings of the Fifth International Conference on Information and Communication Technologies and Development.

Apple. (2016a). Apple Store. Retrieved from http://store.apple.com/us

Apple. (2016b). iOS. Retrieved from https://www.apple.com/my/ios/

Arnhold, M., Quade, M., & Kirch, W. (2014). Mobile Applications for Diabetics: A Systematic Review and Expert-Based Usability Evaluation Considering the Special Requirements of Diabetes Patients Age 50 Years or Older. *Journal of medical Internet research, 16*(4), e104.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., & Rieck, K. (2014). *DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket.* Paper presented at the NDSS, San Diego, CA, USA.

Årsand, E., Frøisland, D. H., Skrøvseth, S. O., Chomutare, T., Tatara, N., Hartvigsen, G., & Tufano, J. T. (2012). Mobile health applications to assist patients with diabetes: lessons learned and design implications. *Journal of diabetes science and technology, 6*(5), 1197-1206.

Arzt, S., Huber, S., Rasthofer, S., & Bodden, E. (2014). *Denial-of-app attack: inhibiting the installation of android apps on stock phones.* Paper presented at the Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices.

Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., . . . McDaniel, P. (2014). Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM SIGPLAN Notices, 49*(6), 259-269.

Au, K. W. Y., Zhou, Y. F., Huang, Z., & Lie, D. (2012). *Pscout: analyzing the android permission specification.* Paper presented at the Proceedings of the 2012 ACM conference on Computer and communications security.

Aungst, T. D. (2013). Medical applications for pharmacists using mobile devices. *Annals of Pharmacotherapy, 47*(7-8), 1088-1095.

Avancha, S., Baxi, A., & Kotz, D. (2012). Privacy in mobile technology for personal healthcare. *ACM Computing Surveys (CSUR), 45*(1), 3.

Azar, K. M., Lesser, L. I., Laing, B. Y., Stephens, J., Aurora, M. S., Burke, L. E., & Palaniappan, L. P. (2013). Mobile applications for weight management: theory-based content analysis. *American journal of preventive medicine, 45*(5), 583-589.

Backes, M., Bugiel, S., Gerling, S., & von Styp-Rekowsky, P. (2014). *Android Security Framework: Extensible multi-layered access control on Android.* Paper presented at the Proceedings of the 30th Annual Computer Security Applications Conference.

Backes, M., Gerling, S., Hammer, C., Maffei, M., & von Styp-Rekowsky, P. (2013). AppGuard–enforcing user requirements on android apps *Tools and Algorithms for the Construction and Analysis of Systems* (pp. 543-548): Springer.

Backes, M., Gerling, S., Hammer, C., Maffei, M., & von Styp-Rekowsky, P. (2014). AppGuard–Fine-grained policy enforcement for untrusted Android applications *Data Privacy Management and Autonomous Spontaneous Security* (pp. 213-231): Springer.

Baheti, M. J., & Toshniwal, N. (2014). Orthodontic apps at fingertips. *Progress in Orthodontics, 15*(1), 1-5.

Bai, G., Gu, L., Feng, T., Guo, Y., & Chen, X. (2010). *Context-aware usage control for android.* Paper presented at the International Conference on Security and Privacy in Communication Systems, Berlin Heidelberg.

Barrera, D., Kayacik, H. G., van Oorschot, P. C., & Somayaji, A. (2010). *A methodology for empirical analysis of permission-based security models and its application to android.* Paper presented at the Proceedings of the 17th ACM conference on Computer and communications security.

Bartel, A., Klein, J., Monperrus, M., Allix, K., & Le Traon, Y. (2012). *Improving privacy on android smartphones through in-vivo bytecode instrumentation* (2879711118). Retrieved from

Becher, M., Freiling, F. C., Hoffmann, J., Holz, T., Uellenbeck, S., & Wolf, C. (2011). *Mobile security catching up? revealing the nuts and bolts of the security of mobile devices.* Paper presented at the 2011 IEEE Symposium on Security and Privacy.

Bellard, F. (2005). *QEMU, a Fast and Portable Dynamic Translator.* Paper presented at the USENIX Annual Technical Conference, FREENIX Track.

Bender, J. L., Yue, R. Y. K., To, M. J., Deacken, L., & Jadad, A. R. (2013). A lot of action, but not in the right direction: systematic review and content analysis of smartphone applications for the prevention, detection, and management of cancer. *Journal of medical Internet research, 15*(12).

Beresford, A. R., Rice, A., Skehin, N., & Sohan, R. (2011). *MockDroid: trading privacy for application functionality on smartphones.* Paper presented at the Proceedings of the 12th Workshop on Mobile Computing Systems and Applications.

Berthome, P., Fecherolle, T., Guilloteau, N., & Lalande, J.-F. (2012). *Repackaging android applications for auditing access to private data.* Paper presented at the Availability, Reliability and Security (ARES), 2012 Seventh International Conference on.

Bhansali, R., & Armstrong, J. (2012). Smartphone applications for pediatric anesthesia. *Pediatric Anesthesia, 22*(4), 400-404.

Bierbrier, R., Lo, V., & Wu, R. C. (2014). Evaluation of the accuracy of smartphone medical calculation apps. *Journal of medical Internet research, 16*(2).

BinDhim, N. F., Freeman, B., & Trevena, L. (2014). Pro-smoking apps for smartphones: the latest vehicle for the tobacco industry? *Tobacco control, 23*(1), e4-e4.

Bishop, J. (2013). mHealth Apps: A Guide to HIPAA, FDA Approvals, and Certifications. Retrieved from http://littlegreensoftware.com/mhealth-apps-hipaa-fda-approvals-certifications/

Bläsing, T., Batyuk, L., Schmidt, A.-D., Camtepe, S. A., & Albayrak, S. (2010). *An android application sandbox system for suspicious software detection.* Paper presented at the Malicious and unwanted software (MALWARE), 2010 5th international conference on.

Boulos, M. N., Wheeler, S., Tavares, C., & Jones, R. (2011). How smartphones are changing the face of mobile and participatory healthcare: an overview, with example from eCAALYX. *Biomedical engineering online, 10*(1), 24.

Boulos, M. N. K., Brewer, A. C., Karimkhani, C., Buller, D. B., & Dellavalle, R. P. (2014). Mobile medical and health apps: state of the art, concerns, regulatory control and certification. *Online journal of public health informatics, 5*(3), 229.

Breland, J. Y., Yeh, V. M., & Yu, J. (2013). Adherence to evidence-based guidelines among diabetes self-management apps. *Translational behavioral medicine, 3*(3), 277-286.

Brennan, P. F., Downs, S., & Casper, G. (2010). Project HealthDesign: Rethinking the power and potential of personal health records. *Journal of biomedical informatics, 43*(5), S3-S5.

Breton, E. R., Fuemmeler, B. F., & Abroms, L. C. (2011). Weight loss—there is an app for that! But does it adhere to evidence-informed practices? *Translational behavioral medicine, 1*(4), 523-529.

Brusco, J. M. (2010). Using smartphone applications in perioperative practice. *AORN journal, 92*(5), 503-508.

Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., & Sadeghi, A.-R. (2011). Xmandroid: A new android evolution to mitigate privilege escalation attacks. *Technische Universität Darmstadt, Technical Report TR-2011-04*.

Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A.-R., & Shastry, B. (2012). *Towards Taming Privilege-Escalation Attacks on Android.* Paper presented at the NDSS.

Bugiel, S., Davi, L., Dmitrienko, A., Heuser, S., Sadeghi, A.-R., & Shastry, B. (2011). *Practical and lightweight domain isolation on android.* Paper presented at the Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices.

Bugiel, S., Heuser, S., & Sadeghi, A.-R. (2012). Towards a framework for android security modules: Extending se android type enforcement to android middleware. *Intel Collaborative Research Institute for Secure Computing*.

Bugiel, S., Heuser, S., & Sadeghi, A.-R. (2013). *Flexible and fine-grained mandatory access control on Android for diverse security and privacy policies.* Paper presented at the Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13).

Burdette, S. D., Herchline, T. E., & Oehler, R. (2008). Practicing medicine in a technological age: using smartphones in clinical practice. *Clinical infectious diseases, 47*(1), 117-122.

Burdette, S. D., Trotman, R., & Cmar, J. (2012). Mobile infectious disease references: from the bedside to the beach. *Clinical infectious diseases, 55*(1), 114-125.

Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. (2011). *Crowdroid: behavior-based malware detection system for android.* Paper presented at the Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices.

Campbell, A., & Choudhury, T. (2012). From Smart to Cognitive Phones. *Pervasive Computing, IEEE, 11*(3), 7-11. doi:10.1109/MPRV.2012.41

Carrera, P. M., & Dalton, A. R. (2014). Do-it-yourself Healthcare: The current landscape, prospects and consequences. *Maturitas, 77*(1), 37-40.

Carter, M. C., Burley, V. J., Nykjaer, C., & Cade, J. E. (2013). Adherence to a smartphone application for weight loss compared to website and paper diary: pilot randomized controlled trial. *Journal of medical Internet research, 15*(4).

Carter, T., O'Neill, S., Johns, N., & Brady, R. R. (2013). Contemporary vascular smartphone medical applications. *Annals of vascular surgery, 27*(6), 804-809.

Chadwick, X., Loescher, L. J., Janda, M., & Soyer, H. P. (2014). *Mobile Medical Applications for Melanoma Risk Assessment: False Assurance or Valuable Tool?* Paper presented at the System Sciences (HICSS), 2014 47th Hawaii International Conference on.

Chakraborty, S., Shen, C., Raghavan, K. R., Shoukry, Y., Millar, M., & Srivastava, M. (2014). *ipShield: a framework for enforcing context-aware privacy.* Paper presented at the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14).

Chan, P. P., Hui, L. C., & Yiu, S.-M. (2012). *Droidchecker: analyzing android applications for capability leak.* Paper presented at the Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks.

Chen, Q. A., Qian, Z., & Mao, Z. M. (2014). *Peeking into your app without actually seeing it: UI state inference and novel android attacks.* Paper presented at the 23rd USENIX Security Symposium (USENIX Security 14).

Cheng, N. M., Chakrabarti, R., & Kam, J. K. (2014). iPhone Applications for Eye Care Professionals: A Review of Current Capabilities and Concerns. *Telemedicine and e-Health, 20*(4), 385-387.

Chhablani, J., Kaja, S., & Shah, V. A. (2012). Smartphones in ophthalmology. *Indian journal of ophthalmology, 60*(2), 127.

Chin, E., Felt, A. P., Greenwood, K., & Wagner, D. (2011). *Analyzing inter-application communication in Android.* Paper presented at the Proceedings of the 9th international conference on Mobile systems, applications, and services.

Cho, J., Park, D., & Lee, H. E. (2014). Cognitive Factors of Using Health Apps: Systematic Analysis of Relationships Among Health Consciousness, Health Information Orientation, eHealth Literacy, and Health App Use Efficacy. *Journal of medical Internet research, 16*(5).

Cho, M. J., Sim, J. L., & Hwang, S. Y. (2014). Development of Smartphone Educational Application for Patients with Coronary Artery Disease. *Healthcare informatics research, 20*(2), 117-124.

Choi, J., Noh, G.-Y., & Park, D.-J. (2014). Smoking Cessation Apps for Smartphones: Content Analysis With the Self-Determination Theory. *Journal of medical Internet research, 16*(2).

Cohn, A. M., Hunter-Reel, D., Hagman, B. T., & Mitchell, J. (2011). Promoting behavior change from alcohol use through mobile technology: the future of ecological momentary assessment. *Alcoholism: Clinical and Experimental Research, 35*(12), 2209-2215.

Connor, K., Brady, R., de Beaux, A., & Tulloh, B. (2013). Contemporary hernia smartphone applications (apps). *Hernia*, 1-5.

Conti, M., Nguyen, V. T. N., & Crispo, B. (2011). CRePE: Context-related policy enforcement for Android *Information Security* (pp. 331-345): Springer.

Crussell, J., Gibler, C., & Chen, H. (2012). *Attack of the clones: Detecting cloned applications on android markets.* Paper presented at the European Symposium on Research in Computer Security.

Crussell, J., Gibler, C., & Chen, H. (2013). *Andarwin: Scalable detection of semantically similar android applications.* Paper presented at the European Symposium on Research in Computer Security.

Dala-Ali, B. M., Lloyd, M. A., & Al-Abed, Y. (2011). The uses of the iPhone for surgeons. *The surgeon, 9*(1), 44-48.

Davi, L., Dmitrienko, A., Sadeghi, A.-R., & Winandy, M. (2011). Privilege escalation attacks on android *Information Security* (pp. 346-360): Springer.

Dayer, L., Heldenbrand, S., Anderson, P., Gubbins, P. O., & Martin, B. C. (2013). Smartphone medication adherence apps: potential benefits to patients and providers. *Journal of the American Pharmacists Association: JAPhA, 53*(2), 172.

Dehling, T., Gao, F., Schneider, S., & Sunyaev, A. (2015). Exploring the Far Side of Mobile Health: Information Security and Privacy of Mobile Health Apps on iOS and Android. *JMIR mHealth and uHealth, 3*(1), e8. doi:10.2196/mhealth.3672

Dennison, L., Morrison, L., Conway, G., & Yardley, L. (2013). Opportunities and challenges for smartphone applications in supporting health behavior change: qualitative study. *Journal of medical Internet research, 15*(4).

Derbyshire, E., & Dancey, D. (2013). Smartphone medical applications for women's health: what is the evidence-base and feedback? *International journal of telemedicine and applications, 2013*, 9.

Deveau, M., & Chilukuri, S. (2012). *Mobile applications for dermatology.* Paper presented at the Seminars in Cutaneous Medicine and Surgery.

Developers, A. (2016a). Application Fundamentals. Retrieved from http://developer. android.com/guide/components/fundamentals.html

Developers, A. (2016b). Intent. Retrieved from http://developer.android.com/reference/ android/content/Intent.html

Dietz, M., Shekhar, S., Pisetsky, Y., Shu, A., & Wallach, D. S. (2011). *QUIRE: Lightweight Provenance for Smart Phone Operating Systems.* Paper presented at the USENIX Security Symposium.

Distefano, A., Me, G., & Pace, F. (2010). Android anti-forensics through a local paradigm. *digital investigation, 7*, S83-S94.

Do, Q., Martini, B., & Choo, K.-K. R. (2014). *Enhancing user privacy on Android mobile devices via permissions removal.* Paper presented at the 2014 47th Hawaii International Conference on System Sciences.

Donker, T., Petrie, K., Proudfoot, J., Clarke, J., Birch, M.-R., & Christensen, H. (2013). Smartphones for smarter delivery of mental health programs: a systematic review. *Journal of medical Internet research, 15*(11).

Dubey, D., Amritphale, A., Sawhney, A., Amritphale, N., Dubey, P., & Pandey, A. (2014). Smart Phone Applications as a Source of Information on Stroke. *Journal of Stroke, 16*(2), 86-90.

Dunton, G. F., Dzubur, E., Kawabata, K., Yanez, B., Bo, B., & Intille, S. (2014). Development of a smartphone application to measure physical activity using sensor-assisted self-report. *Frontiers in public health, 2*.

Edlin, J. C., & Deshpande, R. P. (2013). Caveats of smartphone applications for the cardiothoracic trainee. *The Journal of thoracic and cardiovascular surgery, 146*(6), 1321-1326.

Egele, M., Kruegel, C., Kirda, E., & Vigna, G. (2011). *PiOS: Detecting Privacy Leaks in iOS Applications.* Paper presented at the NDSS.

Elgin, B. (2005). Google Buys Android for Its Mobile Arsenal. Retrieved from http:// tech-insider.org/mobile/research/2005/0817.html

Elias, B. L., Fogger, S. A., McGuinness, T. M., & D'Alessandro, K. R. (2014). Mobile apps for psychiatric nurses. *Journal of psychosocial nursing and mental health services, 52*(4), 42-47.

Elwood, D., Diamond, M. C., Heckman, J., Bonder, J. H., Beltran, J. E., Moroz, A., & Yip, J. (2011). Mobile Health: Exploring Attitudes Among Physical Medicine and Rehabilitation Physicians Toward this Emerging Element of Health Delivery. *PM&R, 3*(7), 678-680. doi:http://dx.doi.org/10.1016/j.pmrj.2011.05.004

Enck, W. (2011). *Defending users against smartphone apps: Techniques and future directions.* Paper presented at the International Conference on Information Systems Security.

Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L. P., . . . Sheth, A. N. (2014). TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS), 32*(2), 5.

Enck, W., Ongtang, M., & McDaniel, P. (2008). Mitigating Android software misuse before it happens.

Enck, W., Ongtang, M., & McDaniel, P. (2009). *On lightweight mobile phone application certification.* Paper presented at the Proceedings of the 16th ACM conference on Computer and communications security.

Eng, D. S., & Lee, J. M. (2013). The promise and peril of mobile health applications for diabetes and endocrinology. *Pediatric diabetes, 14*(4), 231-238.

Faruki, P., Ganmoor, V., Laxmi, V., Gaur, M. S., & Bharmal, A. (2013). *Androsimilar: robust statistical feature signature for android malware detection.* Paper presented at the Proceedings of the 6th International Conference on Security of Information and Networks.

Faudree, B., & Ford, M. (2013). Security and Privacy in Mobile Health. *CIO Journal*.

Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011). *Android permissions demystified.* Paper presented at the Proceedings of the 18th ACM conference on Computer and communications security.

Felt, A. P., Egelman, S., Finifter, M., Akhawe, D., & Wagner, D. (2012). *How to Ask for Permission.* Paper presented at the HotSec.

Felt, A. P., Finifter, M., Chin, E., Hanna, S., & Wagner, D. (2011). *A survey of mobile malware in the wild.* Paper presented at the Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices.

Felt, A. P., Greenwood, K., & Wagner, D. (2011). *The effectiveness of application permissions.* Paper presented at the Proceedings of the 2nd USENIX conference on Web application development.

Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., & Wagner, D. (2012). *Android permissions: User attention, comprehension, and behavior.* Paper presented at the Proceedings of the Eighth Symposium on Usable Privacy and Security.

Felt, A. P., Wang, H. J., Moshchuk, A., Hanna, S., & Chin, E. (2011). *Permission Re-Delegation: Attacks and Defenses.* Paper presented at the USENIX Security Symposium.

Fiordelli, M., Diviani, N., & Schulz, P. J. (2013). Mapping mHealth research: a decade of evolution. *Journal of medical Internet research, 15*(5).

Food, & Administration, D. (2015). Mobile medical applications: guidance for industry and Food and Drug Administration staff. *USA: Food and Drug Administration*, 1-44.

Fox, R., Cooley, J., McGrath, M., & Hauswirth, M. (2012). Mobile health apps - from singular to collaborative. *Stud Health Technol Inform, 177*, 158-163.

Fragkaki, E., Bauer, L., Jia, L., & Swasey, D. (2012). *Modeling and enhancing android's permission system.* Paper presented at the European Symposium on Research in Computer Security.

Franko, O. I. (2011). Smartphone apps for orthopaedic surgeons. *Clinical Orthopaedics and Related Research®, 469*(7), 2042-2048.

Franko, O. I. (2012). Mobile Software Applications for Hand Surgeons. *The Journal of hand surgery, 37*(6), 1273-1275.

Franko, O. I., Bray, C., & Newton, P. O. (2012). Validation of a scoliometer smartphone app to assess scoliosis. *Journal of Pediatric Orthopaedics, 32*(8), e72-e75.

Franko, O. I., & Tirrell, T. F. (2012). Smartphone app use among medical providers in ACGME training programs. *Journal of medical systems, 36*(5), 3135-3139.

French, D., & Casey, W. (2012). 2 Fuzzy Hashing Techniques in Applied Malware Analysis. *Results of SEI Line-Funded Exploratory New Starts Projects*, 2.

Fuchs, A. P., Chaudhuri, A., & Foster, J. S. (2009). Scandroid: Automated security certification of android.

Gay, V., & Leijdekkers, P. (2011). *The Good, The Bad and the Ugly about Social Networks for Health Apps.* Paper presented at the Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on.

Gay, V., & Leijdekkers, P. (2012). Personalised mobile health and fitness apps: lessons learned from myFitnessCompanion(R). *Stud Health Technol Inform, 177*, 248-253.

Gibler, C., Stevens, R., Crussell, J., Chen, H., Zang, H., & Choi, H. (2013). *Adrob: Examining the landscape and impact of android application plagiarism.* Paper presented at the Proceeding of the 11th annual international conference on Mobile systems, applications, and services.

Gilbert, P., Chun, B.-G., Cox, L. P., & Jung, J. (2011). *Vision: automated security validation of mobile apps at app markets.* Paper presented at the Proceedings of the second international workshop on Mobile cloud computing and services.

Gill, P. S., Kamath, A., & Gill, T. S. (2012). Distraction: an assessment of smartphone usage in health care work settings. *Risk management and healthcare policy, 5*, 105.

Glassenberg, R., De Oliveira, G., Glassenberg, S., & McCarthy, R. (2013). Teaching bronchoscopic intubation with an iPhone application: a randomized controlled trial. *Journal of Clinical Anesthesia, 25*(3), 248-249.

Goff, D. A. (2012). iPhones, iPads, and medical applications for antimicrobial stewardship. *Pharmacotherapy: The Journal of Human Pharmacology and Drug Therapy, 32*(7), 657-661.

Goldbach, H., Chang, A. Y., Kyer, A., Ketshogileng, D., Taylor, L., Chandra, A., . . . Fontelo, P. (2013). Evaluation of generic medical information accessed via mobile phones at the point of care in resource-limited settings. *Journal of the American Medical Informatics Association*, amiajnl-2012-001276.

Gomez-Iturriaga, A., Bilbao, P., Casquero, F., Cacicedo, J., & Crook, J. (2012). Smartphones and tablets: Reshaping radiation oncologists' lives. *Reports of Practical Oncology & Radiotherapy, 17*(5), 276-280.

Google. (2016). Google Play. Retrieved from https://play.google.com/store

Gorla, A., Tavecchia, I., Gross, F., & Zeller, A. (2014). *Checking app behavior against app descriptions.* Paper presented at the Proceedings of the 36th International Conference on Software Engineering.

Goyal, S., & Cafazzo, J. A. (2013). Mobile phone health apps for diabetes management: Current evidence and future developments. *QJM*, hct203.

Grace, M., Zhou, Y., Zhang, Q., Zou, S., & Jiang, X. (2012). *Riskranker: scalable and accurate zero-day android malware detection.* Paper presented at the Proceedings of the 10th international conference on Mobile systems, applications, and services.

Grace, M. C., Zhou, W., Jiang, X., & Sadeghi, A.-R. (2012). *Unsafe exposure analysis of mobile in-app advertisements.* Paper presented at the Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks.

Grace, M. C., Zhou, Y., Wang, Z., & Jiang, X. (2012). *Systematic Detection of Capability Leaks in Stock Android Smartphones.* Paper presented at the NDSS.

Gunasekera, S. (2012). *Android Apps Security*: Apress.

Haffey, F., Brady, R. R., & Maxwell, S. (2013). A comparison of the reliability of smartphone apps for opioid conversion. *Drug safety, 36*(2), 111-117.

Haffey, F., Brady, R. R., & Maxwell, S. (2014). Smartphone apps to support hospital prescribing and pharmacology education: a review of current provision. *British journal of clinical pharmacology, 77*(1), 31-38.

Hamilton, A., & Brady, R. (2012). Medical professional involvement in smartphone 'apps' in dermatology. *British Journal of Dermatology, 167*(1), 220-221.

Hamou, A., Guy, S., Lewden, B., Bilyea, A., Gwadry-Sridhar, F., & Bauer, M. (2010). *Data collection with iPhone Web apps efficiently collecting patient data using mobile devices.* Paper presented at the e-Health Networking Applications and Services (Healthcom), 2010 12th IEEE International Conference on.

Handel, M. J. (2011). mHealth (Mobile Health)—Using Apps for Health and Wellness. *EXPLORE: The Journal of Science and Healing, 7*(4), 256-261.

Hanna, S., Huang, L., Wu, E., Li, S., Chen, C., & Song, D. (2012). *Juxtapp: A scalable system for detecting code reuse among android applications.* Paper presented at the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.

Harada, T., Horie, T., & Tanaka, K. (2004). *Task oriented management obviates your onus on Linux.* Paper presented at the Linux Conference.

Hasegawa, S., Hasegawa, A., Takasu, K., Kojima, T., Miyao, M., Sugita, N., . . . Kato, K. (2013). *Multilingual medical dialog system developed as smartphone/tablet application.* Paper presented at the Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE.

Hawkes, C. P., Walsh, B. H., Ryan, C. A., & Dempsey, E. M. (2013). Smartphone technology enhances newborn intubation knowledge and performance amongst paediatric trainees. *Resuscitation, 84*(2), 223-226.

He, D. (2014a). *Security threats to Android apps.* Master's thesis, University of Illinois at Urbana-Champaign.

He, D. (2014b). Security threats to Android apps.

He, D., Naveed, M., Gunter, C. A., & Nahrstedt, K. (2014). *Security Concerns in Android mHealth Apps.* Paper presented at the American Medical Informatics Association (AMIA) Annual Symposium.

Hebden, L., Cook, A., van der Ploeg, H. P., & Allman-Farinelli, M. (2012). Development of smartphone applications for nutrition and physical activity behavior change. *JMIR Research Protocols, 1*(2).

Hilgefort, J., Fitzpatrick, S., Lycans, D., Wilson-Byrne, T., Fisher, C., & Shuler, F. (2013). Smartphone medical applications useful for the rural practitioner. *The West Virginia medical journal, 110*(1), 40-44.

Ho, C.-L., Fu, Y.-C., Lin, M.-C., Chan, S.-C., Hwang, B., & Jan, S.-L. (2014). Smartphone Applications (Apps) for Heart Rate Measurement in Children:

Comparison with Electrocardiography Monitor. *Pediatric cardiology, 35*(4), 726-731.

Hoog, A. (2011). *Android forensics: investigation, analysis and mobile security for Google Android*: Elsevier.

Hornyack, P., Han, S., Jung, J., Schechter, S., & Wetherall, D. (2011). *These aren't the droids you're looking for: retrofitting android to protect data from imperious applications.* Paper presented at the Proceedings of the 18th ACM conference on Computer and communications security.

Huckvale, K., Car, M., Morrison, C., & Car, J. (2012). Apps for asthma self-management: a systematic assessment of content and tools. *BMC medicine, 10*(1), 144.

International Data Corporation. (2016). Smartphone OS Market Share, Q2 2016. Retrieved from http://www.idc.com/prodserv/smartphone-os-market-share.jsp

Istepanian, R., Laxminarayan, S., & Pattichis, C. S. (2006). *M-health: Emerging Mobile Health Systems*: Springer.

Jahns, R.-G. (2014). The 8 drivers and barriers that will shape the mHealth app market in the next 5 years. Retrieved from http://mhealtheconomics.com/the-8-drivers-and-barriers-that-will-shape-the-mhealth-app-market-in-the-next-5-years/

Jana, S., & Shmatikov, V. (2012). *Memento: Learning secrets from process footprints.* Paper presented at the Security and Privacy (SP), 2012 IEEE Symposium on.

Jeon, J., Micinski, K. K., Vaughan, J. A., Fogel, A., Reddy, N., Foster, J. S., & Millstein, T. (2012). *Dr. Android and Mr. Hide: fine-grained permissions in android applications.* Paper presented at the Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices.

Jing, Y., Ahn, G.-J., Zhao, Z., & Hu, H. (2014). *Riskmon: Continuous and automated risk assessment of mobile applications.* Paper presented at the Proceedings of the 4th ACM Conference on Data and Application Security and Privacy.

Kalz, M., Lenssen, N., Felzen, M., Rossaint, R., Tabuenca, B., Specht, M., & Skorning, M. (2014). Smartphone Apps for Cardiopulmonary Resuscitation Training and Real Incident Support: A Mixed-Methods Evaluation Study. *Journal of medical Internet research, 16*(3).

Karami, M., Elsabagh, M., Najafiborazjani, P., & Stavrou, A. (2013). *Behavioral analysis of android applications using automated instrumentation.* Paper presented at the Software Security and Reliability-Companion (SERE-C), 2013 IEEE 7th International Conference on.

Karlsson, K.-J., & Glisson, W. B. (2014). *Android anti-forensics: Modifying cyanogenmod.* Paper presented at the 2014 47th Hawaii International Conference on System Sciences.

Kazi, D. S., Saha, P., & Mastey, N. (2014). PW266 Mobile Phones: Hope or Hype? A Qualitative Study of Best Practices in m-Health Development in a Low Income Country. *Global Heart, 9*(1), e312.

Kelley, P. G., Consolvo, S., Cranor, L. F., Jung, J., Sadeh, N., & Wetherall, D. (2012). *A conundrum of permissions: installing applications on an android smartphone.* Paper presented at the International Conference on Financial Cryptography and Data Security.

Kharrazi, H., Chisholm, R., VanNasdale, D., & Thompson, B. (2012). Mobile personal health records: an evaluation of features and functionality. *International journal of medical informatics, 81*(9), 579-593.

Khatoon, B., Hill, K., & Walmsley, A. (2013). Can we learn, teach and practise dentistry anywhere, anytime? *British dental journal, 215*(7), 345-347.

Kirwan, M., Duncan, M. J., Vandelanotte, C., & Mummery, W. K. (2013). Design, Development, and Formative Evaluation of a Smartphone Application for Recording and Monitoring Physical Activity Levels The 10,000 Steps "iStepLog". *Health Education & Behavior, 40*(2), 140-151.

Kotz, D. (2011). *A threat taxonomy for mHealth privacy.* Paper presented at the COMSNETS.

Kotz, D., Avancha, S., & Baxi, A. (2009). *A privacy framework for mobile health and home-care systems.* Paper presented at the Proceedings of the first ACM workshop on Security and privacy in medical and home-care systems.

Kraidin, J., Ginsberg, S. H., & Solina, A. (2012). Anesthesia apps: overview of current technology and intelligent search techniques. *Journal of cardiothoracic and vascular anesthesia, 26*(2), 322-326.

Kuhn, E., Eftekhari, A., Hoffman, J. E., Crowley, J. J., Ramsey, K. M., Reger, G. M., & Ruzek, J. I. (2014). Clinician Perceptions of Using a Smartphone App with Prolonged Exposure Therapy. *Administration and Policy in Mental Health and Mental Health Services Research*, 1-8.

La Polla, M., Martinelli, F., & Sgandurra, D. (2013). A survey on security for mobile devices. *IEEE communications surveys & tutorials, 15*(1), 446-471.

Lee, B., Lu, L., Wang, T., Kim, T., & Lee, W. (2014). *From zygote to morula: Fortifying weakened aslr on android.* Paper presented at the 2014 IEEE Symposium on Security and Privacy.

Lee, H., Sullivan, S. J., Schneiders, A. G., Ahmed, O. H., Balasundaram, A. P., Williams, D., . . . McCrory, P. (2014). Smartphone and tablet apps for concussion road warriors (team clinicians): a systematic review for practical users. *British journal of sports medicine*, bjsports-2013-092930.

Lessard, J., & Kessler, G. (2010). Android Forensics: Simplifying Cell Phone Examinations.

Lewis, T. (2013). Breast self-examination: A novel health promotion medium. *European Journal of Surgical Oncology (EJSO), 39*(5), 502. doi:http://dx.doi.org/10.1016/j.ejso.2013.01.182

Linares-Vásquez, M., Holtzhauer, A., Bernal-Cárdenas, C., & Poshyvanyk, D. (2014). *Revisiting android reuse studies in the context of code obfuscation and library usages.* Paper presented at the Proceedings of the 11th Working Conference on Mining Software Repositories.

Lineberry, A., Richardson, D. L., & Wyatt, T. (2010). These aren't the permissions you're looking for. *DefCon, 18*, 2010.

Lippman, H. (2013). How apps are changing family medicine. *J Fam Pract, 62*(7), 362-367.

Liu, C., Zhu, Q., Holroyd, K. A., & Seng, E. K. (2011). Status and trends of mobile-health applications for iOS devices: A developer's perspective. *Journal of Systems and Software, 84*(11), 2022-2033.

Loscocco, P., & Smalley, S. (2001). *Integrating flexible support for security policies into the Linux operating system.* Paper presented at the Proceedings of the FREENIX track: USENIX Annual Technical Conference.

Lu, L., Li, Z., Wu, Z., Lee, W., & Jiang, G. (2012). *Chex: statically vetting android apps for component hijacking vulnerabilities.* Paper presented at the Proceedings of the 2012 ACM conference on Computer and communications security.

Luo, T., Hao, H., Du, W., Wang, Y., & Yin, H. (2011). *Attacks on WebView in the Android system.* Paper presented at the Proceedings of the 27th Annual Computer Security Applications Conference.

Luxton, D. D., McCann, R. A., Bush, N. E., Mishkind, M. C., & Reger, G. M. (2011). mHealth for mental health: Integrating smartphone technology in behavioral healthcare. *Professional Psychology: Research and Practice, 42*(6), 505.

Maggi, F., Valdi, A., & Zanero, S. (2013). *AndroTotal: a flexible, scalable toolbox and service for testing mobile malware detectors.* Paper presented at the Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices.

Marforio, C., Francillon, A., & Capkun, S. (2011). *Application collusion attack on the permission-based security model and its implications for modern smartphone systems*: Department of Computer Science, ETH Zurich Zürich, Switzerland.

Marforio, C., Ritzdorf, H., Francillon, A., & Capkun, S. (2012). *Analysis of the communication between colluding applications on modern smartphones.* Paper presented at the Proceedings of the 28th Annual Computer Security Applications Conference.

McCarthy, M. (2013). Experts warn on data security in health and fitness apps. *BMJ, 347*, 1.

McCurdie, T., Taneva, S., Casselman, M., Yeung, M., McDaniel, C., Ho, W., & Cafazzo, J. (2012). mHealth consumer apps: the case for user-centered design. *Biomedical Instrumentation & Technology, 46*(s2), 49-56.

Mersini, P., Sakkopoulos, E., & Tsakalidis, A. (2013). *APPification of hospital healthcare and data management using QRcodes.* Paper presented at the Information, Intelligence, Systems and Applications (IISA), 2013 Fourth International Conference on.

Mertz, L. (2012). Ultrasound? Fetal monitoring? Spectrometer? There's an app for that!: biomedical smart phone apps are taking healthcare by storm. *Pulse, IEEE, 3*(2), 16-21.

Milani, P., Coccetta, C. A., Rabini, A., Sciarra, T., Massazza, G., & Ferriero, G. (2014). A Review of Mobile Smartphone Applications for Body Position Measurement in Rehabilitation: A Focus on Goniometric Tools. *PM&R*.

Min, Y. H., Lee, J. W., Shin, Y.-W., Jo, M.-W., Sohn, G., Lee, J.-H., . . . Ko, B. S. (2014). Daily Collection of Self-Reporting Sleep Disturbance Data via a Smartphone App in Breast Cancer Patients Receiving Chemotherapy: A Feasibility Study. *Journal of medical Internet research, 16*(5), e135.

Mirza, F., Norris, T., & Stockdale, R. (2008). Mobile technologies and the holistic management of chronic diseases. *Health informatics journal, 14*(4), 309-321.

Mitchell, S., Ridley, S., Tharenos, C., Varshney, U., Vetter, R., & Yaylacicegi, U. (2013). *Investigating Privacy and Security Challenges of mHealth Applications.* Paper presented at the Americas Conference on Information Systems (AMCIS), Chicago, Illinois, USA.

Mohan, A. T., & Branford, O. A. (2012). iGuide to Plastic Surgery iPhone Apps, the Plastic Surgeon, and the Health Care Environment. *Aesthetic Surgery Journal, 32*(5), 653-658.

Mollus, K., Westhoff, D., & Markmann, T. (2014). *Curtailing privilege escalation attacks over asynchronous channels on Android.* Paper presented at the Innovations for Community Services (I4CS), 2014 14th International Conference on.

Moodley, A., Mangino, J. E., & Goff, D. A. (2013). Review of infectious diseases applications for iPhone/iPad and Android: from pocket to patient. *Clinical infectious diseases, 57*(8), 1145-1154.

Moore, S., Anderson, J., & Cox, S. (2012). Pros and cons of using apps in clinical practice: Smartphones have the potential to enhance care but, say Sally Moore and colleagues, healthcare apps are not regulated, making it hard for nurse managers to be certain that those available are accurate, reliable and safe. *Nursing Management, 19*(6), 14-17.

Morris, R., Javed, M., Bodger, O., Gorse, S. H., & Williams, D. (2013). A comparison of two smartphone applications and the validation of smartphone applications as tools for fluid calculation for burns resuscitation. *Burns*.

Muessig, K. E., Pike, E. C., LeGrand, S., & Hightow-Weidman, L. B. (2013). Mobile phone applications for the care and prevention of HIV and other sexually transmitted diseases: a review. *Journal of medical Internet research, 15*(1).

Murthy, R., & Kotz, D. (2014). *Assessing blood-pressure measurement in tablet-based mHealth apps.* Paper presented at the Communication Systems and Networks (COMSNETS), 2014 Sixth International Conference on.

Nauman, M., Khan, S., & Zhang, X. (2010). *Apex: extending android permission model and enforcement with user-defined runtime constraints.* Paper presented at the Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security.

Naveed, M., Zhou, X., Demetriou, S., Wang, X., & Gunter, C. A. (2014). *Inside Job: Understanding and Mitigating the Threat of External Device Mis-Bonding on Android.* Paper presented at the Network and Distributed System Security (NDSS) Symposium.

Ni, X., Yang, Z., Bai, X., Champion, A. C., & Xuan, D. (2009). *DiffUser: Differentiated user access control on smartphones.* Paper presented at the 2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems.

Number of Android applications. (2015, 21 June, 2015). Retrieved from https://www.appbrain.com/stats/number-of-android-apps

Nwosu, A. C., & Mason, S. (2012). Palliative medicine and smartphones: an opportunity for innovation? *BMJ supportive & palliative care, 2*(1), 75-77.

O'Neill, K., Holmer, H., Greenberg, S., & Meara, J. (2013). Applying surgical apps: Smartphone and tablet apps prove useful in clinical practice. *Bulletin of the American College of Surgeons, 98*(11), 10-18.

O’Neill, S., & Brady, R. (2012). Colorectal smartphone apps: opportunities and risks. *Colorectal Disease, 14*(9), e530-e534.

O’Reilly, M., Nason, G., Liddy, S., Fitzgerald, C., Kelly, M., & Shields, C. (2013). DOCSS: doctors on-call smartphone study. *Irish journal of medical science*, 1-5.

Oehler, R. L., Smith, K., & Toney, J. F. (2010). Infectious diseases resources for the iPhone. *Clinical infectious diseases, 50*(9), 1268-1274.

Ongtang, M., Butler, K., & McDaniel, P. (2010). *Porscha: Policy oriented secure content handling in Android.* Paper presented at the Proceedings of the 26th Annual Computer Security Applications Conference.

Ongtang, M., McLaughlin, S., Enck, W., & McDaniel, P. (2012). Semantically rich application-centric security in Android. *Security and Communication Networks, 5*(6), 658-673.

Organization, W. H. (2011). *mHealth: New horizons for health through mobile technologies*: World Health Organization.

Orthacker, C., Teufl, P., Kraxberger, S., Lackner, G., Gissing, M., Marsalek, A., . . . Prevenhueber, O. (2011). *Android security permissions–can we trust them?* Paper presented at the International Conference on Security and Privacy in Mobile Information and Communication Systems.

Ozdalga, E., Ozdalga, A., & Ahuja, N. (2012). The smartphone in medicine: a review of current and potential use among physicians and students. *Journal of medical Internet research, 14*(5).

Pandey, A., Hasan, S., Dubey, D., & Sarangi, S. (2013). Smartphone apps as a source of cancer information: changing trends in health information-seeking behavior. *Journal of Cancer Education, 28*(1), 138-142.

Pandita, R., Xiao, X., Yang, W., Enck, W., & Xie, T. (2013). *WHYPER: Towards Automating Risk Assessment of Mobile Applications.* Paper presented at the USENIX Security.

Park, J.-Y., Lee, G., Shin, S.-Y., Kim, J. H., Han, H.-W., Kwon, T.-W., . . . Lee, J. H. (2014). Lessons Learned from the Development of Health Applications in a Tertiary Hospital. *Telemedicine and e-Health, 20*(3), 215-222.

Park, Y., & Chen, J. V. (2007). Acceptance and adoption of the innovative use of smartphone. *Industrial Management & Data Systems, 107*(9), 1349-1365.

Paschou, M., Sakkopoulos, E., & Tsakalidis, A. (2013). easyHealthApps: e-Health Apps Dynamic Generation for Smartphones & Tablets. *Journal of medical systems, 37*(3), 1-12.

Patel, V., Nowostawski, M., Thomson, G., Wilson, N., & Medlin, H. (2013). Developing a smartphone 'app'for public health research: the example of measuring observed smoking in vehicles. *Journal of epidemiology and community health, 67*(5), 446-452.

Payne, K. F. B., Wharrad, H., & Watts, K. (2012). Smartphone and medical related App use among medical students and junior doctors in the United Kingdom (UK): a regional survey. *BMC medical informatics and decision making, 12*(1), 121.

Pearce, P., Felt, A. P., Nunez, G., & Wagner, D. (2012). *Addroid: Privilege separation for applications and advertisers in android.* Paper presented at the Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security.

Peck, J. L., Stanton, M., & Reynolds, G. E. (2014). Smartphone preventive health care: Parental use of an immunization reminder system. *Journal of Pediatric Health Care, 28*(1), 35-42.

Pendragon Software Corporation. (1997). CaffeineMark 3.0 Information. Retrieved from http://www.benchmarkhq.ru/cm30/info.html

Plachkinova, M., Andrés, S., & Chatterjee, S. (2015). *A Taxonomy of mHealth Apps–Security and Privacy Concerns.* Paper presented at the Hawaii International Conference on System Sciences.

Poon, C. C., Zhang, Y.-T., & Bao, S.-D. (2006). A novel biometrics method to secure wireless body area sensor networks for telemedicine and m-health. *Communications Magazine, IEEE, 44*(4), 73-81.

Portokalidis, G., Homburg, P., Anagnostakis, K., & Bos, H. (2010). *Paranoid Android: versatile protection for smartphones.* Paper presented at the Proceedings of the 26th Annual Computer Security Applications Conference.

Project, A. O. S. (2016). Android Compatibility. Retrieved from http://source.android.com/compatibility/

Pulverman, R., & Yellowlees, P. M. (2014). Smart devices and a future of hybrid tobacco cessation programs. *Telemedicine and e-Health, 20*(3), 241-245.

Quick, D., & Alzaabi, M. (2011). Forensic analysis of the android file system yaffs2.

Rabin, C., & Bock, B. (2011). Desired features of smartphone applications promoting physical activity. *Telemedicine and e-Health, 17*(10), 801-803.

Ramachandran, A., & Pai, V. V. S. (2014, 5-7 March 2014). *Patient-centered mobile apps for chronic disease management.* Paper presented at the Computing for Sustainable Global Development (INDIACom), 2014 International Conference on.

Rashidi, B., & Fung, C. (2015). A Survey of Android Security Threats and Defenses. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 6*(3), 3-35.

Rasthofer, S., Arzt, S., & Bodden, E. (2014). *A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks.* Paper presented at the NDSS.

Rastogi, V., Chen, Y., & Jiang, X. (2013). *Droidchameleon: evaluating android anti-malware against transformation attacks.* Paper presented at the Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security.

Robinson, F., & Jones, C. (2014). Women's engagement with mobile device applications in pregnancy and childbirth. *Pract Midwife, 17*(1), 23-25.

Robustillo Cortés, M. d. l. A., Cantudo Cuenca, M. R., Morillo Verdugo, R., & Calvo Cidoncha, E. (2014). High Quantity but Limited Quality in Healthcare Applications Intended for HIV-Infected Patients. *Telemedicine and e-Health*.

Rosser, B., & Eccleston, C. (2011). The current state of healthcare apps for pain: A review of the functionality and validity of commercially available pain-related smartphone applications. *The Journal of Pain, 12*(4), P9.

Rosser, B. A., & Eccleston, C. (2011). Smartphone applications for pain management. *Journal of telemedicine and telecare, 17*(6), 308-312.

Rozenblyum, E. V., Mistry, N., Cellucci, T., Martimianakis, T., & Laxer, R. M. (2014). A144: Resident's Guide to Rheumatology Guide Mobile Application: An International Needs Assessment. *Arthritis & Rheumatology, 66*(S11), S187-S187.

Russello, G., Crispo, B., Fernandes, E., & Zhauniarovich, Y. (2011). *Yaase: Yet another android security extension.* Paper presented at the Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on.

Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P. G., & Álvarez, G. (2013). *Puma: Permission usage to detect malware in android.* Paper presented at the International Joint Conference CISIS'12-ICEUTE´ 12-SOCO´ 12 Special Sessions.

Sarma, B. P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C., & Molloy, I. (2012). *Android permissions: a perspective combining risks and benefits.* Paper presented at the Proceedings of the 17th ACM symposium on Access Control Models and Technologies.

Savic, M., Best, D., Rodda, S., & Lubman, D. I. (2013). Exploring the Focus and Experiences of Smartphone Applications for Addiction Recovery. *Journal of addictive diseases, 32*(3), 310-319.

Schlegel, R., Zhang, K., Zhou, X.-y., Intwala, M., Kapadia, A., & Wang, X. (2011). *Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones.* Paper presented at the NDSS.

Schmitt, S. (2011). *Mobile Phone Forensics: Analysis of the Android Filesystem (YAFFS2).* University of Mannheim.

Schreier, G., Schwarz, M., Modre-Osprian, R., Kastner, P., Scherr, D., & Fruhwald, F. (2013). *Design and evaluation of a multimodal mHealth based medication management system for patient self administration.* Paper presented at the Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE.

Server, S. (2007). Fuzzy Clarity: Using Fuzzy Hashing Techniques to Identify Malicious Code.

Shabtai, A., Fledel, Y., & Elovici, Y. (2010). Securing Android-powered mobile devices using SELinux. *IEEE security & privacy, 3*(8), 36-44.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., & Weiss, Y. (2012). "Andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems, 38*(1), 161-190.

Shand, F. L., Ridani, R., Tighe, J., & Christensen, H. (2013). The effectiveness of a suicide prevention app for indigenous Australian youths: study protocol for a randomized controlled trial. *Trials, 14*. doi:10.1186/1745-6215-14-396

Shebaro, B., Oluwatimi, O., & Bertino, E. (2015). Context-based access control systems for mobile devices. *IEEE Transactions on Dependable and Secure Computing, 12*(2), 150-163.

Shekhar, S., Dietz, M., & Wallach, D. S. (2012). *Adsplit: Separating smartphone advertising from applications.* Paper presented at the Presented as part of the 21st USENIX Security Symposium (USENIX Security 12).

Shen, F., Vishnubhotla, N., Todarka, C., Arora, M., Dhandapani, B., Lehner, E. J., . . . Ziarek, L. (2014). *Information flows as a permission mechanism.* Paper presented at the Proceedings of the 29th ACM/IEEE international conference on Automated software engineering.

Shin, W., Kiyomoto, S., Fukushima, K., & Tanaka, T. (2010). *A formal model to analyze the permission authorization and enforcement in the android framework.* Paper presented at the Social Computing (SocialCom), 2010 IEEE Second International Conference on.

Silow-Carroll, S., & Smith, B. (2013). Clinical management apps: creating partnerships between providers and patients. *Published November, 6*.

Silva, B. M., Rodrigues, J. J., Canelo, F., Lopes, I. C., & Zhou, L. (2013). A data encryption solution for mobile health apps in cooperation environments. *Journal of medical Internet research, 15*(4).

Simão, A. M. d. L., Sícoli, F. C., Melo, L. P. d., Deus, F. E. G. d., & Sousa Júnior, R. T. d. (2011). Acquisition and Analysis of Digital Evidencein Android Smartphones.

Singh, P. (2013). Orthodontic apps for smartphones. *Journal of orthodontics, 40*(3), 249-255.

Slaper, M. R., & Conkol, K. (2014). mHealth Tools for the Pediatric Patient-Centered Medical Home. *Pediatric annals, 43*(2), e39-43.

Smalley, S., & Craig, R. (2013). *Security Enhanced (SE) Android: Bringing Flexible MAC to Android.* Paper presented at the NDSS.

Smalley, S., Vance, C., & Salamon, W. (2001). Implementing SELinux as a Linux security module. *NAI Labs Report, 1*(43), 139.

Sondhi, V., & Devgan, A. (2013). Translating technology into patient care: Smartphone applications in pediatric health care. *medical journal armed forces india, 69*(2), 156-161.

Spain, C. H. J. (2014). A better regulation is required in viral hepatitis smartphone applications. *Farm Hosp, 38*(2), 112-117.

Spreitzenbarth, M. (2011). *Tools and Processes for Forensic Analyses of Smartphones and Mobile Malware.* Paper presented at the SPRING-SIDAR Graduierten-Workshop über Reaktive Sicherheit, 21.-22. März 2011, Bochum, Deutschland.

Stevens, D. J., Jackson, J. A., Howes, N., & Morgan, J. (2014). Obesity Surgery Smartphone Apps: a Review. *Obesity surgery, 24*(1), 32-36.

Stevens, R., Ganz, J., Filkov, V., Devanbu, P., & Chen, H. (2013). *Asking for (and about) permissions used by android apps.* Paper presented at the Proceedings of the 10th Working Conference on Mining Software Repositories.

Sun, M., & Tan, G. (2014). *NativeGuard: Protecting android applications from third-party native libraries.* Paper presented at the Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks.

Tan, D. J., Chua, T.-W., & Thing, V. L. (2015). Securing android: a survey, taxonomy, and challenges. *ACM Computing Surveys (CSUR), 47*(4), 58.

Techopedia. (2016). Definition - What does Mobile Application (Mobile App) mean? Retrieved from https://www.techopedia.com/definition/2953/mobile-application-mobile-app

Tripp, N., Hainey, K., Liu, A., Poulton, A., Peek, M., Kim, J., & Nanan, R. (2014). An emerging model of maternity care: Smartphone, midwife, doctor? *Women and Birth, 27*(1), 64-67.

Tseng, M.-H., Hsu, H.-C., Chang, C.-C., Ting, H., Wu, H.-C., & Tang, P.-H. (2012). *Development of an intelligent app for obstructive sleep apnea prediction on Android smartphone using data mining approach.* Paper presented at the Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2012 9th International Conference on.

Vidas, T., Christin, N., & Cranor, L. (2011). *Curbing android permission creep.* Paper presented at the Proceedings of the Web.

Vidas, T., Zhang, C., & Christin, N. (2011). Toward a general collection methodology for Android devices. *digital investigation, 8*, S14-S24.

Visvanathan, A., Hamilton, A., & Brady, R. (2012). Smartphone apps in microbiology—is better regulation required? *Clinical Microbiology and Infection, 18*(7), E218-E220.

Wackel, P., Beerman, L., West, L., & Arora, G. (2014). Tachycardia Detection Using Smartphone Applications in Pediatric Patients. *The Journal of pediatrics, 164*(5), 1133-1135.

Wallace, L., & Dhingra, L. (2013). A systematic review of smartphone applications for chronic pain available for download in the United States. *Journal of opioid management, 10*(1), 63-68.

Wang, J., Wang, Y., Wei, C., Yao, N., Yuan, A., Shan, Y., & Yuan, C. (2014). Smartphone Interventions for Long-Term Health Management of Chronic Diseases: An Integrative Review. *Telemedicine and e-Health.*

Warnock, G. L. (2012). The use of apps in surgery. *Canadian Journal of Surgery, 55*(2), 77.

Wearing, J. R., Nollen, N., Befort, C., Davis, A. M., & Agemy, C. K. (2014). iPhone App Adherence to Expert-Recommended Guidelines for Pediatric Obesity Prevention. *Childhood Obesity, 10*(2), 132-144.

Weichselbaum, L., Neugschwandtner, M., Lindorfer, M., Fratantonio, Y., van der Veen, V., & Platzer, C. (2014). Andrubis: Android malware under the magnifying glass. *Vienna University of Technology, Tech. Rep. TRISECLAB-0414, 1*, 5.

Workman, A. D., & Gupta, S. C. (2013). A plastic surgeon's guide to applying smartphone technology in patient care. *Aesthetic Surgery Journal*, 1090820X12472338.

Wright, C., Cowan, C., Smalley, S., Morris, J., & Kroah-Hartman, G. (2002). *Linux Security Modules: General Security Support for the Linux Kernel.* Paper presented at the USENIX Security Symposium.

Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M., & Wu, K.-P. (2012). *Droidmat: Android malware detection through manifest and api calls tracing.* Paper presented at the Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on.

Wu, H.-C., Chang, C.-J., Lin, C.-C., Tsai, M.-C., Chang, C.-C., & Tseng, M.-H. (2014). Developing Screening Services for Colorectal Cancer on Android Smartphones. *Telemedicine and e-Health*.

Wu, L., Grace, M., Zhou, Y., Wu, C., & Jiang, X. (2013). *The impact of vendor customizations on android security.* Paper presented at the Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security.

Xing, L., Pan, X., Wang, R., Yuan, K., & Wang, X. (2014). *Upgrading your android, elevating my malware: Privilege escalation through mobile os updating.* Paper presented at the 2014 IEEE Symposium on Security and Privacy.

Xu, C. S., Anderson, B., Armer, J., & Shyu, C.-R. (2012). *Improving disease management through a mobile application for lymphedema patients.* Paper presented at the e-Health Networking, Applications and Services (Healthcom), 2012 IEEE 14th International Conference on.

Xu, R., Saïdi, H., & Anderson, R. (2012). *Aurasium: Practical policy enforcement for android applications.* Paper presented at the Presented as part of the 21st USENIX Security Symposium (USENIX Security 12).

Yan, L. K., & Yin, H. (2012). *Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis.* Paper presented at the Presented as part of the 21st USENIX Security Symposium (USENIX Security 12).

Yang, Y. T., & Silverman, R. D. (2014). Mobile health applications: the patchwork of legal and liability issues suggests strategies to improve oversight. *Health Affairs, 33*(2), 222-227.

Yang, Z., & Yang, M. (2012). *Leakminer: Detect information leakage on android with static taint analysis.* Paper presented at the Software Engineering (WCSE), 2012 Third World Congress on.

Yang, Z., Yang, M., Zhang, Y., Gu, G., Ning, P., & Wang, X. S. (2013). *Appintent: Analyzing sensitive data transmission in android for privacy leakage detection.* Paper presented at the Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security.

Yoo, J.-H. (2013). The Meaning of Information Technology (IT) Mobile Devices to Me, the Infectious Disease Physician. *Infection & chemotherapy, 45*(2), 244-251.

Zanni, G. R. (2013). Medical apps worth having. *Consult Pharm, 28*(5), 322-324. doi:10.4140/TCP.n.2013.322

Zhang, X., Ahlawat, A., & Du, W. (2013). *AFrame: isolating advertisements from mobile applications in Android.* Paper presented at the Proceedings of the 29th Annual Computer Security Applications Conference.

Zhauniarovich, Y., Russello, G., Conti, M., Crispo, B., & Fernandes, E. (2014). MOSES: supporting and enforcing security profiles on smartphones. *IEEE Transactions on Dependable and Secure Computing, 11*(3), 211-223.

Zheng, C., Zhu, S., Dai, S., Gu, G., Gong, X., Han, X., & Zou, W. (2012). *Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications.* Paper presented at the Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices.

Zheng, M., Lee, P. P., & Lui, J. C. (2012). *ADAM: an automatic and extensible platform to stress test android anti-virus systems.* Paper presented at the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.

Zhou, W., Zhang, X., & Jiang, X. (2013). *AppInk: watermarking android apps for repackaging deterrence.* Paper presented at the Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security.

Zhou, W., Zhou, Y., Grace, M., Jiang, X., & Zou, S. (2013). *Fast, scalable detection of piggybacked mobile applications.* Paper presented at the Proceedings of the third ACM conference on Data and application security and privacy.

Zhou, W., Zhou, Y., Jiang, X., & Ning, P. (2012). *Detecting repackaged smartphone applications in third-party android marketplaces.* Paper presented at the Proceedings of the second ACM conference on Data and Application Security and Privacy.

Zhou, X., Demetriou, S., He, D., Naveed, M., Pan, X., Wang, X., . . . Nahrstedt, K. (2013). *Identity, location, disease and more: Inferring your secrets from android public resources.* Paper presented at the Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security.

Zhou, X., Lee, Y., Zhang, N., Naveed, M., & Wang, X. (2014). *The peril of fragmentation: Security hazards in android device driver customizations.* Paper presented at the 2014 IEEE Symposium on Security and Privacy.

Zhou, Y., & Jiang, X. (2012). *Dissecting android malware: Characterization and evolution.* Paper presented at the Security and Privacy (SP), 2012 IEEE Symposium on.

Zhou, Y., Wang, Z., Zhou, W., & Jiang, X. (2012). *Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets.* Paper presented at the NDSS.

Zhou, Y., Zhang, X., Jiang, X., & Freeh, V. W. (2011). Taming information-stealing smartphone applications (on android) *Trust and Trustworthy Computing* (pp. 93-107): Springer.

Zhu, Q., Liu, C., & Holroyd, K. A. (2012). *From a traditional behavioral management program to an m-health app: Lessons learned in developing m-health apps for existing health care programs.* Paper presented at the Software Engineering in Health Care (SEHC), 2012 4th International Workshop on.

LIST OF PUBLICATIONS

- Muzammil Hussain, Ahmed Al-Haiqi, Aws Alaa Zaidan, Bilal Bahaa Zaidan, Miss Laiha Mat Kiah, Nor Badrul Anuar, & Mohamed Abdulnabi, "The rise of keyloggers on smartphones: A survey and insight into motion-based tap inference attacks", Pervasive and Mobile Computing, Volume 25, January 2016, Pages 1-25, ISSN 1574-1192, (ISI Q1, Impact Factor 2.079).
- Muzammil Hussain, Ahmed Al-Haiqi, Aws Alaa Zaidan, Bilal Bahaa Zaidan, Miss Laiha Mat Kiah, Nor Badrul Anuar, & Mohamed Abdulnabi, "The landscape of research on smartphone medical apps: Coherent taxonomy, motivations, open challenges and recommendations" Computer methods and programs in biomedicine, Volume 122, December 2015, Pages 393-408, ISSN 0169-2607, (ISI Q1, Impact Factor 1.897).
- Muzammil Hussain, Miss Laiha Mat Kiah, Nor Badrul Anuar, & Ahmed Al-Haiqi, "A Security Framework for Mobile Health Applications" Journal of biomedical informatics, (Submitted), February 2017, ISSN 1532-0464, (ISI Q1, Impact Factor 2.447).
- Miss Laiha Mat Kiah, SH Al-Bakri, Aws Alaa Zaidan, Bilal Bahaa Zaidan, & Muzammil Hussain, "Design and develop a video conferencing framework for real-time telemedicine applications using secure group-based communication architecture" Journal of medical systems, Volume 38, October 2014, Pages 1-11, ISSN 0148-5598, (ISI Q2, Impact Factor 2.213).
- Salman Iqbal, Miss Laiha Mat Kiah, Babak Dhaghighi, Muzammil Hussain, Suleman Khan, Khan, Muhammad Khurram Khan, & Kim-Kwang Raymond Choo, "On cloud security attacks: A taxonomy and intrusion detection and prevention as a service", Journal of Network and Computer Applications,

Volume 74, October 2016, Pages 98-120, ISSN 1084-8045, (ISI Q1, Impact Factor 2.331).

- Mohamed Abdulnabi, Ahmed Al-Haiqi, MLM Kiah, A. A. Zaidan, BB. Zaidan, Muzammil Hussain, “A Distributed Framework for Health Information Exchange Using Smartphone Technologies” Journal of Biomedical Informatics, Volume 69, May 2017, Pages 230–250, ISSN 1532-0464, (ISI Q1, Impact Factor 2.447).
- Aws Alaa Zaidan, Bilal Bahaa Zaidan, Muzammil Hussain, Ahmed Al-Haiqi, Miss Laiha Mat Kiah, & Mohamed Abdulnabi, “Multi-criteria analysis for OS-EMR software selection problem: a comparative study” Decision Support Systems, Volume 78, October 2015, Pages 15-27, ISSN 0167-9236, (ISI Q1, Impact Factor 2.604).
- Aws Alaa Zaidan, Bilal Bahaa Zaidan, Ahmed Al-Haiqi, Miss Laiha Mat Kiah, Muzammil Hussain, & Mohamed Abdulnabi, “Evaluation and selection of open-source EMR software packages based on integrated AHP and TOPSIS” Journal of biomedical informatics, Volume 53, February 2015, Pages 390-404, ISSN 1532-0464, (ISI Q1, Impact Factor 2.482).
- Bilal Bahaa Zaidan, Ahmed Al-Haiqi, Aws Alaa Zaidan, Mohamed Abdulnabi, Miss Laiha Mat Kiah, & Muzammil Hussain, “A security framework for nationwide health information exchange based on telehealth strategy” Journal of medical systems, Volume 39, May 2015, Pages 1-19, ISSN 0148-5598, (ISI Q2, Impact Factor 2.213).

APPENDICES

APPENDIX A: SAMPLE SET OF APPLICATIONS

Table A-1: List of the Sample Set of Apps

#	Apps	URL
1	Blood Pressure Diary	https://play.google.com/store/apps/details?id=org.fruct.yar.bloodpressurediary
2	Ob (Pregnancy) Wheel	https://play.google.com/store/apps/details?id=com.quartertone.medcalc.obwheel
3	Contraction Timer	https://play.google.com/store/apps/details?id=com.ianhanniballake.contractiontimer
4	Baby Care	https://play.google.com/store/apps/details?id=com.kolsoft.babycare
5	Pregnancy Test & Symptom Quiz	https://play.google.com/store/apps/details?id=com.tsavo.amipregnant
6	Medical Calculators	https://play.google.com/store/apps/details?id=Pedcall.Calculator
7	BP (Blood Pressure) Diary	https://play.google.com/store/apps/details?id=kr.co.openit.bpdiary
8	Feed Baby - Tracker & Monitor	https://play.google.com/store/apps/details?id=au.com.penguinapps.android.babyfeeding.client.android
9	Baby Care - track baby growth!	https://play.google.com/store/apps/details?id=com.luckyxmobile.babycare
10	Pregnancy Due Date Calculator	https://play.google.com/store/apps/details?id=surebaby.pregnancy.calculator
11	My Menstrual Diary	https://play.google.com/store/apps/details?id=com.ecare.menstrualdiary
12	BMI Calculator (free)	https://play.google.com/store/apps/details?id=free.wk.mybodymass
13	Menstrual Calendar	https://play.google.com/store/apps/details?id=com.guillaumegranger.mc
14	My Ovulation Calculator	https://play.google.com/store/apps/details?id=com.ecare.ovulationcalculator
15	Pregnancy +	https://play.google.com/store/apps/details?id=com.hp.pregnancy.lite
16	Blood Pressure (My Heart)	https://play.google.com/store/apps/details?id=com.szyk.myheart
17	Migraine Buddy	https://play.google.com/store/apps/details?id=com.healint.migraineapp
18	Dosage Calc	https://play.google.com/store/apps/details?id=com.sekos.dosagecalc
19	Figure 1 - Medical Images	https://play.google.com/store/apps/details?id=com.figure1.android

Table A-1, Continued

#	Apps	URL
20	Ovia Pregnancy Guide	https://play.google.com/store/apps/details?id=com.ovuline.pregnancy
21	Blood Pressure	https://play.google.com/store/apps/details?id=com.freshware.bloodpressure
22	OnTrack Diabetes	https://play.google.com/store/apps/details?id=com.gexperts.ontrack
23	Glucose Buddy : Diabetes Log	https://play.google.com/store/apps/details?id=com.skyhealth.glucosebuddyfree
24	Pediatric OnCall	https://play.google.com/store/apps/details?id=com.pediatriconcall
25	PMCare+	https://play.google.com/store/apps/details?id=com.mosync.app_PMCare_Mobile
26	Weight Calorie Watch	https://play.google.com/store/apps/details?id=com.ecare.weightcaloriewatch
27	Ramsay Sime Darby Health Care	https://play.google.com/store/apps/details?id=com.lanewaysoftware.ourdoctors
28	Diabetes:M	https://play.google.com/store/apps/details?id=com.mydiabetes
29	FaceLift	https://play.google.com/store/apps/details?id=com.modiface.facelift.free
30	iMom • Pregnancy & Fertility	https://play.google.com/store/apps/details?id=com.obscience.imamma
31	Couple Counseling & Chatting	https://play.google.com/store/apps/details?id=com.abma.couplecounseling
32	BMI Calculator. Healthy Weight	https://play.google.com/store/apps/details?id=com.despdev.weight_loss_calculator
33	MoodTools - Depression Aid	https://play.google.com/store/apps/details?id=com.moodtools.moodtools
34	Spo2	https://play.google.com/store/apps/details?id=com.berry_med.spo2_bt
35	Lady Pill Reminder	https://play.google.com/store/apps/details?id=com.baviux.pillreminder
36	Health-Tracker	https://play.google.com/store/apps/details?id=com.benoved.phr_lite
37	Woman Calendar / Feminap	https://play.google.com/store/apps/details?id=com.kolsoft.feminap
38	HEART ATTACK ALERT SYSTEM	https://play.google.com/store/apps/details?id=com.ha.home
39	Menstrual Calendar	https://play.google.com/store/apps/details?id=com.indhay.menstrualcalendar
40	Ob Wheel Extra data	https://play.google.com/store/apps/details?id=com.quartertone.medcalc.obwheel.extras

Table A-1, Continued

#	Apps	URL
41	Blood Pressure - MyDiary	https://play.google.com/store/apps/details?id=com.zlamanit.blood.pressure
42	f-Ready	https://play.google.com/store/apps/details?id=com.sssllc.fready
43	Menstruation Fertility Pro Lte	https://play.google.com/store/apps/details?id=com.cbgsolutions.mfprotrial
44	Baby growth	https://play.google.com/store/apps/details?id=com.melunasoft.kampylesanaptykshs
45	Depression CBT Self-Help Guide	https://play.google.com/store/apps/details?id=com.excelatlife.depression
46	Mom 2 Be Pregnancy Tracker	https://play.google.com/store/apps/details?id=com.instanceone.android.mom2befree
47	Weight Diary	https://play.google.com/store/apps/details?id=org.fruct.yar.weightdiary
48	Breastfeeding	https://play.google.com/store/apps/details?id=com.whisperarts.kids.breastfeeding
49	Contraction Timer for Labour	https://play.google.com/store/apps/details?id=au.com.penguinapps.android.beautifulcontractiontimer.app
50	Framingham Risk Calculator	https://play.google.com/store/apps/details?id=com.calculaterx.framinghamriskcalculator
51	Baby Growth Apps FREE	https://play.google.com/store/apps/details?id=standard.android.app.BabyApps
52	Kidfolio Baby Tracker & Book	https://play.google.com/store/apps/details?id=com.alt12.kidfolio
53	GFR & BSA Calculator	https://play.google.com/store/apps/details?id=com.medcomis.device.android.egfr
54	Scanadu Scout	https://play.google.com/store/apps/details?id=com.scanadu.schs
55	Pilluling	https://play.google.com/store/apps/details?id=ru.pilluling.android
56	Doctor On Demand: MD & Therapy	https://play.google.com/store/apps/details?id=com.doctorondemand.android.patient
57	My Glycemia	https://play.google.com/store/apps/details?id=com.insyncapp.diabete
58	Diabetes Journal	https://play.google.com/store/apps/details?id=com.suderman.diabeteslog
59	Diabetes - Glucose Diary	https://play.google.com/store/apps/details?id=com.szyk.diabetes
60	Diabetes Plus	https://play.google.com/store/apps/details?id=com.squaremed.diabetesplus.typ1
61	Doctor Mole - Skin cancer app	https://play.google.com/store/apps/details?id=com.revsoft.doctormole

Table A-1, Continued

#	Apps	URL
62	Six-Min Walk Test	https://play.google.com/store/apps/details?id=com.stepic.sixminwt
63	Fluid & Electrolytes	https://play.google.com/store/apps/details?id=com.quartertone.medcalc
64	LabLink On The Go	https://play.google.com/store/apps/details?id=com.kpjlablink.lotg
65	Heart Rate Monitor	https://play.google.com/store/apps/details?id=com.mobmaxime.heartrate
66	MedTouch HD	https://play.google.com/store/apps/details?id=mr.ultrasound.istationpad
67	MedSight HD	https://play.google.com/store/apps/details?id=mr.ultrasound.medsightpad
68	Period Calendar / Tracker	https://play.google.com/store/apps/details?id=com.popularapp.periodcalendar
69	BMI Calculator - Weight Loss	https://play.google.com/store/apps/details?id=tools.bmirechner
70	Calorie Counter - MyFitnessPal	https://play.google.com/store/apps/details?id=com.myfitnesspal.android
71	Mi Fit	https://play.google.com/store/apps/details?id=com.xiaomi.hm.health
72	Instant Heart Rate	https://play.google.com/store/apps/details?id=si.modula.android.instantheartrate
73	Period Tracker	https://play.google.com/store/apps/details?id=com.period.tracker.lite
74	RunKeeper - GPS Track Run Walk	https://play.google.com/store/apps/details?id=com.fitnesskeeper.runkeeper.pro
75	Nike+ Running	https://play.google.com/store/apps/details?id=com.nike.plusgps
76	Monitor Your Weight	https://play.google.com/store/apps/details?id=monitoryourweight.bustan.net
77	Runtastic Running & Fitness	https://play.google.com/store/apps/details?id=com.runtastic.android
78	Noom Walk Pedometer: Fitness	https://play.google.com/store/apps/details?id=com.noom.walk
79	My Diet Coach - Weight Loss	https://play.google.com/store/apps/details?id=com.dietcoacher.sos
80	Google Fit	https://play.google.com/store/apps/details?id=com.google.android.apps.fitness
81	My Tracks	https://play.google.com/store/apps/details?id=com.google.android.maps.mytracks
82	Pedometer	https://play.google.com/store/apps/details?id=com.tayu.tau.pedometer
83	Ovulation & Period Calendar	https://play.google.com/store/apps/details?id=com.ladytimer.ovulationcalendar

Table A-1, Continued

#	Apps	URL
84	My Days - Period & Ovulation	https://play.google.com/store/apps/details?id=com.chris.mydays
85	Endomondo Running Cycling Walk	https://play.google.com/store/apps/details?id=com.endomondo.android
86	I'm Expecting - Pregnancy App	https://play.google.com/store/apps/details?id=org.medhelp.iamexpecting
87	My Cycles Period and Ovulation	https://play.google.com/store/apps/details?id=org.medhelp.mc
88	Strava Running and Cycling GPS	https://play.google.com/store/apps/details?id=com.strava
89	Runtastic Pedometer Step Count	https://play.google.com/store/apps/details?id=com.runtastic.android.pedometer.lite
90	Woman's DIARY period・diet・cal	https://play.google.com/store/apps/details?id=jp.kirei_r.sp.diary_free
91	LoveCycles Menstrual Calendar	https://play.google.com/store/apps/details?id=in.plackal.lovecyclesfree
92	Pedometer	https://play.google.com/store/apps/details?id=cc.pacer.androidapp
93	Runtastic Heart Rate Monitor	https://play.google.com/store/apps/details?id=com.runtastic.android.heartrate.lite
94	Garmin Connect™ Mobile	https://play.google.com/store/apps/details?id=com.garmin.android.apps.connectmobile
95	Pregnancy Tracker	https://play.google.com/store/apps/details?id=com.wte.view
96	My Diet Diary Calorie Counter	https://play.google.com/store/apps/details?id=org.medhelp.mydiet
97	Calorie Counter by FatSecret	https://play.google.com/store/apps/details?id=com.fatsecret.android
98	WomanLog Pregnancy Calendar	https://play.google.com/store/apps/details?id=com.proactiveapp.womanlogpregnancy.free
99	Cardiograph	https://play.google.com/store/apps/details?id=com.macropinch.hydra.android
100	Blood Pressure (BP) Watch	https://play.google.com/store/apps/details?id=com.boxeelab.healthlete.bpwatch

APPENDIX B: LIST OF SYSTEM INTERFACE LAYER FUNCTIONS

Listing B-1: A Complete List of System Interface Layer Functions Based on ASF (Backes, et al., 2014)

```
public interface IAccessControlModule {
/* *********************************************
* General functions
********************************************* */
public boolean init();
public ModuleConfiguration getConfig();
public void shutdown();

/* *********************************************
* Package life−cycle event hooks
********************************************* */
public void security_event_installNewPackage(PackageParser.Package pkg,
        UserHandle user);
public void security_event_replacePackage(PackageParser.Package oldPkg,
        PackageParser.Package newPkg, UserHandle user);
public void security_event_deletePackage(String packageName, int uid, int
        removedAppId, int removedUsers[], UserHandle user);

/* *********************************************
* Generic hooks
********************************************* */
public void security_generic_checkPolicy(Bundle arguments);
public void security_generic_callModule(Bundle arguments);
public boolean security_generic_instrumentApp(String packageName);

/* *********************************************
* Broadcast hooks
********************************************* */
public boolean security_broadcast_deliverToRegisteredReceiver(Intent intent,
        ComponentName targetComp, String requiredPermission, int targetUid,
        int targetPid, String callerPackage, ApplicationInfo callerApp, int
        callingUid, int callingPid);
public boolean security_broadcast_processNextBroadcast(Intent intent,
        ResolveInfo target, String requiredPermission, String callerPackage,
        ApplicationInfo callerApp, int callingUid, int callingPid);

/* *********************************************
* ContentProvider.Transport hooks
********************************************* */
public boolean security_cp_applyOperation(ContentProviderOperation op, int
        uid, int pid);
public boolean security_cp_preQuery(String callingPkg, Uri uri, String[]
        projection, String selection, String[] selectionArgs, String sortOrder, int
        uid, int pid);
public Cursor security_cp_postQuery(Cursor result, String callingPkg, Uri uri,
        String[] projection, String selection, String[] selectionArgs, String
        sortOrder, int uid, int pid);
public boolean security_cp_insert(Uri uri, ContentValues initialValues, int uid,
        int pid);
public boolean security_cp_bulkInsert(Uri uri, ContentValues[] initialValues,
        int uid, int pid);
public boolean security_cp_delete(String callingPkg, Uri uri, String selection,
        String[] selectionArgs, int uid, int pid);
public boolean security_cp_update(String callingPkg, Uri uri, ContentValues
        values, String selection, String[] selectionArgs, int uid, int pid);
public boolean security_cp_openFile(Uri uri, String mode, int uid, int pid);
public boolean security_cp_preCall(String providerClass, String method, String
        arg, Bundle extras, int uid, int pid);
public Bundle security_cp_postCall(Bundle result, String providerClass, String
        method, String arg, Bundle extras, int uid, int pid);

public boolean security_contacts_preQueryDirectory(Uri uri, String
        directoryName, String directoryType, String[] projection, String
        selection, String[] selectionArgs, String sortOrder, int uid, int pid);
public BulkCursorDescriptor
        security_contacts_postQueryDirectory(BulkCursorDescriptor result,
        String directoryName, String directoryType, String providerName, Uri
        uri, String[] projection, String selection, String[] selectionArgs, String
        sortOrder, int uid, int pid);

/* **********************************************
* Activity related hooks
********************************************** */
public boolean security_ams_startActivity(Intent intent, String resolvedType,
        ActivityInfo aInfo, String resultWho, int requestCode, int startFlags,
        Bundle options, ApplicationInfo callerInfo, int callingPid, int callingUid,
        int callingUserId);
public boolean security_ams_finishActivity(ComponentName origActivity,
        ComponentName realActivity, Intent intent, int userId, ApplicationInfo
        info, int resultCode, Intent resultData, int uid, int pid);
public boolean security_ams_moveTaskToFront(ComponentName
        origActivity, ComponentName realActivity, Intent intent, int userId,
        ApplicationInfo info, int flags, Bundle options, int uid, int pid);
public boolean security_ams_moveTaskToBack(ComponentName origActivity,
        ComponentName realActivity, Intent intent, int userId, ApplicationInfo
        info, int uid, int pid);
public boolean security_ams_clearApplicationUserData(String packageName,
        int pkgUid, int userId, int uid, int pid);

/* **********************************************
* Permission check overrides
********************************************** */
public int security_ams_checkComponentPermission(String permission, int
        origUid, int origPid, int tlsUid, int tlsPid, int owningUid, boolean
        exported, int callerUid, int callerPid);
public boolean security_ams_checkCPUriPermission(Uri uri, ProviderInfo cpi,
        int processUid, int processPid, boolean procesIsolated, int
        processUserId, String processName, ApplicationInfo info, int uid, int
        pid);
public boolean security_ams_checkCPUriPermission(Uri uri, ProviderInfo cpi,
        int uid, int pid);
public boolean security_ams_checkGrantUriPermission(int callingUid, String
        targetPkg, int targetUid, Uri uri, int modeFlags);
public int security_ams_checkUriPermission(Uri uri, int origUid, int origPid,
        int tlsUid, int tlsPid, int modeFlags);

/* *********************************************
* PackageManagerService hooks
********************************************* */
public boolean security_pms_getPackageInfo(PackageInfo pi, int flags, int
        userId, boolean isUninstalled, int uid, int pid);
public boolean security_pms_getPackageUid(ApplicationInfo info, int userId,
        int uid, int pid);
public boolean security_pms_getPackageGids(ApplicationInfo info, int[] gids,
        int uid, int pid);
public String[] security_pms_getPackagesForUid(int forUid, String[] packages,
        int uid, int pid);
public boolean security_pms_getNameForUid(int forUid, String name, int uid,
        int pid);
public boolean security_pms_getUidForSharedUser(String sharedUserName,
        int suid, int uid, int pid);
public boolean security_pms_findPreferredActivity(Intent intent, String
        resolvedType, int flags, ResolveInfo ri, int priority, int userId, int uid,
        int pid);
public List<ResolveInfo>
        security_pms_queryIntentActivities(List<ResolveInfo> currentList,
        Intent intent, String resolvedType, int flags, int userId, int uid, int pid);
public List<ResolveInfo>
        security_pms_queryIntentReceivers(List<ResolveInfo> currentList,
        Intent intent, String resolvedType, int flags, int userId, int uid, int pid);
public List<ResolveInfo> security_pms_queryIntentServices(List<ResolveInfo>
        currentList, Intent intent, String resolvedType, int flags, int userId, int
        uid, int pid);
public ArrayList<PackageInfo>
        security_pms_getInstalledPackages(ArrayList<PackageInfo> currentList,
        int flags, int userId, int uid, int pid);
public ArrayList<PackageInfo>
        security_pms_getPackagesHoldingPermissions(ArrayList<PackageInfo>
        currentList, int flags, int userId, String[] permissions, int uid, int pid);
public ArrayList<ApplicationInfo>
        security_pms_getInstalledApplications(ArrayList<ApplicationInfo>
        currentList, int flags, int userId, int uid, int pid);
public ArrayList<ApplicationInfo>
        security_pms_getPersistentApplications(ArrayList<ApplicationInfo>
        currentList, int flags, int uid, int pid);
public boolean security_pms_getProviderInfo(ProviderInfo pi,
        ComponentName component, int flags, int userId, int uid, int pid);
public boolean security_pms_getActivityInfo(ActivityInfo ai, ComponentName
        component, int flags, int userId, int uid, int pid);
public boolean security_pms_getReceiverInfo(ActivityInfo ai,
        ComponentName component, int flags, int userId, int uid, int pid);
public boolean security_pms_getServiceInfo(ServiceInfo si, ComponentName
        component, int flags, int userId, int uid, int pid);
/∗ Pre−init function (packages are scanned before init is called) ∗/
public boolean security_pms_scanPackage(PackageParser.Package pkg);
public boolean security_pms_deletePackage(PackageParser.Package pkg,
        boolean isSystemApp, boolean dataOnly, int flags);
public boolean security_pms_deletePackageSingleUser(PackageParser.Package
        pkg, boolean isSystemApp, int flags, int user);

/∗ ∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗
∗ Content Provider (general) related hooks
∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗ ∗/
/∗ Changed ProcessRecord to public for our module SDK ∗/
public boolean security_ams_checkContentProviderPermission(ProviderInfo
        cpi, String permission, int processUid, int processPid, boolean
        procesIsolated, int processUserId, String processName, ApplicationInfo
        info, int uid, int pid);
public boolean security_ams_checkContentProviderPermission(ProviderInfo
        cpi, String permission, int uid, int pid);
public boolean security_ams_checkPathPermission(ProviderInfo cpi,
        PathPermission pp, String permission, int processUid, int processPid,
        boolean procesIsolated, int processUserId, String processName,
        ApplicationInfo info, int uid, int pid);
public boolean security_ams_checkPathPermission(ProviderInfo cpi,
        PathPermission pp, String permission, int uid, int pid);
public boolean security_ams_checkAppSwitchAllowed(int uid, int pid);

/∗ ∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗
∗ Service related hooks
∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗ ∗/
public List<ActivityManager.RunningServiceInfo> security_ams_getServices
            (ArrayList<ActivityManager.RunningServiceInfo> srvList, int uid,
        int pid);
public boolean security_ams_peekService(Intent service, String resolvedType,
        ServiceInfo serviceInfo, ApplicationInfo appInfo, String packageName,
        String permission, int uid, int pid);
public boolean security_ams_startService(Intent service, String resolvedType,
        ComponentName name, String shortName, ServiceInfo serviceInfo,
        ApplicationInfo appInfo, int srvUserId, String packageName, String
        processName, String permission, int callingPid, int callingUid);
public boolean security_ams_stopService(Intent service, String resolvedType,
        ComponentName name, String shortName, ServiceInfo serviceInfo,
        ApplicationInfo appInfo, int srvUserId, String packageName, String
        processName, String permission, int callingPid, int callingUid);
public boolean security_ams_bindService(Intent service, String resolvedType,
        int flags, ComponentName name, String shortName, ServiceInfo
        serviceInfo, ApplicationInfo appInfo, int srvUserId, String
        packageName, String processName, String permission, int callingPid, int
        callingUid);

/∗ ∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗
```

```
* LocationManagerService hooks
********************************************** */
public void security_location_getAllProviders(List<String> providerList, int
        uid, int pid);
public void security_location_getProviders(List<String> providers, Criteria
        criteria, boolean enabledOnly, int uid, int pid);

/* Unhide LocationRequest for our module SDK */
public void security_location_requestLocationUpdates(LocationRequest
        request, PendingIntent pi, int uid, int pid);
public void security_location_removeLocationUpdates(PendingIntent pi, int
        uid, int pid);
public Location security_location_getLastLocation(Location currentLocation,
        LocationRequest request, int uid, int pid);
public boolean security_location_addGpsStatusListener(int uid, int pid);
public boolean security_location_sendExtraCommand(String provider, String
        command, Bundle extras, int uid, int pid);
 /* Unhide Geofence class for our module SDK */
public void security_location_requestGeofence(LocationRequest request,
        Geofence geofence, PendingIntent intent, int uid, int pid);
public void security_location_removeGeofence(Geofence geofence,
        PendingIntent  intent, int uid, int pid);
public boolean security_location_isProviderEnabled(String provider, int uid,
        int pid);
public Location security_location_reportLocation(Location location, boolean
        passive, int uid, int pid);
public ProviderProperties security_location_addTestProvider(String name,
        ProviderProperties properties, int uid, int pid);
public boolean security_location_removeTestProvider(String provider, int uid,
        int pid);
public boolean security_location_setTestProviderLocation(String provider,
        Location location, int uid, int pid);
public boolean security_location_clearTestProviderLocation(String provider,
        int uid, int pid);
public boolean security_location_setTestProviderEnabled(String provider,
        boolean enabled, int uid, int pid);
public boolean security_location_clearTestProviderEnabled(String provider, int
        uid, int pid);
public boolean security_location_setTestProviderStatus(String provider, int
        status, Bundle extras, long updateTime, int uid, int pid);
public boolean security_location_clearTestProviderStatus(String provider, int
        uid, int pid);
public boolean security_location_sendLocationUpdate(Location location, String
        receiverPackageName, int pid, int uid);
public boolean security_location_updateFence(Location location, Geofence
        fence, PendingIntent fenceIntent, String fencePackageName, int uid);

/* **********************************************
* AudioService hooks
********************************************** */
public boolean security_audio_adjustStreamVolume(int streamType, int
          direction, int flags, int uid, int pid);
public boolean security_audio_setStreamVolume(int streamType, int index, int
          flags, int uid, int pid);
public boolean security_audio_setMasterVolume(int volume, int flags, int uid,
          int pid);
public boolean security_audio_setRingerMode(int mode, int uid, int pid);
public boolean security_audio_setSpeakerphoneOn(boolean on, int uid, int
          pid);

/* **********************************************
* TelephonyService hooks
********************************************** */
public boolean security_telephony_call(String number, int uid, int pid);
public List<NeighboringCellInfo>
          security_telephony_getNeighboringCellInfo(List<NeighboringCellInfo>
          currentList, int uid, int pid);

/* **********************************************
* SMS and MMS Service hooks
********************************************** */
public boolean security_sms_copyMessageToIcc(int status, byte[] pdu, byte[]
          smsc, int uid, int pid);
public boolean security_sms_getAllMessagesFromIcc(int uid, int pid);
public List<RawByteData>
          security_sms_getAllMessagesFromIccFilter(List<RawByteData>
          rawSms, int uid, int pid);
public boolean security_sms_sendData(String destAddr, String scAddr, int
          destPort, byte[] data, PendingIntent sentIntent, PendingIntent
          deliveryIntent, int uid, int pid);
public boolean security_sms_sendText(String destAddr, String scAddr, String
          text, PendingIntent sentIntent, PendingIntent deliveryIntent, int uid, int
          pid);
public boolean security_sms_sendMultipartText(String destAddr, String
          scAddr, List<String> parts, List<PendingIntent> sentIntents,
          List<PendingIntent> deliveryIntents, int uid, int pid);
public boolean security_sms_updateMessageOnIccEf(int index, int status,
          byte[] pdu, int uid, int pid);

/* **********************************************
* WiFi Service hooks
********************************************** */
public List<ScanResult> security_wifi_getScanResult(List<ScanResult> result,
          int uid, int pid);
public boolean security_wifi_startScan(int uid, int pid);
public boolean security_wifi_setWifiEnabled(boolean enable, int uid, int pid);
public boolean security_wifi_setWifiApEnabled(WifiConfiguration wifiConfig,
          boolean enabled, int uid, int pid);
public boolean security_wifi_setWifiApConfiguration(WifiConfiguration
          wifiConfig, int uid, int pid);
public boolean security_wifi_disconnect(int uid, int pid);
public boolean security_wifi_reconnect(int uid, int pid);
public boolean security_wifi_reassociate(int uid, int pid);
public List<WifiConfiguration>
        security_wifi_getConfiguredNetworks(List<WifiConfiguration>
        currentList, int uid, int pid);
public boolean security_wifi_addOrUpdateNetwork(WifiConfiguration config,
        int uid, int pid);
public boolean security_wifi_removeNetwork(int netId, int uid, int pid);
public boolean security_wifi_enableNetwork(int netId, boolean disableOthers,
        int uid, int pid);
public boolean security_wifi_disableNetwork(int netId, int uid, int pid);
public boolean security_wifi_getConnectionInfo(WifiInfo info, int uid, int pid);
public boolean security_wifi_setCountryCode(String countryCode, boolean
        persist, int uid, int pid);
public boolean security_wifi_setFrequencyBand(int band, boolean persist, int
        uid, int pid);
public boolean security_wifi_startWifi(int uid, int pid);
public boolean security_wifi_stopWifi(int uid, int pid);
public boolean security_wifi_addToBlacklist(String bssid, int uid, int pid);
public boolean security_wifi_clearBlacklist(int uid, int pid);
public boolean security_wifi_getWifiServiceMessenger(int uid, int pid);
public boolean security_wifi_getWifiStateMachineMessenger(int uid, int pid);
public boolean security_wifi_getConfigFile(String currentConfig, int uid, int
        pid);

/* **********************************************
* ClipboardService hooks
********************************************** */
public ClipData security_clip_getPrimaryClip(ClipData currentPrimary, int
        clipUid, int uid, int pid);
public boolean security_clip_setPrimaryClip(ClipData clip, int uid, int pid);
public boolean security_clip_informPrimaryClipChanged(ClipData
        currentPrimary, int setByUid, String packageName, int uid);
public ClipDescription
        security_clip_getPrimaryClipDescription(ClipDescription
        currentDescription, int clipUid, int uid, int pid);
public boolean security_clip_hasPrimaryClip(boolean hasClipboard, int
        clipUid, int uid, int pid);
public boolean security_clip_hasClipboardText(String currentText, int clipUid,
        int uid, int pid);

/* **********************************************
* PowerManagerService hooks
********************************************** */
public boolean security_power_acquireWakeLock(String tag, WorkSource ws,
        int uid, int pid);
public boolean security_power_userActivity(long eventTime, int event, int
        flags, int uid, int pid);
public boolean security_power_goToSleep(long eventTime, int reason, int uid,
        int pid);
public boolean security_power_wakeUp(long eventTime, int uid, int pid);
public boolean security_power_nap(long time, int uid, int pid);
public boolean security_power_setBacklightBrightness(int brightness, int uid,
int pid);
public boolean security_power_reboot(boolean confirm, String reason, boolean wait, int uid, int pid);

/∗ ∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗
∗ PhoneSubscriberInfo hooks
∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗ ∗/
public String security_phonesubinfo_getDeviceId(String id, int uid, int pid);
public String security_phonesubinfo_getDeviceSvn(String svn, int uid, int pid);
public String security_phonesubinfo_getSubscriberId(String id, int uid, int pid);
public String security_phonesubinfo_getGroupIdLevel1(String groupid, int uid, int pid);
public String security_phonesubinfo_getIccSerialNumber(String icc, int uid, int pid);
public String security_phonesubinfo_getLine1Number(String number, int uid, int pid);
public String security_phonesubinfo_getLine1AlphaTag(String tag, int uid, int pid);
public String security_phonesubinfo_getMsisdn(String msisdn, int uid, int pid);
public String security_phonesubinfo_getVoiceMailNumber(String number, int uid, int pid);
public String security_phonesubinfo_getVoiceMailAphaTag(String tag, int uid, int pid);
public String security_phonesubinfo_getIsimImpi(String impi, int uid, int pid);
public String security_phonesubinfo_getIsimDomain(String domain, int uid, int pid);
public String[] security_phonesubinfo_getIsimImpu(String impu[], int uid, int pid);
}
```

Interface for Access Control Policy Modules to Linux Security Module

```
public interface KMACAdaptor {
public boolean init();
public boolean isEnabled();
public boolean isEnforcing();
public boolean setEnforcing(boolean value);
public boolean setContext(String path, Bundle context);
public boolean restoreContext(Bundle context);
public Bundle getContext(String path);
public Bundle getPeerContext(FileDescriptor fd); /∗ wrapper around getsockopt call to LSM ∗/
public Bundle getCurrentContext();
public Bundle getProcessContext(int pid);
public Bundle getConfig(Bundle args); /∗ e.g., get list of defined booleans or one specific boolean value ∗/
public boolean setConfig(Bundle conf); /∗ e.g., set a boolean value ∗/
public boolean checkAccess(Bundle args); /∗ args can be, e.g., quadruple of subject ctx, object ctx, object class, op ∗/

/∗ Zygote is statically integrated with the Kernel MAC, thus, each
```

```
        KMACAdaptor must implemented these hooks in ZygoteConnection */
public boolean security_zygote_applyUidSecurityPolicy(Credentials creds,
        Bundle peerSecurityContext);
public boolean security_zygote_applyRlimitSecurityPolicy(Credentials creds,
        Bundle peerSecurityContext);
public boolean security_zygote_applyCapabilitiesSecurityPolicy(Credentials
        creds, Bundle peerSecurityContext);
public boolean security_zygote_applyInvokeWithSecurityPolicy(Credentials
        creds, Bundle peerSecurityContext);
public boolean security_zygote_applySecurityLabelPolicy(Credentials creds,
        Bundle peerSecurityContext);
}
```

Methods for IRM Instrumentation

```
public class Instrumentation {
public static void initClass(Class<?> clazz);

public static int redirectMethod(String fromDescriptor, String toDescriptor);
public static int redirectMethod(Signature from, Signature to);

public static void callVoidMethod(Class<?> caller, Object _this, Object... args);
public static void callVoidMethod(String id, Object _this, Object... args);
public static void callVoidMethod(int methodId, Object _this, Object... args);
public static int callIntMethod(Class<?> caller, Object _this, Object... args);
public static int callIntMethod(String id, Object _this, Object... args);
public static int callIntMethod(int methodId, Object _this, Object... args);
public static boolean callBooleanMethod(Class<?> caller, Object _this,
        Object... args);
public static boolean callBooleanMethod(String id, Object _this, Object...
        args);
public static boolean callBooleanMethod(int methodId, Object _this, Object...
        args);
public static Object callObjectMethod(Class<?> caller, Object _this, Object...
        args);
public static Object callObjectMethod(String id, Object _this, Object... args);
public static Object callObjectMethod(int methodId, Object _this, Object...
        args);
public static void callStaticVoidMethod(Class<?> caller, Class<?> _clazz,
        Object... args);
public static void callStaticVoidMethod(String id, Class<?> _clazz, Object...
        args);
public static void callStaticVoidMethod(int methodId, Class<?> _clazz,
        Object... args);
public static Object callStaticObjectMethod(Class<?> caller, Class<?> _clazz,
        Object... args);
public static Object callStaticObjectMethod(String id, Class<?> _clazz,
        Object... args);
public static Object callStaticObjectMethod(int methodId, Class<?> _clazz,
        Object... args);
}
```